Linux Userspace-api Documentation

The kernel development community

Jun 10, 2024

CONTENTS

While much of the kernel's user-space API is documented elsewhere (particularly in the man-pages project), some user-space information can also be found in the kernel tree itself. This manual is intended to be the place where this information is gathered.

Table of contents

CHAPTER ONE

NO NEW PRIVILEGES FLAG

The execve system call can grant a newly-started program privileges that its parent did not have. The most obvious examples are setuid/setgid programs and file capabilities. To prevent the parent program from gaining these privileges as well, the kernel and user code must be careful to prevent the parent from doing anything that could subvert the child. For example:

- The dynamic loader handles `LD_*` environment variables differently if a program is setuid.
- chroot is disallowed to unprivileged processes, since it would allow `/etc/passwd` to be replaced from the point of view of a process that inherited chroot.
- The exec code has special handling for ptrace.

These are all ad-hoc fixes. The `no_new_privs` bit (since Linux 3.5) is a new, generic mechanism to make it safe for a process to modify its execution environment in a manner that persists across execve. Any task can set `no_new_privs`. Once the bit is set, it is inherited across fork, clone, and execve and cannot be unset. With `no_new_privs` set, `execve()` promises not to grant the privilege to do anything that could not have been done without the execve call. For example, the setuid and setgid bits will no longer change the uid or gid; file capabilities will not add to the permitted set, and LSMs will not relax constraints after execve.

To set `no_new_privs`, use:

```
prctl(PR_SET_NO_NEW_PRIVS, 1, 0, 0, 0);
```

Be careful, though: LSMs might also not tighten constraints on exec in `no_new_privs` mode. (This means that setting up a general-purpose service launcher to set `no_new_privs` before execing daemons may interfere with LSM-based sandboxing.)

Note that `no_new_privs` does not prevent privilege changes that do not involve `execve()`. An appropriately privileged task can still call `setuid(2)` and receive SCM_RIGHTS datagrams.

There are two main use cases for `no_new_privs` so far:

- Filters installed for the seccomp mode 2 sandbox persist across execve and can change the behavior of newly-executed programs. Unprivileged users are therefore only allowed to install such filters if `no_new_privs` is set.

- By itself, no_new_privs can be used to reduce the attack surface available to an unprivileged user. If everything running with a given uid has no_new_privs set, then that uid will be unable to escalate its privileges by directly attacking setuid, setgid, and fcap-using binaries; it will need to compromise something without the no_new_privs bit set first.

In the future, other potentially dangerous kernel features could become available to unprivileged tasks if no_new_privs is set. In principle, several options to unshare(2) and clone(2) would be safe when no_new_privs is set, and no_new_privs + chroot is considerable less dangerous than chroot by itself.

CHAPTER

TWO

SECCOMP BPF (SECURE COMPUTING WITH FILTERS)

2.1 Introduction

A large number of system calls are exposed to every userland process with many of them going unused for the entire lifetime of the process. As system calls change and mature, bugs are found and eradicated. A certain subset of userland applications benefit by having a reduced set of available system calls. The resulting set reduces the total kernel surface exposed to the application. System call filtering is meant for use with those applications.

Seccomp filtering provides a means for a process to specify a filter for incoming system calls. The filter is expressed as a Berkeley Packet Filter (BPF) program, as with socket filters, except that the data operated on is related to the system call being made: system call number and the system call arguments. This allows for expressive filtering of system calls using a filter program language with a long history of being exposed to userland and a straightforward data set.

Additionally, BPF makes it impossible for users of seccomp to fall prey to time-of-check-time-of-use (TOCTOU) attacks that are common in system call interposition frameworks. BPF programs may not dereference pointers which constrains all filters to solely evaluating the system call arguments directly.

2.2 What it isn't

System call filtering isn't a sandbox. It provides a clearly defined mechanism for minimizing the exposed kernel surface. It is meant to be a tool for sandbox developers to use. Beyond that, policy for logical behavior and information flow should be managed with a combination of other system hardening techniques and, potentially, an LSM of your choosing. Expressive, dynamic filters provide further options down this path (avoiding pathological sizes or selecting which of the multiplexed system calls in socketcall() is allowed, for instance) which could be construed, incorrectly, as a more complete sandboxing solution.

2.3 Usage

An additional seccomp mode is added and is enabled using the same prctl(2) call as the strict seccomp. If the architecture has `CONFIG_HAVE_ARCH_SECCOMP_FILTER`, then filters may be added as below:

PR_SET_SECCOMP:
Now takes an additional argument which specifies a new filter using a BPF program. The BPF program will be executed over struct seccomp_data reflecting the system call number, arguments, and other metadata. The BPF program must then return one of the acceptable values to inform the kernel which action should be taken.

Usage:

```
prctl(PR_SET_SECCOMP, SECCOMP_MODE_FILTER, prog);
```

The 'prog' argument is a pointer to a struct sock_fprog which will contain the filter program. If the program is invalid, the call will return -1 and set errno to `EINVAL`.

If `fork`/`clone` and `execve` are allowed by @prog, any child processes will be constrained to the same filters and system call ABI as the parent.

Prior to use, the task must call `prctl(PR_SET_NO_NEW_PRIVS, 1)` or run with `CAP_SYS_ADMIN` privileges in its namespace. If these are not true, `-EACCES` will be returned. This requirement ensures that filter programs cannot be applied to child processes with greater privileges than the task that installed them.

Additionally, if `prctl(2)` is allowed by the attached filter, additional filters may be layered on which will increase evaluation time, but allow for further decreasing the attack surface during execution of a process.

The above call returns 0 on success and non-zero on error.

2.4 Return values

A seccomp filter may return any of the following values. If multiple filters exist, the return value for the evaluation of a given system call will always use the highest precedent value. (For example, `SECCOMP_RET_KILL_PROCESS` will always take precedence.)

In precedence order, they are:

SECCOMP_RET_KILL_PROCESS:
Results in the entire process exiting immediately without executing the system call. The exit status of the task (`status & 0x7f`) will be `SIGSYS`, not `SIGKILL`.

SECCOMP_RET_KILL_THREAD:
Results in the task exiting immediately without executing the system call. The exit status of the task (`status & 0x7f`) will be `SIGSYS`, not `SIGKILL`.

SECCOMP_RET_TRAP:
Results in the kernel sending a SIGSYS signal to the triggering task without executing the system call. siginfo->si_call_addr will show the address of the system call instruction, and siginfo->si_syscall and siginfo->si_arch will indicate which syscall was attempted. The program counter will be as though the syscall happened (i.e. it will not point to the syscall instruction). The return value register will contain an arch- dependent value – if resuming execution, set it to something sensible. (The architecture dependency is because replacing it with -ENOSYS could overwrite some useful information.)

The SECCOMP_RET_DATA portion of the return value will be passed as si_errno.

SIGSYS triggered by seccomp will have a si_code of SYS_SECCOMP.

SECCOMP_RET_ERRNO:
Results in the lower 16-bits of the return value being passed to userland as the errno without executing the system call.

SECCOMP_RET_USER_NOTIF:
Results in a struct seccomp_notif message sent on the userspace notification fd, if it is attached, or -ENOSYS if it is not. See below on discussion of how to handle user notifications.

SECCOMP_RET_TRACE:
When returned, this value will cause the kernel to attempt to notify a ptrace()-based tracer prior to executing the system call. If there is no tracer present, -ENOSYS is returned to userland and the system call is not executed.

A tracer will be notified if it requests PTRACE_O_TRACESECCOMP using ptrace(PTRACE_SETOPTIONS). The tracer will be notified of a PTRACE_EVENT_SECCOMP and the SECCOMP_RET_DATA portion of the BPF program return value will be available to the tracer via PTRACE_GETEVENTMSG.

The tracer can skip the system call by changing the syscall number to -1. Alternatively, the tracer can change the system call requested by changing the system call to a valid syscall number. If the tracer asks to skip the system call, then the system call will appear to return the value that the tracer puts in the return value register.

The seccomp check will not be run again after the tracer is notified. (This means that seccomp-based sandboxes MUST NOT allow use of ptrace, even of other sandboxed processes, without extreme care; ptracers can use this mechanism to escape.)

SECCOMP_RET_LOG:
Results in the system call being executed after it is logged. This should be used by application developers to learn which syscalls their application needs without having to iterate through multiple test and development cycles to build the list.

This action will only be logged if "log" is present in the actions_logged sysctl string.

SECCOMP_RET_ALLOW:
Results in the system call being executed.

If multiple filters exist, the return value for the evaluation of a given system call will always use the highest precedent value.

Precedence is only determined using the SECCOMP_RET_ACTION mask. When multiple filters return values of the same precedence, only the SECCOMP_RET_DATA from the most recently installed filter will be returned.

2.5 Pitfalls

The biggest pitfall to avoid during use is filtering on system call number without checking the architecture value. Why? On any architecture that supports multiple system call invocation conventions, the system call numbers may vary based on the specific invocation. If the numbers in the different calling conventions overlap, then checks in the filters may be abused. Always check the arch value!

2.6 Example

The samples/seccomp/ directory contains both an x86-specific example and a more generic example of a higher level macro interface for BPF program generation.

2.7 Userspace Notification

The SECCOMP_RET_USER_NOTIF return code lets seccomp filters pass a particular syscall to userspace to be handled. This may be useful for applications like container managers, which wish to intercept particular syscalls (mount(), finit_module(), etc.) and change their behavior.

To acquire a notification FD, use the SECCOMP_FILTER_FLAG_NEW_LISTENER argument to the seccomp() syscall:

```
fd = seccomp(SECCOMP_SET_MODE_FILTER, SECCOMP_FILTER_FLAG_NEW_
↪LISTENER, &prog);
```

which (on success) will return a listener fd for the filter, which can then be passed around via SCM_RIGHTS or similar. Note that filter fds correspond to a particular filter, and not a particular task. So if this task then forks, notifications from both tasks will appear on the same filter fd. Reads and writes to/from a filter fd are also synchronized, so a filter fd can safely have many readers.

The interface for a seccomp notification fd consists of two structures:

```
struct seccomp_notif_sizes {
    __u16 seccomp_notif;
    __u16 seccomp_notif_resp;
    __u16 seccomp_data;
};
```

(continues on next page)

(continued from previous page)

```
struct seccomp_notif {
    __u64 id;
    __u32 pid;
    __u32 flags;
    struct seccomp_data data;
};

struct seccomp_notif_resp {
    __u64 id;
    __s64 val;
    __s32 error;
    __u32 flags;
};
```

The `struct seccomp_notif_sizes` structure can be used to determine the size of the various structures used in seccomp notifications. The size of `struct seccomp_data` may change in the future, so code should use:

```
struct seccomp_notif_sizes sizes;
seccomp(SECCOMP_GET_NOTIF_SIZES, 0, &sizes);
```

to determine the size of the various structures to allocate. See samples/seccomp/user-trap.c for an example.

Users can read via `ioctl(SECCOMP_IOCTL_NOTIF_RECV)` (or `poll()`) on a seccomp notification fd to receive a `struct seccomp_notif`, which contains five members: the input length of the structure, a unique-per-filter `id`, the `pid` of the task which triggered this request (which may be 0 if the task is in a pid ns not visible from the listener's pid namespace). The notification also contains the `data` passed to seccomp, and a filters flag. The structure should be zeroed out prior to calling the ioctl.

Userspace can then make a decision based on this information about what to do, and `ioctl(SECCOMP_IOCTL_NOTIF_SEND)` a response, indicating what should be returned to userspace. The `id` member of `struct seccomp_notif_resp` should be the same `id` as in `struct seccomp_notif`.

It is worth noting that `struct seccomp_data` contains the values of register arguments to the syscall, but does not contain pointers to memory. The task's memory is accessible to suitably privileged traces via `ptrace()` or `/proc/pid/mem`. However, care should be taken to avoid the TOCTOU mentioned above in this document: all arguments being read from the tracee's memory should be read into the tracer's memory before any policy decisions are made. This allows for an atomic decision on syscall arguments.

2.8 Sysctls

Seccomp's sysctl files can be found in the `/proc/sys/kernel/seccomp/` directory. Here's a description of each file in that directory:

actions_avail:
A read-only ordered list of seccomp return values (refer to the `SECCOMP_RET_*` macros above) in string form. The ordering, from left-to-right, is the least permissive return value to the most permissive return value.

The list represents the set of seccomp return values supported by the kernel. A userspace program may use this list to determine if the actions found in the `seccomp.h`, when the program was built, differs from the set of actions actually supported in the current running kernel.

actions_logged:
A read-write ordered list of seccomp return values (refer to the `SECCOMP_RET_*` macros above) that are allowed to be logged. Writes to the file do not need to be in ordered form but reads from the file will be ordered in the same way as the actions_avail sysctl.

The `allow` string is not accepted in the `actions_logged` sysctl as it is not possible to log `SECCOMP_RET_ALLOW` actions. Attempting to write `allow` to the sysctl will result in an EINVAL being returned.

2.9 Adding architecture support

See `arch/Kconfig` for the authoritative requirements. In general, if an architecture supports both ptrace_event and seccomp, it will be able to support seccomp filter with minor fixup: `SIGSYS` support and seccomp return value checking. Then it must just add `CONFIG_HAVE_ARCH_SECCOMP_FILTER` to its arch-specific Kconfig.

2.10 Caveats

The vDSO can cause some system calls to run entirely in userspace, leading to surprises when you run programs on different machines that fall back to real syscalls. To minimize these surprises on x86, make sure you test with `/sys/devices/system/clocksource/clocksource0/current_clocksource` set to something like `acpi_pm`.

On x86-64, vsyscall emulation is enabled by default. (vsyscalls are legacy variants on vDSO calls.) Currently, emulated vsyscalls will honor seccomp, with a few oddities:

- A return value of `SECCOMP_RET_TRAP` will set a `si_call_addr` pointing to the vsyscall entry for the given call and not the address after the 'syscall' instruction. Any code which wants to restart the call should be aware that (a) a ret instruction has been emulated and (b) trying to resume the syscall will again trigger the standard vsyscall emulation security checks, making resuming the syscall mostly pointless.

- A return value of `SECCOMP_RET_TRACE` will signal the tracer as usual, but the syscall may not be changed to another system call using the orig_rax register. It may only be changed to -1 order to skip the currently emulated call. Any other change MAY terminate the process. The rip value seen by the tracer will be the syscall entry address; this is different from normal behavior. The tracer MUST NOT modify rip or rsp. (Do not rely on other changes terminating the process. They might work. For example, on some kernels, choosing a syscall that only exists in future kernels will be correctly emulated (by returning `-ENOSYS`).

To detect this quirky behavior, check for `addr & ~0x0C00 == 0xFFFFFFFFFF600000`. (For `SECCOMP_RET_TRACE`, use rip. For `SECCOMP_RET_TRAP`, use `siginfo->si_call_addr`.) Do not check any other condition: future kernels may improve vsyscall emulation and current kernels in vsyscall=native mode will behave differently, but the instructions at `0xF...F600{0,4,8,C}00` will not be system calls in these cases.

Note that modern systems are unlikely to use vsyscalls at all – they are a legacy feature and they are considerably slower than standard syscalls. New code will use the vDSO, and vDSO-issued system calls are indistinguishable from normal system calls.

CHAPTER

THREE

UNSHARE SYSTEM CALL

This document describes the new system call, unshare(). The document provides an overview of the feature, why it is needed, how it can be used, its interface specification, design, implementation and how it can be tested.

3.1 Change Log

version 0.1 Initial document, Janak Desai (janak@us.ibm.com), Jan 11, 2006

3.2 Contents

1) Overview
2) Benefits
3) Cost
4) Requirements
5) Functional Specification
6) High Level Design
7) Low Level Design
8) Test Specification
9) Future Work

3.3 1) Overview

Most legacy operating system kernels support an abstraction of threads as multiple execution contexts within a process. These kernels provide special resources and mechanisms to maintain these "threads". The Linux kernel, in a clever and simple manner, does not make distinction between processes and "threads". The kernel allows processes to share resources and thus they can achieve legacy "threads" behavior without requiring additional data structures and mechanisms in the kernel. The power of implementing threads in this manner comes not only from its simplicity but also from allowing application programmers to work outside the confinement of all-or-nothing shared resources of legacy threads. On Linux, at the time of thread creation using the clone system call, applications can selectively choose which resources to share between threads.

unshare() system call adds a primitive to the Linux thread model that allows threads to selectively 'unshare' any resources that were being shared at the time of their creation. unshare() was conceptualized by Al Viro in the August of 2000, on the Linux-Kernel mailing list, as part of the discussion on POSIX threads on Linux. unshare() augments the usefulness of Linux threads for applications that would like to control shared resources without creating a new process. unshare() is a natural addition to the set of available primitives on Linux that implement the concept of process/thread as a virtual machine.

3.4 2) Benefits

unshare() would be useful to large application frameworks such as PAM where creating a new process to control sharing/unsharing of process resources is not possible. Since namespaces are shared by default when creating a new process using fork or clone, unshare() can benefit even non-threaded applications if they have a need to disassociate from default shared namespace. The following lists two use-cases where unshare() can be used.

3.4.1 2.1 Per-security context namespaces

unshare() can be used to implement polyinstantiated directories using the kernel's per-process namespace mechanism. Polyinstantiated directories, such as per-user and/or per-security context instance of /tmp, /var/tmp or per-security context instance of a user's home directory, isolate user processes when working with these directories. Using unshare(), a PAM module can easily setup a private namespace for a user at login. Polyinstantiated directories are required for Common Criteria certification with Labeled System Protection Profile, however, with the availability of shared-tree feature in the Linux kernel, even regular Linux systems can benefit from setting up private namespaces at login and polyinstantiating /tmp, /var/tmp and other directories deemed appropriate by system administrators.

3.4.2 2.2 unsharing of virtual memory and/or open files

Consider a client/server application where the server is processing client requests by creating processes that share resources such as virtual memory and open files. Without unshare(), the server has to decide what needs to be shared at the time of creating the process which services the request. unshare() allows the server an ability to disassociate parts of the context during the servicing of the request. For large and complex middleware application frameworks, this ability to unshare() after the process was created can be very useful.

3.5 3) Cost

In order to not duplicate code and to handle the fact that unshare() works on an active task (as opposed to clone/fork working on a newly allocated inactive task) unshare() had to make minor reorganizational changes to copy_* functions utilized by clone/fork system call. There is a cost associated with altering existing, well tested and stable code to implement a new feature that may not get exercised extensively in the beginning. However, with proper design and code review of the changes and creation of an unshare() test for the LTP the benefits of this new feature can exceed its cost.

3.6 4) Requirements

unshare() reverses sharing that was done using clone(2) system call, so unshare() should have a similar interface as clone(2). That is, since flags in clone(int flags, void *stack) specifies what should be shared, similar flags in unshare(int flags) should specify what should be unshared. Unfortunately, this may appear to invert the meaning of the flags from the way they are used in clone(2). However, there was no easy solution that was less confusing and that allowed incremental context unsharing in future without an ABI change.

unshare() interface should accommodate possible future addition of new context flags without requiring a rebuild of old applications. If and when new context flags are added, unshare() design should allow incremental unsharing of those resources on an as needed basis.

3.7 5) Functional Specification

NAME

unshare - disassociate parts of the process execution context

SYNOPSIS

#include <sched.h>

int unshare(int flags);

DESCRIPTION

unshare() allows a process to disassociate parts of its execution context that are currently being shared with other processes. Part of execution context, such as the namespace, is shared by default when a new process is created using fork(2), while other parts, such as the virtual memory, open file descriptors, etc, may be shared by explicit request to share them when creating a process using clone(2).

The main use of unshare() is to allow a process to control its shared execution context without creating a new process.

The flags argument specifies one or bitwise-or'ed of several of the following constants.

CLONE_FS
    If CLONE_FS is set, file system information of the caller is disassociated from the shared file system information.

CLONE_FILES
    If CLONE_FILES is set, the file descriptor table of the caller is disassociated from the shared file descriptor table.

CLONE_NEWNS
    If CLONE_NEWNS is set, the namespace of the caller is disassociated from the shared namespace.

CLONE_VM
    If CLONE_VM is set, the virtual memory of the caller is disassociated from the shared virtual memory.

RETURN VALUE
    On success, zero returned. On failure, -1 is returned and errno is

ERRORS

EPERM CLONE_NEWNS was specified by a non-root process (process
    without CAP_SYS_ADMIN).

ENOMEM Cannot allocate sufficient memory to copy parts of caller' s
    context that need to be unshared.

EINVAL Invalid flag was specified as an argument.

CONFORMING TO
    The unshare() call is Linux-specific and should not be used in programs intended to be portable.

SEE ALSO
    clone(2), fork(2)

3.8 6) High Level Design

Depending on the flags argument, the unshare() system call allocates appropriate process context structures, populates it with values from the current shared version, associates newly duplicated structures with the current task structure and releases corresponding shared versions. Helper functions of clone (copy_*) could not be used directly by unshare() because of the following two reasons.

1) clone operates on a newly allocated not-yet-active task structure, where as unshare() operates on the current active task. Therefore unshare() has to take appropriate task_lock() before associating newly duplicated context structures

2) unshare() has to allocate and duplicate all context structures that are being unshared, before associating them with the current task and releasing older shared structures. Failure do so will create race conditions and/or oops when trying to backout due to an error. Consider the case of unsharing both virtual memory and namespace. After successfully unsharing vm, if the system call encounters an error while allocating new namespace structure, the error return code will have to reverse the unsharing of vm. As part of the reversal

the system call will have to go back to older, shared, vm structure, which may not exist anymore.

Therefore code from copy_* functions that allocated and duplicated current context structure was moved into new dup_* functions. Now, copy_* functions call dup_* functions to allocate and duplicate appropriate context structures and then associate them with the task structure that is being constructed. unshare() system call on the other hand performs the following:

1) Check flags to force missing, but implied, flags
2) For each context structure, call the corresponding unshare() helper function to allocate and duplicate a new context structure, if the appropriate bit is set in the flags argument.
3) If there is no error in allocation and duplication and there are new context structures then lock the current task structure, associate new context structures with the current task structure, and release the lock on the current task structure.
4) Appropriately release older, shared, context structures.

3.9 7) Low Level Design

Implementation of unshare() can be grouped in the following 4 different items:

a) Reorganization of existing copy_* functions
b) unshare() system call service function
c) unshare() helper functions for each different process context
d) Registration of system call number for different architectures

3.9.1 7.1) Reorganization of copy_* functions

Each copy function such as copy_mm, copy_namespace, copy_files, etc, had roughly two components. The first component allocated and duplicated the appropriate structure and the second component linked it to the task structure passed in as an argument to the copy function. The first component was split into its own function. These dup_* functions allocated and duplicated the appropriate context structure. The reorganized copy_* functions invoked their corresponding dup_* functions and then linked the newly duplicated structures to the task structure with which the copy function was called.

3.9.2 7.2) unshare() system call service function

- Check flags Force implied flags. If CLONE_THREAD is set force CLONE_VM. If CLONE_VM is set, force CLONE_SIGHAND. If CLONE_SIGHAND is set and signals are also being shared, force CLONE_THREAD. If CLONE_NEWNS is set, force CLONE_FS.
- For each context flag, invoke the corresponding unshare_* helper routine with flags passed into the system call and a reference to pointer pointing the new unshared structure
- If any new structures are created by unshare_* helper functions, take the task_lock() on the current task, modify appropriate context pointers, and release the task lock.
- For all newly unshared structures, release the corresponding older, shared, structures.

3.9.3 7.3) unshare_* helper functions

For unshare_* helpers corresponding to CLONE_SYSVSEM, CLONE_SIGHAND, and CLONE_THREAD, return -EINVAL since they are not implemented yet. For others, check the flag value to see if the unsharing is required for that structure. If it is, invoke the corresponding dup_* function to allocate and duplicate the structure and return a pointer to it.

3.9.4 7.4) Finally

Appropriately modify architecture specific code to register the new system call.

3.10 8) Test Specification

The test for unshare() should test the following:

1) Valid flags: Test to check that clone flags for signal and signal handlers, for which unsharing is not implemented yet, return -EINVAL.
2) Missing/implied flags: Test to make sure that if unsharing namespace without specifying unsharing of filesystem, correctly unshares both namespace and filesystem information.
3) For each of the four (namespace, filesystem, files and vm) supported unsharing, verify that the system call correctly unshares the appropriate structure. Verify that unsharing them individually as well as in combination with each other works as expected.
4) Concurrent execution: Use shared memory segments and futex on an address in the shm segment to synchronize execution of about 10 threads. Have a couple of threads execute execve, a couple _exit and the rest unshare with different combination of flags. Verify that unsharing is performed as expected and that there are no oops or hangs.

3.11 9) Future Work

The current implementation of unshare() does not allow unsharing of signals and signal handlers. Signals are complex to begin with and to unshare signals and/or signal handlers of a currently running process is even more complex. If in the future there is a specific need to allow unsharing of signals and/or signal handlers, it can be incrementally added to unshare() without affecting legacy applications using unshare().

CHAPTER

FOUR

SPECULATION CONTROL

Quite some CPUs have speculation-related misfeatures which are in fact vulnerabilities causing data leaks in various forms even across privilege domains.

The kernel provides mitigation for such vulnerabilities in various forms. Some of these mitigations are compile-time configurable and some can be supplied on the kernel command line.

There is also a class of mitigations which are very expensive, but they can be restricted to a certain set of processes or tasks in controlled environments. The mechanism to control these mitigations is via *prctl(2)*.

There are two prctl options which are related to this:

- PR_GET_SPECULATION_CTRL
- PR_SET_SPECULATION_CTRL

4.1 PR_GET_SPECULATION_CTRL

PR_GET_SPECULATION_CTRL returns the state of the speculation misfeature which is selected with arg2 of prctl(2). The return value uses bits 0-3 with the following meaning:

Bit	Define	Description
0	PR_SPEC_PRCTL	Mitigation can be controlled per task by PR_SET_SPECULATION_CTRL.
1	PR_SPEC_ENABL	The speculation feature is enabled, mitigation is disabled.
2	PR_SPEC_DISAB	The speculation feature is disabled, mitigation is enabled.
3	PR_SPEC_FORCE	Same as PR_SPEC_DISABLE, but cannot be undone. A subsequent prctl(…, PR_SPEC_ENABLE) will fail.
4	PR_SPEC_DISAB	Same as PR_SPEC_DISABLE, but the state will be cleared on *execve(2)*.

If all bits are 0 the CPU is not affected by the speculation misfeature.

If PR_SPEC_PRCTL is set, then the per-task control of the mitigation is available. If not set, prctl(PR_SET_SPECULATION_CTRL) for the speculation misfeature will fail.

4.2 PR_SET_SPECULATION_CTRL

PR_SET_SPECULATION_CTRL allows to control the speculation misfeature, which is selected by arg2 of *prctl(2)* per task. arg3 is used to hand in the control value, i.e. either PR_SPEC_ENABLE or PR_SPEC_DISABLE or PR_SPEC_FORCE_DISABLE.

4.3 Common error codes

Value	Meaning
EIN-VAL	The prctl is not implemented by the architecture or unused prctl(2) arguments are not 0.
EN-ODEV	arg2 is selecting a not supported speculation misfeature.

4.4 PR_SET_SPECULATION_CTRL error codes

Value	Meaning
0	Success
ERANG	arg3 is incorrect, i.e. it' s neither PR_SPEC_ENABLE nor PR_SPEC_DISABLE nor PR_SPEC_FORCE_DISABLE.
ENXIO	Control of the selected speculation misfeature is not possible. See PR_GET_SPECULATION_CTRL.
EPERM	Speculation was disabled with PR_SPEC_FORCE_DISABLE and caller tried to enable it again.

4.5 Speculation misfeature controls

- PR_SPEC_STORE_BYPASS: Speculative Store Bypass

  Invocations:

  - prctl(PR_GET_SPECULATION_CTRL, PR_SPEC_STORE_BYPASS, 0, 0, 0);
  - prctl(PR_SET_SPECULATION_CTRL, PR_SPEC_STORE_BYPASS, PR_SPEC_ENABLE, 0, 0);
  - prctl(PR_SET_SPECULATION_CTRL, PR_SPEC_STORE_BYPASS, PR_SPEC_DISABLE, 0, 0);
  - prctl(PR_SET_SPECULATION_CTRL, PR_SPEC_STORE_BYPASS, PR_SPEC_FORCE_DISABLE, 0, 0);
  - prctl(PR_SET_SPECULATION_CTRL, PR_SPEC_STORE_BYPASS, PR_SPEC_DISABLE_NOEXEC, 0, 0);

- **PR_SPEC_INDIR_BRANCH: Indirect Branch Speculation in User Processes**
  (Mitigate Spectre V2 style attacks against user processes)

  Invocations:

  - prctl(PR_GET_SPECULATION_CTRL, PR_SPEC_INDIRECT_BRANCH, 0, 0, 0);
  - prctl(PR_SET_SPECULATION_CTRL, PR_SPEC_INDIRECT_BRANCH, PR_SPEC_ENABLE, 0, 0);
  - prctl(PR_SET_SPECULATION_CTRL, PR_SPEC_INDIRECT_BRANCH, PR_SPEC_DISABLE, 0, 0);
  - prctl(PR_SET_SPECULATION_CTRL, PR_SPEC_INDIRECT_BRANCH, PR_SPEC_FORCE_DISABLE, 0, 0);

CHAPTER

FIVE

OPENCAPI (OPEN COHERENT ACCELERATOR PROCESSOR INTERFACE)

OpenCAPI is an interface between processors and accelerators. It aims at being low-latency and high-bandwidth. The specification is developed by the OpenCAPI Consortium.

It allows an accelerator (which could be a FPGA, ASICs, …) to access the host memory coherently, using virtual addresses. An OpenCAPI device can also host its own memory, that can be accessed from the host.

OpenCAPI is known in linux as ‘ocxl’, as the open, processor-agnostic evolution of ‘cxl’ (the driver for the IBM CAPI interface for powerpc), which was named that way to avoid confusion with the ISDN CAPI subsystem.

5.1 High-level view

OpenCAPI defines a Data Link Layer (DL) and Transaction Layer (TL), to be implemented on top of a physical link. Any processor or device implementing the DL and TL can start sharing memory.

```
+-----------+                        +-------------+
|           |                        |             |
|           |                        | Accelerated |
| Processor |                        |  Function   |
|           |  +--------+            |    Unit     |  +--------+
|           |--| Memory |            |    (AFU)    |--| Memory |
|           |  +--------+            |             |  +--------+
+-----------+                        +-------------+
     |                                      |
+-----------+                        +-------------+
|    TL     |                        |    TLX      |
+-----------+                        +-------------+
     |                                      |
+-----------+                        +-------------+
|    DL     |                        |    DLX      |
+-----------+                        +-------------+
     |                                      |
     |                  PHY                 |
     +--------------------------------------+
```

5.2 Device discovery

OpenCAPI relies on a PCI-like configuration space, implemented on the device. So the host can discover AFUs by querying the config space.

OpenCAPI devices in Linux are treated like PCI devices (with a few caveats). The firmware is expected to abstract the hardware as if it was a PCI link. A lot of the existing PCI infrastructure is reused: devices are scanned and BARs are assigned during the standard PCI enumeration. Commands like 'lspci' can therefore be used to see what devices are available.

The configuration space defines the AFU(s) that can be found on the physical adapter, such as its name, how many memory contexts it can work with, the size of its MMIO areas, …

5.3 MMIO

OpenCAPI defines two MMIO areas for each AFU:

- the global MMIO area, with registers pertinent to the whole AFU.
- a per-process MMIO area, which has a fixed size for each context.

5.4 AFU interrupts

OpenCAPI includes the possibility for an AFU to send an interrupt to a host process. It is done through a 'intrp_req' defined in the Transaction Layer, specifying a 64-bit object handle which defines the interrupt.

The driver allows a process to allocate an interrupt and obtain its 64-bit object handle, that can be passed to the AFU.

5.5 char devices

The driver creates one char device per AFU found on the physical device. A physical device may have multiple functions and each function can have multiple AFUs. At the time of this writing though, it has only been tested with devices exporting only one AFU.

Char devices can be found in /dev/ocxl/ and are named as: /dev/ocxl/<AFU name>.<location>.<index>

where <AFU name> is a max 20-character long name, as found in the config space of the AFU. <location> is added by the driver and can help distinguish devices when a system has more than one instance of the same OpenCAPI device. <index> is also to help distinguish AFUs in the unlikely case where a device carries multiple copies of the same AFU.

5.6 Sysfs class

An ocxl class is added for the devices representing the AFUs. See /sys/class/ocxl. The layout is described in Documentation/ABI/testing/sysfs-class-ocxl

5.7 User API

5.7.1 open

Based on the AFU definition found in the config space, an AFU may support working with more than one memory context, in which case the associated char device may be opened multiple times by different processes.

5.7.2 ioctl

OCXL_IOCTL_ATTACH:

  Attach the memory context of the calling process to the AFU so that the AFU can access its memory.

OCXL_IOCTL_IRQ_ALLOC:

  Allocate an AFU interrupt and return an identifier.

OCXL_IOCTL_IRQ_FREE:

  Free a previously allocated AFU interrupt.

OCXL_IOCTL_IRQ_SET_FD:

  Associate an event fd to an AFU interrupt so that the user process can be notified when the AFU sends an interrupt.

OCXL_IOCTL_GET_METADATA:

  Obtains configuration information from the card, such at the size of MMIO areas, the AFU version, and the PASID for the current context.

OCXL_IOCTL_ENABLE_P9_WAIT:

  Allows the AFU to wake a userspace thread executing 'wait'. Returns information to userspace to allow it to configure the AFU. Note that this is only available on POWER9.

OCXL_IOCTL_GET_FEATURES:

  Reports on which CPU features that affect OpenCAPI are usable from userspace.

5.7.3 mmap

A process can mmap the per-process MMIO area for interactions with the AFU.

CHAPTER

SIX

IOCTLS

6.1 Ioctl Numbers

19 October 1999

Michael Elizabeth Chastain <mec@shout.net>

If you are adding new ioctl's to the kernel, you should use the _IO macros defined in <linux/ioctl.h>:

_IO	an	ioctl with no parameters
_IOW	an	ioctl with write parameters (copy_from_user)
_IOR	an	ioctl with read parameters (copy_to_user)
_IOWR	an	ioctl with both write and read parameters.

'Write' and 'read' are from the user's point of view, just like the system calls 'write' and 'read'. For example, a SET_FOO ioctl would be _IOW, although the kernel would actually read data from user space; a GET_FOO ioctl would be _IOR, although the kernel would actually write data to user space.

The first argument to _IO, _IOW, _IOR, or _IOWR is an identifying letter or number from the table below. Because of the large number of drivers, many drivers share a partial letter with other drivers.

If you are writing a driver for a new device and need a letter, pick an unused block with enough room for expansion: 32 to 256 ioctl commands. You can register the block by patching this file and submitting the patch to Linus Torvalds. Or you can e-mail me at <mec@shout.net> and I'll register one for you.

The second argument to _IO, _IOW, _IOR, or _IOWR is a sequence number to distinguish ioctls from each other. The third argument to _IOW, _IOR, or _IOWR is the type of the data going into the kernel or coming out of the kernel (e.g. 'int' or 'struct foo'). NOTE! Do NOT use sizeof(arg) as the third argument as this results in your ioctl thinking it passes an argument of type size_t.

Some devices use their major number as the identifier; this is OK, as long as it is unique. Some devices are irregular and don't follow any convention at all.

Following this convention is good because:

(1) Keeping the ioctl's globally unique helps error checking: if a program calls an ioctl on the wrong device, it will get an error rather than some unexpected behaviour.

(2) The 'strace' build procedure automatically finds ioctl numbers defined with _IO, _IOW, _IOR, or _IOWR.

(3) 'strace' can decode numbers back into useful names when the numbers are unique.

(4) People looking for ioctls can grep for them more easily when this convention is used to define the ioctl numbers.

(5) When following the convention, the driver code can use generic code to copy the parameters between user and kernel space.

This table lists ioctls visible from user land for Linux/x86. It contains most drivers up to 2.6.31, but I know I am missing some. There has been no attempt to list non-X86 architectures or ioctls from drivers/staging/.

Code	Seq# (hex)	Include File
0x00	00-1F	linux/fs.h
0x00	00-1F	scsi/scsi_ioctl.h
0x00	00-1F	linux/fb.h
0x00	00-1F	linux/wavefront.h
0x02	all	linux/fd.h
0x03	all	linux/hdreg.h
0x04	D2-DC	linux/umsdos_fs.h
0x06	all	linux/lp.h
0x09	all	linux/raid/md_u.h
0x10	00-0F	drivers/char/s390/vmcp.h
0x10	10-1F	arch/s390/include/uapi/sclp_ctl.h
0x10	20-2F	arch/s390/include/uapi/asm/hypfs.h
0x12	all	linux/fs.h linux/blkpg.h
0x1b	all	
0x20	all	drivers/cdrom/cm206.h
0x22	all	scsi/sg.h
'!'	00-1F	uapi/linux/seccomp.h
'#'	00-3F	
'$'	00-0F	linux/perf_counter.h, linux/perf_event.h
'%'	00-0F	include/uapi/linux/stm.h
'&'	00-07	drivers/firewire/nosy-user.h
'1'	00-1F	linux/timepps.h
'2'	01-04	linux/i2o.h
'3'	00-0F	drivers/s390/char/raw3270.h
'3'	00-1F	linux/suspend_ioctls.h, kernel/power/user.c
'8'	all	
';'	64-7F	linux/vfio.h
'@'	00-0F	linux/radeonfb.h
'@'	00-0F	drivers/video/aty/aty128fb.c
'A'	00-1F	linux/apm_bios.h
'A'	00-0F	linux/agpgart.h, drivers/char/agp/compat_ioctl.h
'A'	00-7F	sound/asound.h
'B'	00-1F	linux/cciss_ioctl.h
'B'	00-0F	include/linux/pmu.h

Table 1 – c

Code	Seq# (hex)	Include File
‘B’	C0-FF	advanced bbus
‘C’	all	linux/soundcard.h
‘C’	01-2F	linux/capi.h
‘C’	F0-FF	drivers/net/wan/cosa.h
‘D’	all	arch/s390/include/asm/dasd.h
‘D’	40-5F	drivers/scsi/dpt/dtpi_ioctl.h
‘D’	05	drivers/scsi/pmcraid.h
‘E’	all	linux/input.h
‘E’	00-0F	xen/evtchn.h
‘F’	all	linux/fb.h
‘F’	01-02	drivers/scsi/pmcraid.h
‘F’	20	drivers/video/fsl-diu-fb.h
‘F’	20	drivers/video/intelfb/intelfb.h
‘F’	20	linux/ivtvfb.h
‘F’	20	linux/matroxfb.h
‘F’	20	drivers/video/aty/atyfb_base.c
‘F’	00-0F	video/da8xx-fb.h
‘F’	80-8F	linux/arcfb.h
‘F’	DD	video/sstfb.h
‘G’	00-3F	drivers/misc/sgi-gru/grulib.h
‘H’	00-7F	linux/hiddev.h
‘H’	00-0F	linux/hidraw.h
‘H’	01	linux/mei.h
‘H’	02	linux/mei.h
‘H’	03	linux/mei.h
‘H’	00-0F	sound/asound.h
‘H’	20-40	sound/asound_fm.h
‘H’	80-8F	sound/sfnt_info.h
‘H’	10-8F	sound/emu10k1.h
‘H’	10-1F	sound/sb16_csp.h
‘H’	10-1F	sound/hda_hwdep.h
‘H’	40-4F	sound/hdspm.h
‘H’	40-4F	sound/hdsp.h
‘H’	90	sound/usb/usx2y/usb_stream.h
‘H’	00-0F	uapi/misc/habanalabs.h
‘H’	A0	uapi/linux/usb/cdc-wdm.h
‘H’	C0-F0	net/bluetooth/hci.h
‘H’	C0-DF	net/bluetooth/hidp/hidp.h
‘H’	C0-DF	net/bluetooth/cmtp/cmtp.h
‘H’	C0-DF	net/bluetooth/bnep/bnep.h
‘H’	F1	linux/hid-roccat.h
‘H’	F8-FA	sound/firewire.h
‘I’	all	linux/isdn.h
‘I’	00-0F	drivers/isdn/divert/isdn_divert.h
‘I’	40-4F	linux/mISDNif.h
‘J’	00-1F	drivers/scsi/gdth_ioctl.h
‘K’	all	linux/kd.h
‘L’	00-1F	linux/loop.h

Table 1 – c

Code	Seq# (hex)	Include File
'L'	10-1F	drivers/scsi/mpt3sas/mpt3sas_ctl.h
'L'	20-2F	linux/lightnvm.h
'L'	E0-FF	linux/ppdd.h
'M'	all	linux/soundcard.h
'M'	01-16 and	mtd/mtd-abi.h drivers/mtd/mtdchar.c
'M'	01-03	drivers/scsi/megaraid/megaraid_sas.h
'M'	00-0F	drivers/video/fsl-diu-fb.h
'N'	00-1F	drivers/usb/scanner.h
'N'	40-7F	drivers/block/nvme.c
'O'	00-06	mtd/ubi-user.h
'P'	all	linux/soundcard.h
'P'	60-6F	sound/sscape_ioctl.h
'P'	00-0F	drivers/usb/class/usblp.c
'P'	01-09	drivers/misc/pci_endpoint_test.c
'Q'	all	linux/soundcard.h
'R'	00-1F	linux/random.h
'R'	01	linux/rfkill.h
'R'	C0-DF	net/bluetooth/rfcomm.h
'S'	all	linux/cdrom.h
'S'	80-81	scsi/scsi_ioctl.h
'S'	82-FF	scsi/scsi.h
'S'	00-7F	sound/asequencer.h
'T'	all	linux/soundcard.h
'T'	00-AF	sound/asound.h
'T'	all	arch/x86/include/asm/ioctls.h
'T'	C0-DF	linux/if_tun.h
'U'	all	sound/asound.h
'U'	00-CF	linux/uinput.h
'U'	00-EF	linux/usbdevice_fs.h
'U'	C0-CF	drivers/bluetooth/hci_uart.h
'V'	all	linux/vt.h
'V'	all	linux/videodev2.h
'V'	C0	linux/ivtvfb.h
'V'	C0	linux/ivtv.h
'V'	C0	media/davinci/vpfe_capture.h
'V'	C0	media/si4713.h
'W'	00-1F	linux/watchdog.h
'W'	00-1F	linux/wanrouter.h
'W'	00-3F	sound/asound.h
'W'	40-5F	drivers/pci/switch/switchtec.c
'W'	60-61	linux/watch_queue.h
'X'	all	fs/xfs/xfs_fs.h, fs/xfs/linux-2.6/xfs_ioctl32.h, include/linux/falloc.h, linux/f
'X'	all	fs/ocfs2/ocfs_fs.h
'X'	01	linux/pktcdvd.h
'Y'	all	linux/cyclades.h
'Z'	14-15	drivers/message/fusion/mptctl.h
'['	00-3F	linux/usb/tmc.h
'a'	all	linux/atm*.h, linux/sonet.h

Table 1 – co

Code	Seq# (hex)	Include File
‘a’	00-0F	drivers/crypto/qat/qat_common/adf_cfg_common.h
‘b’	00-FF	
‘c’	all	linux/cm4000_cs.h
‘c’	00-7F	linux/comstats.h
‘c’	00-7F	linux/coda.h
‘c’	00-1F	linux/chio.h
‘c’	80-9F	arch/s390/include/asm/chsc.h
‘c’	A0-AF	arch/x86/include/asm/msr.h conflict!
‘d’	00-FF	linux/char/drm/drm.h
‘d’	02-40	pcmcia/ds.h
‘d’	F0-FF	linux/digi1.h
‘e’	all	linux/digi1.h
‘f’	00-1F	linux/ext2_fs.h
‘f’	00-1F	linux/ext3_fs.h
‘f’	00-0F	fs/jfs/jfs_dinode.h
‘f’	00-0F	fs/ext4/ext4.h
‘f’	00-0F	linux/fs.h
‘f’	00-0F	fs/ocfs2/ocfs2_fs.h
‘f’	13-27	linux/fscrypt.h
‘f’	81-8F	linux/fsverity.h
‘g’	00-0F	linux/usb/gadgetfs.h
‘g’	20-2F	linux/usb/g_printer.h
‘h’	00-7F	
‘h’	00-1F	linux/hpet.h
‘h’	80-8F	fs/hfsplus/ioctl.c
‘i’	00-3F	linux/i2o-dev.h
‘i’	0B-1F	linux/ipmi.h
‘i’	80-8F	linux/i8k.h
‘j’	00-3F	linux/joystick.h
‘k’	00-0F	linux/spi/spidev.h
‘k’	00-05	video/kyro.h
‘k’	10-17	linux/hsi/hsi_char.h
‘l’	00-3F	linux/tcfs_fs.h
‘l’	40-7F	linux/udf_fs_i.h
‘m’	00-09	linux/mmtimer.h
‘m’	all	linux/mtio.h
‘m’	all	linux/soundcard.h
‘m’	all	linux/synclink.h
‘m’	00-19	drivers/message/fusion/mptctl.h
‘m’	00	drivers/scsi/megaraid/megaraid_ioctl.h
‘n’	00-7F	linux/ncp_fs.h and fs/ncpfs/ioctl.c
‘n’	80-8F	uapi/linux/nilfs2_api.h
‘n’	E0-FF	linux/matroxfb.h
‘o’	00-1F	fs/ocfs2/ocfs2_fs.h
‘o’	00-03	mtd/ubi-user.h
‘o’	40-41	mtd/ubi-user.h
‘o’	01-A1	*linux/dvb/*.h*
‘p’	00-0F	linux/phantom.h

Table 1 – c

Code	Seq# (hex)	Include File
'p'	00-1F	linux/rtc.h
'p'	40-7F	linux/nvram.h
'p'	80-9F	linux/ppdev.h
'p'	A1-A5	linux/pps.h
'q'	00-1F	linux/serio.h
'q'	80-FF	linux/telephony.h linux/ixjuser.h
'r'	00-1F	linux/msdos_fs.h and fs/fat/dir.c
's'	all	linux/cdk.h
't'	00-7F	linux/ppp-ioctl.h
't'	80-8F	linux/isdn_ppp.h
't'	90-91	linux/toshiba.h
'u'	00-1F	linux/smb_fs.h
'u'	20-3F	linux/uvcvideo.h
'u'	40-4f	linux/udmabuf.h
'v'	00-1F	linux/ext2_fs.h
'v'	00-1F	linux/fs.h
'v'	00-0F	linux/sonypi.h
'v'	00-0F	media/v4l2-subdev.h
'v'	20-27	arch/powerpc/include/uapi/asm/vas-api.h
'v'	C0-FF	linux/meye.h
'w'	all	
'y'	00-1F	
'z'	00-3F	
'z'	40-7F	
'z'	10-4F	drivers/s390/crypto/zcrypt_api.h
'|'	00-7F	linux/media.h
0x80	00-1F	linux/fb.h
0x89	00-06	arch/x86/include/asm/sockios.h
0x89	0B-DF	linux/sockios.h
0x89	E0-EF	linux/sockios.h
0x89	F0-FF	linux/sockios.h
0x8B	all	linux/wireless.h
0x8C	00-3F	
0x90	00	drivers/cdrom/sbpcd.h
0x92	00-0F	drivers/usb/mon/mon_bin.c
0x93	60-7F	linux/auto_fs.h
0x94	all	fs/btrfs/ioctl.h and linux/fs.h
0x97	00-7F	fs/ceph/ioctl.h
0x99	00-0F	
0xA0	all	linux/sdp/sdp.h
0xA1	0	linux/vtpm_proxy.h
0xA3	80-8F	
0xA3	90-9F	linux/dtlk.h
0xA4	00-1F	uapi/linux/tee.h
0xAA	00-3F	linux/uapi/linux/userfaultfd.h
0xAB	00-1F	linux/nbd.h
0xAC	00-1F	linux/raw.h
0xAD	00	

Table 1 – c

Code	Seq# (hex)	Include File
0xAE	00-1F	linux/kvm.h
0xAE	40-FF	linux/kvm.h
0xAE	20-3F	linux/nitro_enclaves.h
0xAF	00-1F	linux/fsl_hypervisor.h
0xB0	all	
0xB1	00-1F	
0xB3	00	linux/mmc/ioctl.h
0xB4	00-0F	linux/gpio.h
0xB5	00-0F	uapi/linux/rpmsg.h
0xB6	all	linux/fpga-dfl.h
0xB7	all	uapi/linux/remoteproc_cdev.h
0xC0	00-0F	linux/usb/iowarrior.h
0xCA	00-0F	uapi/misc/cxl.h
0xCA	10-2F	uapi/misc/ocxl.h
0xCA	80-BF	uapi/scsi/cxlflash_ioctl.h
0xCB	00-1F	
0xCC	00-0F	drivers/misc/ibmvmc.h
0xCD	01	linux/reiserfs_fs.h
0xCF	02	fs/cifs/ioctl.c
0xDB	00-0F	drivers/char/mwave/mwavepub.h
0xDD	00-3F	
0xE5	00-3F	linux/fuse.h
0xEC	00-01	drivers/platform/chrome/cros_ec_dev.h
0xF3	00-3F	drivers/usb/misc/sisusbvga/sisusb.h
0xF6	all	
0xFD	all	linux/dm-ioctl.h
0xFE	all	linux/isst_if.h

6.2 Decoding an IOCTL Magic Number

To decode a hex IOCTL code:

Most architectures use this generic format, but check include/ARCH/ioctl.h for specifics, e.g. powerpc uses 3 bits to encode read/write and 13 bits for size.

bits	meaning
31-30	00 - no parameters: uses _IO macro 10 - read: _IOR 01 - write: _IOW 11 - read/write: _IOWR
29-16	size of arguments
15-8	ascii character supposedly unique to each driver
7-0	function #

So for example 0x82187201 is a read with arg length of 0x218, character 'r' function 1. Grepping the source reveals this is:

```
#define VFAT_IOCTL_READDIR_BOTH         _IOR('r', 1, struct dirent␣
↪[2])
```

6.3 Summary of CDROM ioctl calls

- Edward A. Falk <efalk@google.com>

November, 2004

This document attempts to describe the ioctl(2) calls supported by the CDROM layer. These are by-and-large implemented (as of Linux 2.6) in drivers/cdrom/cdrom.c and drivers/block/scsi_ioctl.c

ioctl values are listed in <linux/cdrom.h>. As of this writing, they are as follows:

CDROMPAUSE	Pause Audio Operation
CDROMRESUME	Resume paused Audio Operation
CDROMPLAYMSF	Play Audio MSF (struct cdrom_msf)
CDROMPLAYTRKIND	Play Audio Track/index (struct cdrom_ti)
CDROMREADTOCHDR	Read TOC header (struct cdrom_tochdr)
CDROMREADTOCENTRY	Read TOC entry (struct cdrom_tocentry)
CDROMSTOP	Stop the cdrom drive
CDROMSTART	Start the cdrom drive
CDROMEJECT	Ejects the cdrom media
CDROMVOLCTRL	Control output volume (struct cdrom_volctrl)
CDROMSUBCHNL	Read subchannel data (struct cdrom_subchnl)
CDROMREADMODE2	Read CDROM mode 2 data (2336 Bytes) (struct cdrom_r
CDROMREADMODE1	Read CDROM mode 1 data (2048 Bytes) (struct cdrom_r
CDROMREADAUDIO	(struct cdrom_read_audio)
CDROMEJECT_SW	enable(1)/disable(0) auto-ejecting
CDROMMULTISESSION	Obtain the start-of-last-session address of multi session d
CDROM_GET_MCN	Obtain the "Universal Product Code" if available (struc
CDROM_GET_UPC	Deprecated, use CDROM_GET_MCN instead.
CDROMRESET	hard-reset the drive
CDROMVOLREAD	Get the drive's volume setting (struct cdrom_volctrl)
CDROMREADRAW	read data in raw mode (2352 Bytes) (struct cdrom_read)
CDROMREADCOOKED	read data in cooked mode
CDROMSEEK	seek msf address
CDROMPLAYBLK	scsi-cd only, (struct cdrom_blk)
CDROMREADALL	read all 2646 bytes
CDROMGETSPINDOWN	return 4-bit spindown value
CDROMSETSPINDOWN	set 4-bit spindown value
CDROMCLOSETRAY	pendant of CDROMEJECT
CDROM_SET_OPTIONS	Set behavior options
CDROM_CLEAR_OPTIONS	Clear behavior options
CDROM_SELECT_SPEED	Set the CD-ROM speed
CDROM_SELECT_DISC	Select disc (for juke-boxes)
CDROM_MEDIA_CHANGED	Check is media changed

Table 2 – continued from previous page

CDROM_DRIVE_STATUS	Get tray position, etc.
CDROM_DISC_STATUS	Get disc type, etc.
CDROM_CHANGER_NSLOTS	Get number of slots
CDROM_LOCKDOOR	lock or unlock door
CDROM_DEBUG	Turn debug messages on/off
CDROM_GET_CAPABILITY	get capabilities
CDROMAUDIOBUFSIZ	set the audio buffer size
DVD_READ_STRUCT	Read structure
DVD_WRITE_STRUCT	Write structure
DVD_AUTH	Authentication
CDROM_SEND_PACKET	send a packet to the drive
CDROM_NEXT_WRITABLE	get next writable block
CDROM_LAST_WRITTEN	get last block written on disc

The information that follows was determined from reading kernel source code. It is likely that some corrections will be made over time.

General:

Unless otherwise specified, all ioctl calls return 0 on success and -1 with errno set to an appropriate value on error. (Some ioctls return non-negative data values.)

Unless otherwise specified, all ioctl calls return -1 and set errno to EFAULT on a failed attempt to copy data to or from user address space.

Individual drivers may return error codes not listed here.

Unless otherwise specified, all data structures and constants are defined in <linux/cdrom.h>

CDROMPAUSE

Pause Audio Operation

usage:

```
ioctl(fd, CDROMPAUSE, 0);
```

inputs:

none

outputs:

none

error return:

- ENOSYS cd drive not audio-capable.

CDROMRESUME

Resume paused Audio Operation

usage:

```
ioctl(fd, CDROMRESUME, 0);
```

inputs:
none

outputs:
none

error return:

- ENOSYS cd drive not audio-capable.

CDROMPLAYMSF
Play Audio MSF

(struct cdrom_msf)

usage:

```
struct cdrom_msf msf;

ioctl(fd, CDROMPLAYMSF, &msf);
```

inputs:
cdrom_msf structure, describing a segment of music to play

outputs:
none

error return:

- ENOSYS cd drive not audio-capable.

notes:

- MSF stands for minutes-seconds-frames
- LBA stands for logical block address
- Segment is described as start and end times, where each time is described as minutes:seconds:frames. A frame is 1/75 of a second.

CDROMPLAYTRKIND
Play Audio Track/index

(struct cdrom_ti)

usage:

```
struct cdrom_ti ti;

ioctl(fd, CDROMPLAYTRKIND, &ti);
```

inputs:
cdrom_ti structure, describing a segment of music to play

outputs:
none

error return:

- ENOSYS cd drive not audio-capable.

notes:

- Segment is described as start and end times, where each time is described as a track and an index.

CDROMREADTOCHDR

Read TOC header

(struct cdrom_tochdr)

usage:

```
cdrom_tochdr header;

ioctl(fd, CDROMREADTOCHDR, &header);
```

inputs:

cdrom_tochdr structure

outputs:

cdrom_tochdr structure

error return:

- ENOSYS cd drive not audio-capable.

CDROMREADTOCENTRY

Read TOC entry

(struct cdrom_tocentry)

usage:

```
struct cdrom_tocentry entry;

ioctl(fd, CDROMREADTOCENTRY, &entry);
```

inputs:

cdrom_tocentry structure

outputs:

cdrom_tocentry structure

error return:

- ENOSYS cd drive not audio-capable.
- EINVAL entry.cdte_format not CDROM_MSF or CDROM_LBA
- EINVAL requested track out of bounds
- EIO I/O error reading TOC

notes:

- TOC stands for Table Of Contents
- MSF stands for minutes-seconds-frames
- LBA stands for logical block address

CDROMSTOP

Stop the cdrom drive

usage:

```
ioctl(fd, CDROMSTOP, 0);
```

inputs:
none

outputs:
none

error return:

- ENOSYS cd drive not audio-capable.

notes:

- Exact interpretation of this ioctl depends on the device, but most seem to spin the drive down.

CDROMSTART

Start the cdrom drive

usage:

```
ioctl(fd, CDROMSTART, 0);
```

inputs:
none

outputs:
none

error return:

- ENOSYS cd drive not audio-capable.

notes:

- Exact interpretation of this ioctl depends on the device, but most seem to spin the drive up and/or close the tray. Other devices ignore the ioctl completely.

CDROMEJECT

- Ejects the cdrom media

usage:

```
ioctl(fd, CDROMEJECT, 0);
```

inputs:
none

outputs:
none

error returns:

- ENOSYS cd drive not capable of ejecting
- EBUSY other processes are accessing drive, or door is locked

notes:

- See CDROM_LOCKDOOR, below.

CDROMCLOSETRAY
pendant of CDROMEJECT

usage:

```
ioctl(fd, CDROMCLOSETRAY, 0);
```

inputs:
none

outputs:
none

error returns:

- ENOSYS cd drive not capable of closing the tray
- EBUSY other processes are accessing drive, or door is locked

notes:

- See CDROM_LOCKDOOR, below.

CDROMVOLCTRL
Control output volume (struct cdrom_volctrl)

usage:

```
struct cdrom_volctrl volume;

ioctl(fd, CDROMVOLCTRL, &volume);
```

inputs:
cdrom_volctrl structure containing volumes for up to 4 channels.

outputs:
none

error return:

- ENOSYS cd drive not audio-capable.

CDROMVOLREAD
Get the drive's volume setting

(struct cdrom_volctrl)

usage:

```
struct cdrom_volctrl volume;

ioctl(fd, CDROMVOLREAD, &volume);
```

inputs:
none

outputs:
The current volume settings.

error return:

- ENOSYS cd drive not audio-capable.

CDROMSUBCHNL
Read subchannel data

(struct cdrom_subchnl)

usage:

```
struct cdrom_subchnl q;

ioctl(fd, CDROMSUBCHNL, &q);
```

inputs:
cdrom_subchnl structure

outputs:
cdrom_subchnl structure

error return:

- ENOSYS cd drive not audio-capable.
- EINVAL format not CDROM_MSF or CDROM_LBA

notes:

- Format is converted to CDROM_MSF or CDROM_LBA as per user request on return

CDROMREADRAW
read data in raw mode (2352 Bytes)

(struct cdrom_read)

usage:

```
union {

  struct cdrom_msf msf;               /* input */
  char buffer[CD_FRAMESIZE_RAW];      /* return */
} arg;
ioctl(fd, CDROMREADRAW, &arg);
```

inputs:
cdrom_msf structure indicating an address to read.

Only the start values are significant.

outputs:
Data written to address provided by user.

error return:

- EINVAL address less than 0, or msf less than 0:2:0
- ENOMEM out of memory

notes:

- As of 2.6.8.1, comments in <linux/cdrom.h> indicate that this ioctl accepts a cdrom_read structure, but actual source code reads a cdrom_msf structure and writes a buffer of data to the same address.
- MSF values are converted to LBA values via this formula:

```
lba = (((m * CD_SECS) + s) * CD_FRAMES + f) - CD_MSF_
→OFFSET;
```

CDROMREADMODE1
Read CDROM mode 1 data (2048 Bytes)

(struct cdrom_read)

notes:
Identical to CDROMREADRAW except that block size is CD_FRAMESIZE (2048) bytes

CDROMREADMODE2
Read CDROM mode 2 data (2336 Bytes)

(struct cdrom_read)

notes:
Identical to CDROMREADRAW except that block size is CD_FRAMESIZE_RAW0 (2336) bytes

CDROMREADAUDIO
(struct cdrom_read_audio)

usage:

```
struct cdrom_read_audio ra;

ioctl(fd, CDROMREADAUDIO, &ra);
```

inputs:
cdrom_read_audio structure containing read start point and length

outputs:
audio data, returned to buffer indicated by ra

error return:

- EINVAL format not CDROM_MSF or CDROM_LBA
- EINVAL nframes not in range [1 75]
- ENXIO drive has no queue (probably means invalid fd)
- ENOMEM out of memory

CDROMEJECT_SW
enable(1)/disable(0) auto-ejecting

usage:

```
int val;

ioctl(fd, CDROMEJECT_SW, val);
```

inputs:
Flag specifying auto-eject flag.

outputs:
none

error return:

- ENOSYS Drive is not capable of ejecting.
- EBUSY Door is locked

CDROMMULTISESSION
Obtain the start-of-last-session address of multi session disks

(struct cdrom_multisession)

usage:

```
struct cdrom_multisession ms_info;

ioctl(fd, CDROMMULTISESSION, &ms_info);
```

inputs:

cdrom_multisession structure containing desired

format.

outputs:
cdrom_multisession structure is filled with last_session information.

error return:

- EINVAL format not CDROM_MSF or CDROM_LBA

CDROM_GET_MCN
Obtain the "Universal Product Code" if available

(struct cdrom_mcn)

usage:

```
struct cdrom_mcn mcn;

ioctl(fd, CDROM_GET_MCN, &mcn);
```

inputs:
none

outputs:
Universal Product Code

error return:

- ENOSYS Drive is not capable of reading MCN data.

notes:

- Source code comments state:

```
The following function is implemented, although very few
audio discs give Universal Product Code information,␣
↪which
should just be the Medium Catalog Number on the box. ␣
↪Note,
that the way the code is written on the CD is /not/␣
↪uniform
across all discs!
```

CDROM_GET_UPC
CDROM_GET_MCN (deprecated)

Not implemented, as of 2.6.8.1

CDROMRESET
hard-reset the drive

usage:

```
ioctl(fd, CDROMRESET, 0);
```

inputs:
none

outputs:
none

error return:

- EACCES Access denied: requires CAP_SYS_ADMIN
- ENOSYS Drive is not capable of resetting.

CDROMREADCOOKED
read data in cooked mode

usage:

```
u8 buffer[CD_FRAMESIZE]

ioctl(fd, CDROMREADCOOKED, buffer);
```

inputs:
none

outputs:
2048 bytes of data, "cooked" mode.

notes:
Not implemented on all drives.

CDROMREADALL
read all 2646 bytes

Same as CDROMREADCOOKED, but reads 2646 bytes.

CDROMSEEK
seek msf address

usage:

```
struct cdrom_msf msf;

ioctl(fd, CDROMSEEK, &msf);
```

inputs:
MSF address to seek to.

outputs:
none

CDROMPLAYBLK
scsi-cd only

(struct cdrom_blk)

usage:

```
struct cdrom_blk blk;

ioctl(fd, CDROMPLAYBLK, &blk);
```

inputs:
Region to play

outputs:
none

CDROMGETSPINDOWN
usage:

```
char spindown;

ioctl(fd, CDROMGETSPINDOWN, &spindown);
```

inputs:
none

outputs:
The value of the current 4-bit spindown value.

CDROMSETSPINDOWN
usage:

```
char spindown

ioctl(fd, CDROMSETSPINDOWN, &spindown);
```

inputs:
4-bit value used to control spindown (TODO: more detail here)

outputs:
none

CDROM_SET_OPTIONS
Set behavior options

usage:

```
int options;

ioctl(fd, CDROM_SET_OPTIONS, options);
```

inputs:

New values for drive options. The logical 'or' of:

CDO_AUTO_CLOSE	close tray on first open(2)
CDO_AUTO_EJECT	open tray on last release
CDO_USE_FFLAGS	use O_NONBLOCK information on open
CDO_LOCK	lock tray on open files
CDO_CHECK_TYPE	check type on open for data

outputs:
Returns the resulting options settings in the ioctl return value. Returns -1 on error.

error return:

- ENOSYS selected option(s) not supported by drive.

CDROM_CLEAR_OPTIONS
Clear behavior options

Same as CDROM_SET_OPTIONS, except that selected options are turned off.

CDROM_SELECT_SPEED
Set the CD-ROM speed

usage:

```
int speed;

ioctl(fd, CDROM_SELECT_SPEED, speed);
```

inputs:
New drive speed.

outputs:
none

error return:

- ENOSYS speed selection not supported by drive.

CDROM_SELECT_DISC
Select disc (for juke-boxes)

usage:

```
int disk;

ioctl(fd, CDROM_SELECT_DISC, disk);
```

inputs:
Disk to load into drive.

outputs:
none

error return:

- EINVAL Disk number beyond capacity of drive

CDROM_MEDIA_CHANGED
Check is media changed

usage:

```
int slot;

ioctl(fd, CDROM_MEDIA_CHANGED, slot);
```

inputs:
Slot number to be tested, always zero except for jukeboxes.

May also be special values CDSL_NONE or CDSL_CURRENT

outputs:
Ioctl return value is 0 or 1 depending on whether the media

has been changed, or -1 on error.

error returns:

- ENOSYS Drive can't detect media change
- EINVAL Slot number beyond capacity of drive
- ENOMEM Out of memory

CDROM_DRIVE_STATUS
Get tray position, etc.

usage:

```
int slot;

ioctl(fd, CDROM_DRIVE_STATUS, slot);
```

inputs:
Slot number to be tested, always zero except for jukeboxes.

May also be special values CDSL_NONE or CDSL_CURRENT

outputs:
Ioctl return value will be one of the following values

from <linux/cdrom.h>:

CDS_NO_INFO	Information not available.
CDS_NO_DISC	
CDS_TRAY_OPEN	
CDS_DRIVE_NOT_READY	
CDS_DISC_OK	
-1	error

error returns:

- ENOSYS Drive can't detect drive status
- EINVAL Slot number beyond capacity of drive
- ENOMEM Out of memory

CDROM_DISC_STATUS
Get disc type, etc.

usage:

```
ioctl(fd, CDROM_DISC_STATUS, 0);
```

inputs:
none

outputs:

Ioctl return value will be one of the following values

from <linux/cdrom.h>:

- CDS_NO_INFO
- CDS_AUDIO
- CDS_MIXED
- CDS_XA_2_2
- CDS_XA_2_1
- CDS_DATA_1

error returns:
none at present

notes:

- Source code comments state:

```
Ok, this is where problems start.  The current interface␣
↪for
the CDROM_DISC_STATUS ioctl is flawed.  It makes the␣
↪false
assumption that CDs are all CDS_DATA_1 or all CDS_AUDIO,␣
↪etc.
Unfortunately, while this is often the case, it is also
very common for CDs to have some tracks with data, and␣
```

(continues on next page)

(continued from previous page)

```
↪some
tracks with audio.      Just because I feel like it, I␣
↪declare
the following to be the best way to cope.  If the CD has
ANY data tracks on it, it will be returned as a data CD.
If it has any XA tracks, I will return it as that.     ␣
↪Now I
could simplify this interface by combining these returns␣
↪with
the above, but this more clearly demonstrates the problem
with the current interface.  Too bad this wasn't designed
to use bitmasks...             -Erik

Well, now we have the option CDS_MIXED: a mixed-type CD.
User level programmers might feel the ioctl is not very
useful.
                ---david
```

CDROM_CHANGER_NSLOTS
Get number of slots

usage:

```
ioctl(fd, CDROM_CHANGER_NSLOTS, 0);
```

inputs:
none

outputs:
The ioctl return value will be the number of slots in a CD changer. Typically 1 for non-multi-disk devices.

error returns:
none

CDROM_LOCKDOOR
lock or unlock door

usage:

```
int lock;

ioctl(fd, CDROM_LOCKDOOR, lock);
```

inputs:
Door lock flag, 1=lock, 0=unlock

outputs:
none

error returns:

- EDRIVE_CANT_DO_THIS

  Door lock function not supported.

- EBUSY

  Attempt to unlock when multiple users have the drive open and not CAP_SYS_ADMIN

notes:
As of 2.6.8.1, the lock flag is a global lock, meaning that all CD drives will be locked or unlocked together. This is probably a bug.

The EDRIVE_CANT_DO_THIS value is defined in <linux/cdrom.h> and is currently (2.6.8.1) the same as EOPNOTSUPP

CDROM_DEBUG
Turn debug messages on/off

usage:

```
int debug;

ioctl(fd, CDROM_DEBUG, debug);
```

inputs:
Cdrom debug flag, 0=disable, 1=enable

outputs:
The ioctl return value will be the new debug flag.

error return:

- EACCES Access denied: requires CAP_SYS_ADMIN

CDROM_GET_CAPABILITY
get capabilities

usage:

```
ioctl(fd, CDROM_GET_CAPABILITY, 0);
```

inputs:
none

outputs:
The ioctl return value is the current device capability flags. See CDC_CLOSE_TRAY, CDC_OPEN_TRAY, etc.

CDROMAUDIOBUFSIZ
set the audio buffer size

usage:

```
int arg;

ioctl(fd, CDROMAUDIOBUFSIZ, val);
```

inputs:
New audio buffer size

outputs:
The ioctl return value is the new audio buffer size, or -1 on error.

error return:

- ENOSYS Not supported by this driver.

notes:
Not supported by all drivers.

DVD_READ_STRUCT Read structure

usage:

```
dvd_struct s;

ioctl(fd, DVD_READ_STRUCT, &s);
```

inputs:

dvd_struct structure, containing:

type	specifies the information desired, one of DVD_STRUCT_PHYSICAL, DVD_STRUCT_COPYRIGHT, DVD_STRUCT_DISCKEY, DVD_STRUCT_BCA, DVD_STRUCT_MANUFACT
physi-cal.laye	desired layer, indexed from 0
copy-right.lay	desired layer, indexed from 0
dis-ckey.agi	

outputs:

dvd_struct structure, containing:

physical	for type == DVD_STRUCT_PHYSICAL
copyright	for type == DVD_STRUCT_COPYRIGHT
disckey.value	for type == DVD_STRUCT_DISCKEY
bca.{len,value}	for type == DVD_STRUCT_BCA
manufact.{len,valu}	for type == DVD_STRUCT_MANUFACT

error returns:

- EINVAL physical.layer_num exceeds number of layers
- EIO Received invalid response from drive

DVD_WRITE_STRUCT Write structure

Not implemented, as of 2.6.8.1

DVD_AUTH Authentication

usage:

```
dvd_authinfo ai;

ioctl(fd, DVD_AUTH, &ai);
```

inputs:
dvd_authinfo structure. See <linux/cdrom.h>

outputs:
dvd_authinfo structure.

error return:

- ENOTTY ai.type not recognized.

CDROM_SEND_PACKET
send a packet to the drive

usage:

```
struct cdrom_generic_command cgc;

ioctl(fd, CDROM_SEND_PACKET, &cgc);
```

inputs:
cdrom_generic_command structure containing the packet to send.

outputs:

none

cdrom_generic_command structure containing results.

error return:

- EIO

  command failed.

- EPERM

  Operation not permitted, either because a write command was attempted on a drive which is opened read-only, or because the command requires CAP_SYS_RAWIO

- EINVAL

  cgc.data_direction not set

CDROM_NEXT_WRITABLE
get next writable block

usage:

```
long next;

ioctl(fd, CDROM_NEXT_WRITABLE, &next);
```

inputs:
none

outputs:
The next writable block.

notes:

If the device does not support this ioctl directly, the

ioctl will return CDROM_LAST_WRITTEN + 7.

CDROM_LAST_WRITTEN
get last block written on disc

usage:

```
long last;

ioctl(fd, CDROM_LAST_WRITTEN, &last);
```

inputs:
none

outputs:
The last block written on disc

notes:
If the device does not support this ioctl directly, the result is derived from the disc's table of contents. If the table of contents can't be read, this ioctl returns an error.

6.4 Summary of *HDIO_* ioctl calls

- Edward A. Falk <efalk@google.com>

November, 2004

This document attempts to describe the ioctl(2) calls supported by the HD/IDE layer. These are by-and-large implemented (as of Linux 2.6) in drivers/ide/ide.c and drivers/block/scsi_ioctl.c

ioctl values are listed in <linux/hdreg.h>. As of this writing, they are as follows:

ioctls that pass argument pointers to user space:

HDIO_GETGEO	get device geometry
HDIO_GET_UNMASKINT:	get current unmask setting
HDIO_GET_MULTCOUNT	get current IDE blockmode setting
HDIO_GET_QDMA	get use-qdma flag
HDIO_SET_XFER	set transfer rate via proc
HDIO_OBSOLETE_IDENT	OBSOLETE, DO NOT USE
HDIO_GET_KEEPSETTIN	get keep-settings-on-reset flag
HDIO_GET_32BIT	get current io_32bit setting
HDIO_GET_NOWERR	get ignore-write-error flag
HDIO_GET_DMA	get use-dma flag
HDIO_GET_NICE	get nice flags
HDIO_GET_IDENTITY	get IDE identification info
HDIO_GET_WCACHE	get write cache mode on|off
HDIO_GET_ACOUSTIC	get acoustic value
HDIO_GET_ADDRESS	get sector addressing mode
HDIO_GET_BUSSTATE	get the bus state of the hwif
HDIO_TRISTATE_HWIF	execute a channel tristate
HDIO_DRIVE_RESET	execute a device reset
HDIO_DRIVE_TASKFILE	execute raw taskfile
HDIO_DRIVE_TASK	execute task and special drive command
HDIO_DRIVE_CMD	execute a special drive command
HDIO_DRIVE_CMD_AEB	HDIO_DRIVE_TASK

ioctls that pass non-pointer values:

HDIO_SET_MULTCOUNT	change IDE blockmode
HDIO_SET_UNMASKINTR	permit other irqs during I/O
HDIO_SET_KEEPSETTING:	keep ioctl settings on reset
HDIO_SET_32BIT	change io_32bit flags
HDIO_SET_NOWERR	change ignore-write-error flag
HDIO_SET_DMA	change use-dma flag
HDIO_SET_PIO_MODE	reconfig interface to new speed
HDIO_SCAN_HWIF	register and (re)scan interface
HDIO_SET_NICE	set nice flags
HDIO_UNREGISTER_HWIF	unregister interface
HDIO_SET_WCACHE	change write cache enable-disable
HDIO_SET_ACOUSTIC	change acoustic behavior
HDIO_SET_BUSSTATE	set the bus state of the hwif
HDIO_SET_QDMA	change use-qdma flag
HDIO_SET_ADDRESS	change lba addressing modes
HDIO_SET_IDE_SCSI	Set scsi emulation mode on/off
HDIO_SET_SCSI_IDE	not implemented yet

The information that follows was determined from reading kernel source code. It is likely that some corrections will be made over time.

General:

Unless otherwise specified, all ioctl calls return 0 on success and -1 with errno set to an appropriate value on error.

Unless otherwise specified, all ioctl calls return -1 and set errno to EFAULT on a failed attempt to copy data to or from user address space.

Unless otherwise specified, all data structures and constants are defined in <linux/hdreg.h>

HDIO_GETGEO
get device geometry

usage:

```
struct hd_geometry geom;

ioctl(fd, HDIO_GETGEO, &geom);
```

inputs:
none

outputs:

hd_geometry structure containing:

heads	number of heads
sectors	number of sectors/track
cylinders	number of cylinders, mod 65536
start	starting sector of this partition.

error returns:

- EINVAL

  if the device is not a disk drive or floppy drive, or if the user passes a null pointer

notes:
Not particularly useful with modern disk drives, whose geometry is a polite fiction anyway. Modern drives are addressed purely by sector number nowadays (lba addressing), and the drive geometry is an abstraction which is actually subject to change. Currently (as of Nov 2004), the geometry values are the "bios" values – presumably the values the drive had when Linux first booted.

In addition, the cylinders field of the hd_geometry is an unsigned short, meaning that on most architectures, this ioctl will not return a meaningful value on drives with more than 65535 tracks.

The start field is unsigned long, meaning that it will not contain a meaningful value for disks over 219 Gb in size.

HDIO_GET_UNMASKINTR
get current unmask setting

usage:

```
long val;

ioctl(fd, HDIO_GET_UNMASKINTR, &val);
```

inputs:
none

outputs:
The value of the drive's current unmask setting

HDIO_SET_UNMASKINTR
permit other irqs during I/O

usage:

```
unsigned long val;

ioctl(fd, HDIO_SET_UNMASKINTR, val);
```

inputs:
New value for unmask flag

outputs:
none

error return:

- EINVAL Called on a partition instead of the whole disk device
- EACCES Access denied: requires CAP_SYS_ADMIN
- EINVAL value out of range [0 1]
- EBUSY Controller busy

HDIO_GET_MULTCOUNT
get current IDE blockmode setting

usage:

```
long val;

ioctl(fd, HDIO_GET_MULTCOUNT, &val);
```

inputs:
none

outputs:
The value of the current IDE block mode setting. This controls how many sectors the drive will transfer per interrupt.

HDIO_SET_MULTCOUNT
change IDE blockmode

usage:

```
int val;

ioctl(fd, HDIO_SET_MULTCOUNT, val);
```

inputs:
New value for IDE block mode setting. This controls how many sectors the drive will transfer per interrupt.

outputs:
none

error return:

- EINVAL Called on a partition instead of the whole disk device
- EACCES Access denied: requires CAP_SYS_ADMIN
- EINVAL value out of range supported by disk.
- EBUSY Controller busy or blockmode already set.
- EIO Drive did not accept new block mode.

notes:
Source code comments read:

```
This is tightly woven into the driver->do_special cannot
touch.  DON'T do it again until a total personality rewrite
is committed.
```

If blockmode has already been set, this ioctl will fail with -EBUSY

HDIO_GET_QDMA
get use-qdma flag

Not implemented, as of 2.6.8.1

HDIO_SET_XFER
set transfer rate via proc

Not implemented, as of 2.6.8.1

HDIO_OBSOLETE_IDENTITY
OBSOLETE, DO NOT USE

Same as HDIO_GET_IDENTITY (see below), except that it only returns the first 142 bytes of drive identity information.

HDIO_GET_IDENTITY
get IDE identification info

usage:

```
unsigned char identity[512];

ioctl(fd, HDIO_GET_IDENTITY, identity);
```

inputs:
none

outputs:
ATA drive identity information. For full description, see the IDENTIFY DEVICE and IDENTIFY PACKET DEVICE commands in the ATA specification.

error returns:

- EINVAL Called on a partition instead of the whole disk device
- ENOMSG IDENTIFY DEVICE information not available

notes:
Returns information that was obtained when the drive was probed. Some of this information is subject to change, and this ioctl does not re-probe the drive to update the information.

This information is also available from /proc/ide/hdX/identify

HDIO_GET_KEEPSETTINGS
get keep-settings-on-reset flag

usage:

```
long val;

ioctl(fd, HDIO_GET_KEEPSETTINGS, &val);
```

inputs:
none

outputs:
The value of the current "keep settings" flag

notes:
When set, indicates that kernel should restore settings after a drive reset.

HDIO_SET_KEEPSETTINGS
keep ioctl settings on reset

usage:

```
long val;

ioctl(fd, HDIO_SET_KEEPSETTINGS, val);
```

inputs:
New value for keep_settings flag

outputs:
none

error return:

- EINVAL Called on a partition instead of the whole disk device
- EACCES Access denied: requires CAP_SYS_ADMIN
- EINVAL value out of range [0 1]
- EBUSY Controller busy

HDIO_GET_32BIT
get current io_32bit setting

usage:

```
long val;

ioctl(fd, HDIO_GET_32BIT, &val);
```

inputs:
none

outputs:
The value of the current io_32bit setting

notes:
0=16-bit, 1=32-bit, 2,3 = 32bit+sync

HDIO_GET_NOWERR
get ignore-write-error flag

usage:

```
long val;

ioctl(fd, HDIO_GET_NOWERR, &val);
```

inputs:
none

outputs:
The value of the current ignore-write-error flag

HDIO_GET_DMA
get use-dma flag

usage:

```
long val;

ioctl(fd, HDIO_GET_DMA, &val);
```

inputs:
none

outputs:
The value of the current use-dma flag

HDIO_GET_NICE
get nice flags

usage:

```
long nice;

ioctl(fd, HDIO_GET_NICE, &nice);
```

inputs:
none

outputs:
The drive's "nice" values.

notes:
Per-drive flags which determine when the system will give more bandwidth to other devices sharing the same IDE bus.

See <linux/hdreg.h>, near symbol IDE_NICE_DSC_OVERLAP.

HDIO_SET_NICE
set nice flags

usage:

```
unsigned long nice;

...
ioctl(fd, HDIO_SET_NICE, nice);
```

inputs:
bitmask of nice flags.

outputs:
none

error returns:

- EACCES Access denied: requires CAP_SYS_ADMIN
- EPERM Flags other than DSC_OVERLAP and NICE_1 set.
- EPERM DSC_OVERLAP specified but not supported by drive

notes:
This ioctl sets the DSC_OVERLAP and NICE_1 flags from values provided by the user.

Nice flags are listed in <linux/hdreg.h>, starting with IDE_NICE_DSC_OVERLAP. These values represent shifts.

HDIO_GET_WCACHE
get write cache mode on|off

usage:

```
long val;

ioctl(fd, HDIO_GET_WCACHE, &val);
```

inputs:
none

outputs:
The value of the current write cache mode

HDIO_GET_ACOUSTIC
get acoustic value

usage:

```
long val;

ioctl(fd, HDIO_GET_ACOUSTIC, &val);
```

inputs:
none

outputs:
The value of the current acoustic settings

notes:
See HDIO_SET_ACOUSTIC

HDIO_GET_ADDRESS
usage:

```
long val;

ioctl(fd, HDIO_GET_ADDRESS, &val);
```

inputs:
none

outputs:

The value of the current addressing mode:

0	28-bit
1	48-bit
2	48-bit doing 28-bit
3	64-bit

HDIO_GET_BUSSTATE
get the bus state of the hwif

usage:

```
long state;

ioctl(fd, HDIO_SCAN_HWIF, &state);
```

inputs:
none

outputs:
Current power state of the IDE bus. One of BUSSTATE_OFF, BUSSTATE_ON, or BUSSTATE_TRISTATE

error returns:

- EACCES Access denied: requires CAP_SYS_ADMIN

HDIO_SET_BUSSTATE
set the bus state of the hwif

usage:

```
int state;

...
ioctl(fd, HDIO_SCAN_HWIF, state);
```

inputs:
Desired IDE power state. One of BUSSTATE_OFF, BUSSTATE_ON, or BUSSTATE_TRISTATE

outputs:
none

error returns:

- EACCES Access denied: requires CAP_SYS_RAWIO
- EOPNOTSUPP Hardware interface does not support bus power control

HDIO_TRISTATE_HWIF
execute a channel tristate

Not implemented, as of 2.6.8.1. See HDIO_SET_BUSSTATE

HDIO_DRIVE_RESET
execute a device reset

usage:

```
int args[3]

...
ioctl(fd, HDIO_DRIVE_RESET, args);
```

inputs:
none

outputs:
none

error returns:

- EACCES Access denied: requires CAP_SYS_ADMIN
- ENXIO No such device: phy dead or ctl_addr == 0
- EIO I/O error: reset timed out or hardware error

notes:

- Execute a reset on the device as soon as the current IO operation has completed.
- Executes an ATAPI soft reset if applicable, otherwise executes an ATA soft reset on the controller.

HDIO_DRIVE_TASKFILE
execute raw taskfile

Note:
If you don't have a copy of the ANSI ATA specification handy, you should probably ignore this ioctl.

- Execute an ATA disk command directly by writing the "taskfile" registers of the drive. Requires ADMIN and RAWIO access privileges.

usage:

```
struct {

  ide_task_request_t req_task;
  u8 outbuf[OUTPUT_SIZE];
  u8 inbuf[INPUT_SIZE];
} task;
memset(&task.req_task, 0, sizeof(task.req_task));
task.req_task.out_size = sizeof(task.outbuf);
task.req_task.in_size = sizeof(task.inbuf);
...
ioctl(fd, HDIO_DRIVE_TASKFILE, &task);
...
```

inputs:

(See below for details on memory area passed to ioctl.)

io_ports[8]	values to be written to taskfile registers
hob_ports[8]	high-order bytes, for extended commands.
out_flags	flags indicating which registers are valid
in_flags	flags indicating which registers should be returned
data_phase	see below
req_cmd	command type to be executed
out_size	size of output buffer
outbuf	buffer of data to be transmitted to disk
inbuf	buffer of data to be received from disk (see [1])

outputs:

io_ports[]	values returned in the taskfile registers
hob_ports[]	high-order bytes, for extended commands.
out_flags	flags indicating which registers are valid (see [2])
in_flags	flags indicating which registers should be returned
outbuf	buffer of data to be transmitted to disk (see [1])
inbuf	buffer of data to be received from disk

error returns:

- EACCES CAP_SYS_ADMIN or CAP_SYS_RAWIO privilege not set.
- ENOMSG Device is not a disk drive.
- ENOMEM Unable to allocate memory for task

- EFAULT req_cmd == TASKFILE_IN_OUT (not implemented as of 2.6.8)
- EPERM

  req_cmd == TASKFILE_MULTI_OUT and drive multi-count not yet set.
- EIO Drive failed the command.

notes:

[1] READ THE FOLLOWING NOTES *CAREFULLY*. THIS IOCTL IS FULL OF GOTCHAS. Extreme caution should be used with using this ioctl. A mistake can easily corrupt data or hang the system.

[2] Both the input and output buffers are copied from the user and written back to the user, even when not used.

[3] If one or more bits are set in out_flags and in_flags is zero, the following values are used for in_flags.all and written back into in_flags on completion.

- IDE_TASKFILE_STD_IN_FLAGS | (IDE_HOB_STD_IN_FLAGS << 8) if LBA48 addressing is enabled for the drive
- IDE_TASKFILE_STD_IN_FLAGS if CHS/LBA28

The association between in_flags.all and each enable bitfield flips depending on endianness; fortunately, TASKFILE only uses inflags.b.data bit and ignores all other bits. The end result is that, on any endian machines, it has no effect other than modifying in_flags on completion.

[4] The default value of SELECT is (0xa0|DEV_bit|LBA_bit) except for four drives per port chipsets. For four drives per port chipsets, it's (0xa0|DEV_bit|LBA_bit) for the first pair and (0x80|DEV_bit|LBA_bit) for the second pair.

[5] The argument to the ioctl is a pointer to a region of memory containing a ide_task_request_t structure, followed by an optional buffer of data to be transmitted to the drive, followed by an optional buffer to receive data from the drive.

Command is passed to the disk drive via the ide_task_request_t structure, which contains these fields:

io_port	values for the taskfile registers
hob_po	high-order bytes, for extended commands
out_fla	flags indicating which entries in the io_ports[] and hob_ports[] arrays contain valid values. Type ide_reg_valid_t.
in_flag	flags indicating which entries in the io_ports[] and hob_ports[] arrays are expected to contain valid values on return.
data_p	See below
req_cm	Command type, see below
out_siz	output (user->drive) buffer size, bytes
in_size	input (drive->user) buffer size, bytes

When out_flags is zero, the following registers are loaded.

HOB_FE	If the drive supports LBA48
HOB_NS	If the drive supports LBA48
HOB_SE	If the drive supports LBA48
HOB_LC	If the drive supports LBA48
HOB_HC	If the drive supports LBA48
FEATURE	
NSECTOR	
SECTOR	
LCYL	
HCYL	
SELECT	First, masked with 0xE0 if LBA48, 0xEF otherwise; then, or'ed with the default value of SELECT.

If any bit in out_flags is set, the following registers are loaded.

HOB_D	If out_flags.b.data is set. HOB_DATA will travel on DD8-DD15 on little endian machines and on DD0-DD7 on big endian machines.
DATA	If out_flags.b.data is set. DATA will travel on DD0-DD7 on little endian machines and on DD8-DD15 on big endian machines.
HOB_N	If out_flags.b.nsector_hob is set
HOB_S	If out_flags.b.sector_hob is set
HOB_L	If out_flags.b.lcyl_hob is set
HOB_H	If out_flags.b.hcyl_hob is set
FEATURE	If out_flags.b.feature is set
NSECTOR	If out_flags.b.nsector is set
SECTOR	If out_flags.b.sector is set
LCYL	If out_flags.b.lcyl is set
HCYL	If out_flags.b.hcyl is set
SELECT	Or'ed with the default value of SELECT and loaded regardless of out_flags.b.select.

Taskfile registers are read back from the drive into {io|hob}_ports[] after the command completes iff one of the following conditions is met; otherwise, the original values will be written back, unchanged.

1. The drive fails the command (EIO).
2. One or more than one bits are set in out_flags.
3. The requested data_phase is TASKFILE_NO_DATA.

HOB_DA	If in_flags.b.data is set. It will contain DD8-DD15 on little endian machines and DD0-DD7 on big endian machines.
DATA	If in_flags.b.data is set. It will contain DD0-DD7 on little endian machines and DD8-DD15 on big endian machines.
HOB_FE	If the drive supports LBA48
HOB_NS	If the drive supports LBA48
HOB_SE	If the drive supports LBA48
HOB_LC	If the drive supports LBA48
HOB_HC	If the drive supports LBA48
NSECTOR	
SECTOR	
LCYL	
HCYL	

The data_phase field describes the data transfer to be performed. Value is one of:

TASKFILE_IN	
TASKFILE_MULTI_IN	
TASKFILE_OUT	
TASK-FILE_MULTI_OUT	
TASKFILE_IN_OUT	
TASKFILE_IN_DMA	
TASKFILE_IN_DMAQ	== IN_DMA (queueing not supported)
TASK-FILE_OUT_DMA	
TASK-FILE_OUT_DMAQ	== OUT_DMA (queueing not supported)
TASKFILE_P_IN	unimplemented
TASK-FILE_P_IN_DMA	unimplemented
TASK-FILE_P_IN_DMAQ	unimplemented
TASKFILE_P_OUT	unimplemented
TASK-FILE_P_OUT_DMA	unimplemented
TASK-FILE_P_OUT_DMAQ	unimplemented

The req_cmd field classifies the command type. It may be one of:

IDE_DRIVE_TASK_NO_DATA	
IDE_DRIVE_TASK_SET_XFER	unimplemented
IDE_DRIVE_TASK_IN	
IDE_DRIVE_TASK_OUT	unimplemented
IDE_DRIVE_TASK_RAW_WRITE	

[6] Do not access {in|out}_flags->all except for resetting all the bits. Always access individual bit fields. ->all value will flip depending on endianness. For the same reason, do not use IDE_{TASKFILE|HOB}_STD_{OUT|IN}_FLAGS constants defined in hdreg.h.

HDIO_DRIVE_CMD

execute a special drive command

Note: If you don't have a copy of the ANSI ATA specification handy, you should probably ignore this ioctl.

usage:

```
u8 args[4+XFER_SIZE];
```

(continues on next page)

(continued from previous page)

```
...
ioctl(fd, HDIO_DRIVE_CMD, args);
```

inputs:

Commands other than WIN_SMART:

args[0]	COMMAND
args[1]	NSECTOR
args[2]	FEATURE
args[3]	NSECTOR

WIN_SMART:

args[0]	COMMAND
args[1]	SECTOR
args[2]	FEATURE
args[3]	NSECTOR

outputs:

args[] buffer is filled with register values followed by any

data returned by the disk.

args[0]	status
args[1]	error
args[2]	NSECTOR
args[3]	undefined
args[4+]	NSECTOR * 512 bytes of data returned by the command.

error returns:

- EACCES Access denied: requires CAP_SYS_RAWIO
- ENOMEM Unable to allocate memory for task
- EIO Drive reports error

notes:

[1] For commands other than WIN_SMART, args[1] should equal args[3]. SECTOR, LCYL and HCYL are undefined. For WIN_SMART, 0x4f and 0xc2 are loaded into LCYL and HCYL respectively. In both cases SELECT will contain the default value for the drive. Please refer to HDIO_DRIVE_TASKFILE notes for the default value of SELECT.

[2] If NSECTOR value is greater than zero and the drive sets DRQ when interrupting for the command, NSECTOR * 512 bytes are read

from the device into the area following NSECTOR. In the above example, the area would be args[4..4+XFER_SIZE]. 16bit PIO is used regardless of HDIO_SET_32BIT setting.

[3] If COMMAND == WIN_SETFEATURES && FEATURE == SETFEATURES_XFER && NSECTOR >= XFER_SW_DMA_0 && the drive supports any DMA mode, IDE driver will try to tune the transfer mode of the drive accordingly.

HDIO_DRIVE_TASK

execute task and special drive command

Note: If you don't have a copy of the ANSI ATA specification handy, you should probably ignore this ioctl.

usage:

```
u8 args[7];

...
ioctl(fd, HDIO_DRIVE_TASK, args);
```

inputs:

Taskfile register values:

args[0]	COMMAND
args[1]	FEATURE
args[2]	NSECTOR
args[3]	SECTOR
args[4]	LCYL
args[5]	HCYL
args[6]	SELECT

outputs:

Taskfile register values:

args[0]	status
args[1]	error
args[2]	NSECTOR
args[3]	SECTOR
args[4]	LCYL
args[5]	HCYL
args[6]	SELECT

error returns:

- EACCES Access denied: requires CAP_SYS_RAWIO
- ENOMEM Unable to allocate memory for task
- ENOMSG Device is not a disk drive.
- EIO Drive failed the command.

notes:

[1] DEV bit (0x10) of SELECT register is ignored and the appropriate value for the drive is used. All other bits are used unaltered.

HDIO_DRIVE_CMD_AEB
HDIO_DRIVE_TASK

Not implemented, as of 2.6.8.1

HDIO_SET_32BIT
change io_32bit flags

usage:

```
int val;

ioctl(fd, HDIO_SET_32BIT, val);
```

inputs:
New value for io_32bit flag

outputs:
none

error return:

- EINVAL Called on a partition instead of the whole disk device
- EACCES Access denied: requires CAP_SYS_ADMIN
- EINVAL value out of range [0 3]
- EBUSY Controller busy

HDIO_SET_NOWERR
change ignore-write-error flag

usage:

```
int val;

ioctl(fd, HDIO_SET_NOWERR, val);
```

inputs:

New value for ignore-write-error flag. Used for ignoring

WRERR_STAT

outputs:
none

error return:

- EINVAL Called on a partition instead of the whole disk device
- EACCES Access denied: requires CAP_SYS_ADMIN
- EINVAL value out of range [0 1]
- EBUSY Controller busy

HDIO_SET_DMA
change use-dma flag

usage:

```
long val;

ioctl(fd, HDIO_SET_DMA, val);
```

inputs:
New value for use-dma flag

outputs:
none

error return:

- EINVAL Called on a partition instead of the whole disk device
- EACCES Access denied: requires CAP_SYS_ADMIN
- EINVAL value out of range [0 1]
- EBUSY Controller busy

HDIO_SET_PIO_MODE
reconfig interface to new speed

usage:

```
long val;

ioctl(fd, HDIO_SET_PIO_MODE, val);
```

inputs:
New interface speed.

outputs:
none

error return:

- EINVAL Called on a partition instead of the whole disk device
- EACCES Access denied: requires CAP_SYS_ADMIN
- EINVAL value out of range [0 255]
- EBUSY Controller busy

HDIO_SCAN_HWIF
register and (re)scan interface

usage:

```
int args[3]

...
ioctl(fd, HDIO_SCAN_HWIF, args);
```

inputs:

args[0]	io address to probe
args[1]	control address to probe
args[2]	irq number

outputs:
none

error returns:

- EACCES Access denied: requires CAP_SYS_RAWIO
- EIO Probe failed.

notes:
This ioctl initializes the addresses and irq for a disk controller, probes for drives, and creates /proc/ide interfaces as appropriate.

HDIO_UNREGISTER_HWIF
unregister interface

usage:

```
int index;

ioctl(fd, HDIO_UNREGISTER_HWIF, index);
```

inputs:
index index of hardware interface to unregister

outputs:
none

error returns:

- EACCES Access denied: requires CAP_SYS_RAWIO

notes:
This ioctl removes a hardware interface from the kernel.

Currently (2.6.8) this ioctl silently fails if any drive on the interface is busy.

HDIO_SET_WCACHE
change write cache enable-disable

usage:

```
int val;

ioctl(fd, HDIO_SET_WCACHE, val);
```

inputs:
New value for write cache enable

outputs:
none

error return:

- EINVAL Called on a partition instead of the whole disk device
- EACCES Access denied: requires CAP_SYS_ADMIN
- EINVAL value out of range [0 1]
- EBUSY Controller busy

HDIO_SET_ACOUSTIC
change acoustic behavior

usage:

```
int val;

ioctl(fd, HDIO_SET_ACOUSTIC, val);
```

inputs:
New value for drive acoustic settings

outputs:
none

error return:

- EINVAL Called on a partition instead of the whole disk device
- EACCES Access denied: requires CAP_SYS_ADMIN
- EINVAL value out of range [0 254]
- EBUSY Controller busy

HDIO_SET_QDMA
change use-qdma flag

Not implemented, as of 2.6.8.1

HDIO_SET_ADDRESS
change lba addressing modes

usage:

```
int val;

ioctl(fd, HDIO_SET_ADDRESS, val);
```

inputs:

New value for addressing mode

0	28-bit
1	48-bit
2	48-bit doing 28-bit

outputs:
none

error return:

- EINVAL Called on a partition instead of the whole disk device
- EACCES Access denied: requires CAP_SYS_ADMIN
- EINVAL value out of range [0 2]
- EBUSY Controller busy
- EIO Drive does not support lba48 mode.

HDIO_SET_IDE_SCSI

usage:

```
long val;

ioctl(fd, HDIO_SET_IDE_SCSI, val);
```

inputs:

New value for scsi emulation mode (?)

outputs:

none

error return:

- EINVAL Called on a partition instead of the whole disk device
- EACCES Access denied: requires CAP_SYS_ADMIN
- EINVAL value out of range [0 1]
- EBUSY Controller busy

HDIO_SET_SCSI_IDE

Not implemented, as of 2.6.8.1

CHAPTER

SEVEN

IOMMU USERSPACE API

IOMMU UAPI is used for virtualization cases where communications are needed between physical and virtual IOMMU drivers. For baremetal usage, the IOMMU is a system device which does not need to communicate with userspace directly.

The primary use cases are guest Shared Virtual Address (SVA) and guest IO virtual address (IOVA), wherein the vIOMMU implementation relies on the physical IOMMU and for this reason requires interactions with the host driver.

7.1 Functionalities

Communications of user and kernel involve both directions. The supported user-kernel APIs are as follows:

1. Bind/Unbind guest PASID (e.g. Intel VT-d)
2. Bind/Unbind guest PASID table (e.g. ARM SMMU)
3. Invalidate IOMMU caches upon guest requests
4. Report errors to the guest and serve page requests

7.2 Requirements

The IOMMU UAPIs are generic and extensible to meet the following requirements:

1. Emulated and para-virtualised vIOMMUs
2. Multiple vendors (Intel VT-d, ARM SMMU, etc.)
3. Extensions to the UAPI shall not break existing userspace

7.3 Interfaces

Although the data structures defined in IOMMU UAPI are self-contained, there are no user API functions introduced. Instead, IOMMU UAPI is designed to work with existing user driver frameworks such as VFIO.

7.3.1 Extension Rules & Precautions

When IOMMU UAPI gets extended, the data structures can *only* be modified in two ways:

1. Adding new fields by re-purposing the padding[] field. No size change.
2. Adding new union members at the end. May increase the structure sizes.

No new fields can be added *after* the variable sized union in that it will break backward compatibility when offset moves. A new flag must be introduced whenever a change affects the structure using either method. The IOMMU driver processes the data based on flags which ensures backward compatibility.

Version field is only reserved for the unlikely event of UAPI upgrade at its entirety.

It's *always* the caller's responsibility to indicate the size of the structure passed by setting argsz appropriately. Though at the same time, argsz is user provided data which is not trusted. The argsz field allows the user app to indicate how much data it is providing; it's still the kernel's responsibility to validate whether it's correct and sufficient for the requested operation.

7.3.2 Compatibility Checking

When IOMMU UAPI extension results in some structure size increase, IOMMU UAPI code shall handle the following cases:

1. User and kernel has exact size match
2. An older user with older kernel header (smaller UAPI size) running on a newer kernel (larger UAPI size)
3. A newer user with newer kernel header (larger UAPI size) running on an older kernel.
4. A malicious/misbehaving user passing illegal/invalid size but within range. The data may contain garbage.

7.3.3 Feature Checking

While launching a guest with vIOMMU, it is strongly advised to check the compatibility upfront, as some subsequent errors happening during vIOMMU operation, such as cache invalidation failures cannot be nicely escalated to the guest due to IOMMU specifications. This can lead to catastrophic failures for the users.

User applications such as QEMU are expected to import kernel UAPI headers. Backward compatibility is supported per feature flags. For example, an older QEMU (with older kernel header) can run on newer kernel. Newer QEMU (with new kernel header) may refuse to initialize on an older kernel if new feature flags are not supported by older kernel. Simply recompiling existing code with newer kernel header should not be an issue in that only existing flags are used.

IOMMU vendor driver should report the below features to IOMMU UAPI consumers (e.g. via VFIO).

1. IOMMU_NESTING_FEAT_SYSWIDE_PASID
2. IOMMU_NESTING_FEAT_BIND_PGTBL
3. IOMMU_NESTING_FEAT_BIND_PASID_TABLE
4. IOMMU_NESTING_FEAT_CACHE_INVLD
5. IOMMU_NESTING_FEAT_PAGE_REQUEST

Take VFIO as example, upon request from VFIO userspace (e.g. QEMU), VFIO kernel code shall query IOMMU vendor driver for the support of the above features. Query result can then be reported back to the userspace caller. Details can be found in Documentation/driver-api/vfio.rst.

7.3.4 Data Passing Example with VFIO

As the ubiquitous userspace driver framework, VFIO is already IOMMU aware and shares many key concepts such as device model, group, and protection domain. Other user driver frameworks can also be extended to support IOMMU UAPI but it is outside the scope of this document.

In this tight-knit VFIO-IOMMU interface, the ultimate consumer of the IOMMU UAPI data is the host IOMMU driver. VFIO facilitates user-kernel transport, capability checking, security, and life cycle management of process address space ID (PASID).

VFIO layer conveys the data structures down to the IOMMU driver. It follows the pattern below:

```
struct {
     __u32 argsz;
     __u32 flags;
     __u8  data[];
};
```

Here data[] contains the IOMMU UAPI data structures. VFIO has the freedom to bundle the data as well as parse data size based on its own flags.

In order to determine the size and feature set of the user data, argsz and flags (or the equivalent) are also embedded in the IOMMU UAPI data structures.

A "__u32 argsz" field is *always* at the beginning of each structure.

For example:

```
struct iommu_cache_invalidate_info {
     __u32   argsz;
     #define IOMMU_CACHE_INVALIDATE_INFO_VERSION_1 1
     __u32   version;
     /* IOMMU paging structure cache */
     #define IOMMU_CACHE_INV_TYPE_IOTLB      (1 << 0) /* IOMMU␣
↪IOTLB */
     #define IOMMU_CACHE_INV_TYPE_DEV_IOTLB  (1 << 1) /* Device␣
↪IOTLB */
     #define IOMMU_CACHE_INV_TYPE_PASID      (1 << 2) /* PASID␣
↪cache */
     #define IOMMU_CACHE_INV_TYPE_NR         (3)
     __u8    cache;
     __u8    granularity;
     __u8    padding[6];
     union {
             struct iommu_inv_pasid_info pasid_info;
             struct iommu_inv_addr_info addr_info;
     } granu;
};
```

VFIO is responsible for checking its own argsz and flags. It then invokes appropriate IOMMU UAPI functions. The user pointers are passed to the IOMMU layer for further processing. The responsibilities are divided as follows:

- Generic IOMMU layer checks argsz range based on UAPI data in the current kernel version.
- Generic IOMMU layer checks content of the UAPI data for non-zero reserved bits in flags, padding fields, and unsupported version. This is to ensure not breaking userspace in the future when these fields or flags are used.
- Vendor IOMMU driver checks argsz based on vendor flags. UAPI data is consumed based on flags. Vendor driver has access to unadulterated argsz value in case of vendor specific future extensions. Currently, it does not perform the copy_from_user() itself. A __user pointer can be provided in some future scenarios where there's vendor data outside of the structure definition.

IOMMU code treats UAPI data in two categories:

- structure contains vendor data (Example: iommu_uapi_cache_invalidate())
- structure contains only generic data (Example: iommu_uapi_sva_bind_gpasid())

7.3.5 Sharing UAPI with in-kernel users

For UAPIs that are shared with in-kernel users, a wrapper function is provided to distinguish the callers. For example,

Userspace caller

```
int iommu_uapi_sva_unbind_gpasid(struct iommu_domain *domain,
                                 struct device *dev,
                                 void __user *udata)
```

In-kernel caller

```
int iommu_sva_unbind_gpasid(struct iommu_domain *domain,
                            struct device *dev, ioasid_t ioasid);
```

CHAPTER

EIGHT

LINUX MEDIA INFRASTRUCTURE USERSPACE API

This section contains the driver development information and Kernel APIs used by media devices.

Please see:

- **/admin-guide/media/index**
  for usage information about media subsystem and supported drivers;
- **/driver-api/media/index**
  for driver development information and Kernel APIs used by media devices;

8.1 Introduction

This document covers the Linux Kernel to Userspace API's used by video and radio streaming devices, including video cameras, analog and digital TV receiver cards, AM/FM receiver cards, Software Defined Radio (SDR), streaming capture and output devices, codec devices and remote controllers.

A typical media device hardware is shown at *Typical Media Device*.

The media infrastructure API was designed to control such devices. It is divided into five parts.

1. The *first part* covers radio, video capture and output, cameras, analog TV devices and codecs.
2. The *second part* covers the API used for digital TV and Internet reception via one of the several digital tv standards. While it is called as DVB API, in fact it covers several different video standards including DVB-T/T2, DVB-S/S2, DVB-C, ATSC, ISDB-T, ISDB-S, DTMB, etc. The complete list of supported standards can be found at `fe_delivery_system`.
3. The *third part* covers the Remote Controller API.
4. The *fourth part* covers the Media Controller API.
5. The *fifth part* covers the CEC (Consumer Electronics Control) API.

It should also be noted that a media device may also have audio components, like mixers, PCM capture, PCM playback, etc, which are controlled via ALSA API. For additional information and for the latest development code, see: https://linuxtv.

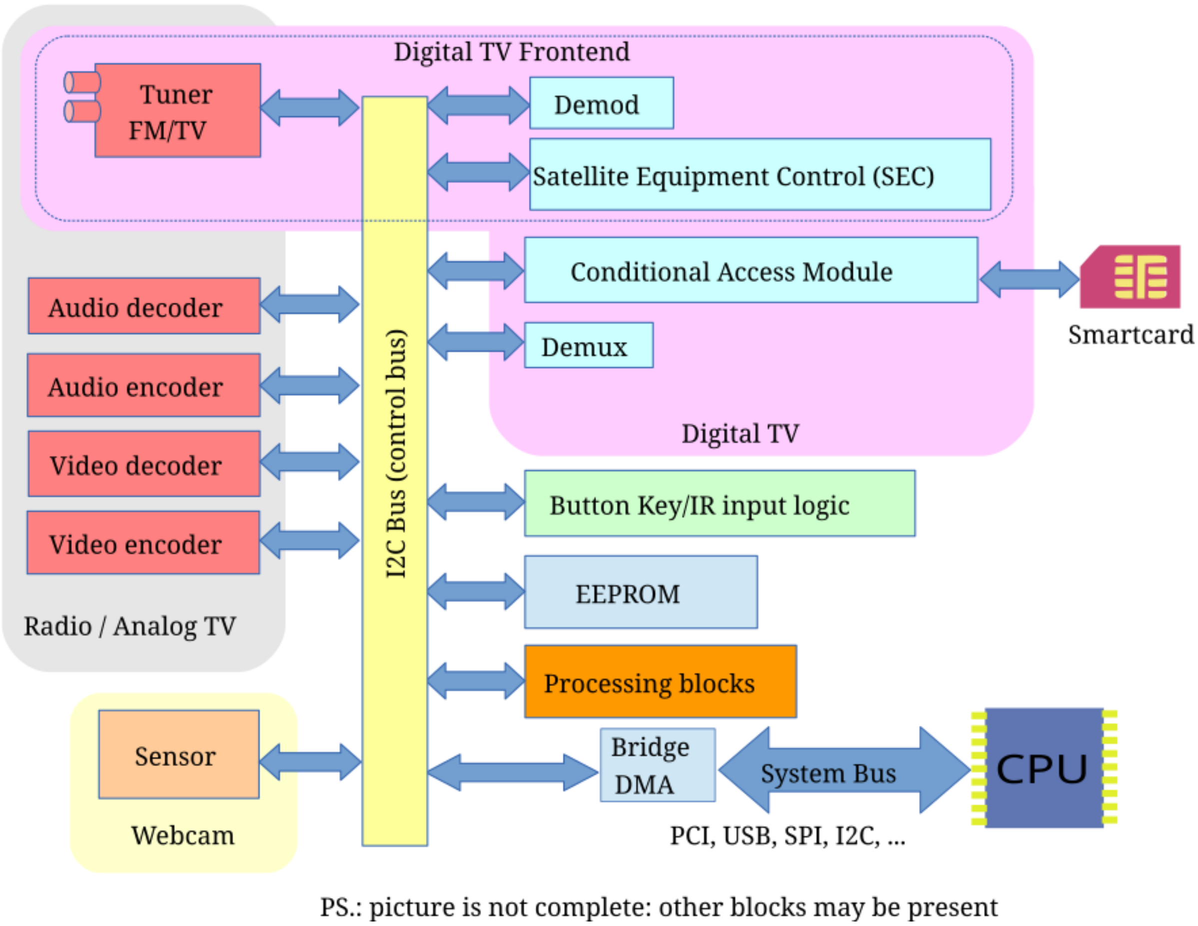

Fig. 1: Typical Media Device

org. For discussing improvements, reporting troubles, sending new drivers, etc, please mail to: Linux Media Mailing List (LMML).

8.2 Part I - Video for Linux API

This part describes the Video for Linux API version 2 (V4L2 API) specification.

Revision 4.5

8.2.1 Common API Elements

Programming a V4L2 device consists of these steps:

- Opening the device
- Changing device properties, selecting a video and audio input, video standard, picture brightness a. o.
- Negotiating a data format
- Negotiating an input/output method
- The actual input/output loop
- Closing the device

In practice most steps are optional and can be executed out of order. It depends on the V4L2 device type, you can read about the details in *Interfaces*. In this chapter we will discuss the basic concepts applicable to all devices.

Opening and Closing Devices

Controlling a hardware peripheral via V4L2

Hardware that is supported using the V4L2 uAPI often consists of multiple devices or peripherals, each of which have their own driver.

The bridge driver exposes one or more V4L2 device nodes (see *V4L2 Device Node Naming*).

There are other drivers providing support for other components of the hardware, which may also expose device nodes, called V4L2 sub-devices.

When such V4L2 sub-devices are exposed, they allow controlling those other hardware components - usually connected via a serial bus (like I^2C, SMBus or SPI). Depending on the bridge driver, those sub-devices can be controlled indirectly via the bridge driver or explicitly via the *Media Controller* and via the *V4L2 sub-devices*.

The devices that require the use of the *Media Controller* are called **MC-centric** devices. The devices that are fully controlled via V4L2 device nodes are called **video-node-centric**.

Userspace can check if a V4L2 hardware peripheral is MC-centric by calling *ioctl VIDIOC_QUERYCAP* and checking the *device_caps field*.

If the device returns `V4L2_CAP_IO_MC` flag at `device_caps`, then it is MC-centric, otherwise, it is video-node-centric.

It is required for MC-centric drivers to identify the V4L2 sub-devices and to configure the pipelines via the *media controller API* before using the peripheral. Also, the sub-devices' configuration shall be controlled via the *sub-device API*.

Note:

> A video-node-centric may still provide media-controller and sub-device interfaces as well.

However, in that case the media-controller and the sub-device interfaces are read-only and just provide information about the device. The actual configuration is done via the video nodes.

V4L2 Device Node Naming

V4L2 drivers are implemented as kernel modules, loaded manually by the system administrator or automatically when a device is first discovered. The driver modules plug into the `videodev` kernel module. It provides helper functions and a common application interface specified in this document.

Each driver thus loaded registers one or more device nodes with major number 81. Minor numbers are allocated dynamically unless the kernel is compiled with the kernel option CONFIG_VIDEO_FIXED_MINOR_RANGES. In that case minor numbers are allocated in ranges depending on the device node type.

The device nodes supported by the Video4Linux subsystem are:

Default device node name	Usage
`/dev/videoX`	Video and metadata for capture/output devices
`/dev/vbiX`	Vertical blank data (i.e. closed captions, teletext)
`/dev/radioX`	Radio tuners and modulators
`/dev/swradioX`	Software Defined Radio tuners and modulators
`/dev/v4l-touchX`	Touch sensors
`/dev/v4l-subdevX`	Video sub-devices (used by sensors and other components of the hardware peripheral)[1]

Where `X` is a non-negative integer.

Note:

1. The actual device node name is system-dependent, as udev rules may apply.

[1] **V4L2 sub-device nodes** (e. g. `/dev/v4l-subdevX`) use a different set of system calls, as covered at *Sub-device Interface*.

2. There is no guarantee that X will remain the same for the same device, as the number depends on the device driver’s probe order. If you need an unique name, udev default rules produce /dev/v4l/by-id/ and /dev/v4l/by-path/ directories containing links that can be used uniquely to identify a V4L2 device node:

```
$ tree /dev/v4l
/dev/v4l
├── by-id
│   └── usb-OmniVision._USB_Camera-B4.04.27.1-video-index0 -> ..
↪/../video0
└── by-path
    └── pci-0000:00:14.0-usb-0:2:1.0-video-index0 -> ../../
↪video0
```

Many drivers support “video_nr”, “radio_nr” or “vbi_nr” module options to select specific video/radio/vbi node numbers. This allows the user to request that the device node is named e.g. /dev/video5 instead of leaving it to chance. When the driver supports multiple devices of the same type more than one device node number can be assigned, separated by commas:

```
# modprobe mydriver video_nr=0,1 radio_nr=0,1
```

In /etc/modules.conf this may be written as:

```
options mydriver video_nr=0,1 radio_nr=0,1
```

When no device node number is given as module option the driver supplies a default.

Normally udev will create the device nodes in /dev automatically for you. If udev is not installed, then you need to enable the CONFIG_VIDEO_FIXED_MINOR_RANGES kernel option in order to be able to correctly relate a minor number to a device node number. I.e., you need to be certain that minor number 5 maps to device node name video5. With this kernel option different device types have different minor number ranges. These ranges are listed in *Interfaces*.

The creation of character special files (with mknod) is a privileged operation and devices cannot be opened by major and minor number. That means applications cannot *reliably* scan for loaded or installed drivers. The user must enter a device name, or the application can try the conventional device names.

Related Devices

Devices can support several functions. For example video capturing, VBI capturing and radio support.

The V4L2 API creates different V4L2 device nodes for each of these functions.

The V4L2 API was designed with the idea that one device node could support all functions. However, in practice this never worked: this 'feature' was never used by applications and many drivers did not support it and if they did it was certainly never tested. In addition, switching a device node between different functions only works when using the streaming I/O API, not with the *read()*/*write()* API.

Today each V4L2 device node supports just one function.

Besides video input or output the hardware may also support audio sampling or playback. If so, these functions are implemented as ALSA PCM devices with optional ALSA audio mixer devices.

One problem with all these devices is that the V4L2 API makes no provisions to find these related V4L2 device nodes. Some really complex hardware use the Media Controller (see *Part IV - Media Controller API*) which can be used for this purpose. But several drivers do not use it, and while some code exists that uses sysfs to discover related V4L2 device nodes (see libmedia_dev in the v4l-utils git repository), there is no library yet that can provide a single API towards both Media Controller-based devices and devices that do not use the Media Controller. If you want to work on this please write to the linux-media mailing list: https://linuxtv.org/lists.php.

Multiple Opens

V4L2 devices can be opened more than once.[2] When this is supported by the driver, users can for example start a "panel" application to change controls like brightness or audio volume, while another application captures video and audio. In other words, panel applications are comparable to an ALSA audio mixer application. Just opening a V4L2 device should not change the state of the device.[3]

Once an application has allocated the memory buffers needed for streaming data (by calling the *ioctl VIDIOC_REQBUFS* or *ioctl VIDIOC_CREATE_BUFS* ioctls, or implicitly by calling the *read()* or *write()* functions) that application (filehandle) becomes the owner of the device. It is no longer allowed to make changes that would affect the buffer sizes (e.g. by calling the *VIDIOC_S_FMT* ioctl) and other applications are no longer allowed to allocate buffers or start or stop streaming. The EBUSY error code will be returned instead.

Merely opening a V4L2 device does not grant exclusive access.[4] Initiating data

[2] There are still some old and obscure drivers that have not been updated to allow for multiple opens. This implies that for such drivers *open()* can return an EBUSY error code when the device is already in use.

[3] Unfortunately, opening a radio device often switches the state of the device to radio mode in many drivers. This behavior should be fixed eventually as it violates the V4L2 specification.

[4] Drivers could recognize the `O_EXCL` open flag. Presently this is not required, so applications cannot know if it really works.

exchange however assigns the right to read or write the requested type of data, and to change related properties, to this file descriptor. Applications can request additional access privileges using the priority mechanism described in *Application Priority*.

Shared Data Streams

V4L2 drivers should not support multiple applications reading or writing the same data stream on a device by copying buffers, time multiplexing or similar means. This is better handled by a proxy application in user space.

Functions

To open and close V4L2 devices applications use the *open()* and *close()* function, respectively. Devices are programmed using the *ioctl()* function as explained in the following sections.

Querying Capabilities

Because V4L2 covers a wide variety of devices not all aspects of the API are equally applicable to all types of devices. Furthermore devices of the same type have different capabilities and this specification permits the omission of a few complicated and less important parts of the API.

The *ioctl VIDIOC_QUERYCAP* ioctl is available to check if the kernel device is compatible with this specification, and to query the *functions* and *I/O methods* supported by the device.

Starting with kernel version 3.1, *ioctl VIDIOC_QUERYCAP* will return the V4L2 API version used by the driver, with generally matches the Kernel version. There's no need of using *ioctl VIDIOC_QUERYCAP* to check if a specific ioctl is supported, the V4L2 core now returns ENOTTY if a driver doesn't provide support for an ioctl.

Other features can be queried by calling the respective ioctl, for example *ioctl VIDIOC_ENUMINPUT* to learn about the number, types and names of video connectors on the device. Although abstraction is a major objective of this API, the *ioctl VIDIOC_QUERYCAP* ioctl also allows driver specific applications to reliably identify the driver.

All V4L2 drivers must support *ioctl VIDIOC_QUERYCAP*. Applications should always call this ioctl after opening the device.

Application Priority

When multiple applications share a device it may be desirable to assign them different priorities. Contrary to the traditional "rm -rf /" school of thought, a video recording application could for example block other applications from changing video controls or switching the current TV channel. Another objective is to permit low priority applications working in background, which can be preempted by user controlled applications and automatically regain control of the device at a later time.

Since these features cannot be implemented entirely in user space V4L2 defines the *VIDIOC_G_PRIORITY* and *VIDIOC_S_PRIORITY* ioctls to request and query the access priority associate with a file descriptor. Opening a device assigns a medium priority, compatible with earlier versions of V4L2 and drivers not supporting these ioctls. Applications requiring a different priority will usually call *VIDIOC_S_PRIORITY* after verifying the device with the *ioctl VIDIOC_QUERYCAP* ioctl.

Ioctls changing driver properties, such as *VIDIOC_S_INPUT*, return an `EBUSY` error code after another application obtained higher priority.

Video Inputs and Outputs

Video inputs and outputs are physical connectors of a device. These can be for example: RF connectors (antenna/cable), CVBS a.k.a. Composite Video, S-Video and RGB connectors. Camera sensors are also considered to be a video input. Video and VBI capture devices have inputs. Video and VBI output devices have outputs, at least one each. Radio devices have no video inputs or outputs.

To learn about the number and attributes of the available inputs and outputs applications can enumerate them with the *ioctl VIDIOC_ENUMINPUT* and *ioctl VIDIOC_ENUMOUTPUT* ioctl, respectively. The struct `v4l2_input` returned by the *ioctl VIDIOC_ENUMINPUT* ioctl also contains signal status information applicable when the current video input is queried.

The *VIDIOC_G_INPUT* and *VIDIOC_G_OUTPUT* ioctls return the index of the current video input or output. To select a different input or output applications call the *VIDIOC_S_INPUT* and *VIDIOC_S_OUTPUT* ioctls. Drivers must implement all the input ioctls when the device has one or more inputs, all the output ioctls when the device has one or more outputs.

Example: Information about the current video input

```
struct v4l2_input input;
int index;

if (-1 == ioctl(fd, VIDIOC_G_INPUT, &index)) {
    perror("VIDIOC_G_INPUT");
    exit(EXIT_FAILURE);
}

```

(continues on next page)

(continued from previous page)

```
memset(&input, 0, sizeof(input));
input.index = index;

if (-1 == ioctl(fd, VIDIOC_ENUMINPUT, &input)) {
    perror("VIDIOC_ENUMINPUT");
    exit(EXIT_FAILURE);
}

printf("Current input: %s\\n", input.name);
```

Example: Switching to the first video input

```
int index;

index = 0;

if (-1 == ioctl(fd, VIDIOC_S_INPUT, &index)) {
    perror("VIDIOC_S_INPUT");
    exit(EXIT_FAILURE);
}
```

Audio Inputs and Outputs

Audio inputs and outputs are physical connectors of a device. Video capture devices have inputs, output devices have outputs, zero or more each. Radio devices have no audio inputs or outputs. They have exactly one tuner which in fact *is* an audio source, but this API associates tuners with video inputs or outputs only, and radio devices have none of these.[1] A connector on a TV card to loop back the received audio signal to a sound card is not considered an audio output.

Audio and video inputs and outputs are associated. Selecting a video source also selects an audio source. This is most evident when the video and audio source is a tuner. Further audio connectors can combine with more than one video input or output. Assumed two composite video inputs and two audio inputs exist, there may be up to four valid combinations. The relation of video and audio connectors is defined in the `audioset` field of the respective struct `v4l2_input` or struct `v4l2_output`, where each bit represents the index number, starting at zero, of one audio input or output.

To learn about the number and attributes of the available inputs and outputs applications can enumerate them with the *ioctl VIDIOC_ENUMAUDIO* and *VIDIOC_ENUMAUDOUT* ioctl, respectively. The struct `v4l2_audio` returned by the *ioctl VIDIOC_ENUMAUDIO* ioctl also contains signal status information applicable when the current audio input is queried.

The *VIDIOC_G_AUDIO* and *VIDIOC_G_AUDOUT* ioctls report the current audio input and output, respectively.

[1] Actually struct `v4l2_audio` ought to have a `tuner` field like struct `v4l2_input`, not only making the API more consistent but also permitting radio devices with multiple tuners.

Note: Note that, unlike *VIDIOC_G_INPUT* and *VIDIOC_G_OUTPUT* these ioctls return a structure as *ioctl VIDIOC_ENUMAUDIO* and *VIDIOC_ENUMAUDOUT* do, not just an index.

To select an audio input and change its properties applications call the *VIDIOC_S_AUDIO* ioctl. To select an audio output (which presently has no changeable properties) applications call the *VIDIOC_S_AUDOUT* ioctl.

Drivers must implement all audio input ioctls when the device has multiple selectable audio inputs, all audio output ioctls when the device has multiple selectable audio outputs. When the device has any audio inputs or outputs the driver must set the `V4L2_CAP_AUDIO` flag in the struct `v4l2_capability` returned by the *ioctl VIDIOC_QUERYCAP* ioctl.

Example: Information about the current audio input

```
struct v4l2_audio audio;

memset(&audio, 0, sizeof(audio));

if (-1 == ioctl(fd, VIDIOC_G_AUDIO, &audio)) {
    perror("VIDIOC_G_AUDIO");
    exit(EXIT_FAILURE);
}

printf("Current input: %s\\n", audio.name);
```

Example: Switching to the first audio input

```
struct v4l2_audio audio;

memset(&audio, 0, sizeof(audio)); /* clear audio.mode, audio.
↪reserved */

audio.index = 0;

if (-1 == ioctl(fd, VIDIOC_S_AUDIO, &audio)) {
    perror("VIDIOC_S_AUDIO");
    exit(EXIT_FAILURE);
}
```

Tuners and Modulators

Tuners

Video input devices can have one or more tuners demodulating a RF signal. Each tuner is associated with one or more video inputs, depending on the number of RF connectors on the tuner. The `type` field of the respective struct `v4l2_input` returned by the *ioctl VIDIOC_ENUMINPUT* ioctl is set to `V4L2_INPUT_TYPE_TUNER` and its `tuner` field contains the index number of the tuner.

Radio input devices have exactly one tuner with index zero, no video inputs.

To query and change tuner properties applications use the *VIDIOC_G_TUNER* and *VIDIOC_S_TUNER* ioctls, respectively. The struct `v4l2_tuner` returned by *VIDIOC_G_TUNER* also contains signal status information applicable when the tuner of the current video or radio input is queried.

Note: *VIDIOC_S_TUNER* does not switch the current tuner, when there is more than one. The tuner is solely determined by the current video input. Drivers must support both ioctls and set the `V4L2_CAP_TUNER` flag in the struct `v4l2_capability` returned by the *ioctl VIDIOC_QUERYCAP* ioctl when the device has one or more tuners.

Modulators

Video output devices can have one or more modulators, that modulate a video signal for radiation or connection to the antenna input of a TV set or video recorder. Each modulator is associated with one or more video outputs, depending on the number of RF connectors on the modulator. The `type` field of the respective struct `v4l2_output` returned by the *ioctl VIDIOC_ENUMOUTPUT* ioctl is set to `V4L2_OUTPUT_TYPE_MODULATOR` and its `modulator` field contains the index number of the modulator.

Radio output devices have exactly one modulator with index zero, no video outputs.

A video or radio device cannot support both a tuner and a modulator. Two separate device nodes will have to be used for such hardware, one that supports the tuner functionality and one that supports the modulator functionality. The reason is a limitation with the *VIDIOC_S_FREQUENCY* ioctl where you cannot specify whether the frequency is for a tuner or a modulator.

To query and change modulator properties applications use the *VIDIOC_G_MODULATOR* and *VIDIOC_S_MODULATOR* ioctl. Note that *VIDIOC_S_MODULATOR* does not switch the current modulator, when there is more than one at all. The modulator is solely determined by the current video output. Drivers must support both ioctls and set the `V4L2_CAP_MODULATOR` flag in the struct `v4l2_capability` returned by the *ioctl VIDIOC_QUERYCAP* ioctl when the device has one or more modulators.

Radio Frequency

To get and set the tuner or modulator radio frequency applications use the *VIDIOC_G_FREQUENCY* and *VIDIOC_S_FREQUENCY* ioctl which both take a pointer to a struct `v4l2_frequency`. These ioctls are used for TV and radio devices alike. Drivers must support both ioctls when the tuner or modulator ioctls are supported, or when the device is a radio device.

Video Standards

Video devices typically support one or more different video standards or variations of standards. Each video input and output may support another set of standards. This set is reported by the `std` field of struct `v4l2_input` and struct `v4l2_output` returned by the *ioctl VIDIOC_ENUMINPUT* and *ioctl VIDIOC_ENUMOUTPUT* ioctls, respectively.

V4L2 defines one bit for each analog video standard currently in use worldwide, and sets aside bits for driver defined standards, e. g. hybrid standards to watch NTSC video tapes on PAL TVs and vice versa. Applications can use the predefined bits to select a particular standard, although presenting the user a menu of supported standards is preferred. To enumerate and query the attributes of the supported standards applications use the *ioctl VIDIOC_ENUMSTD, VIDIOC_SUBDEV_ENUMSTD* ioctl.

Many of the defined standards are actually just variations of a few major standards. The hardware may in fact not distinguish between them, or do so internal and switch automatically. Therefore enumerated standards also contain sets of one or more standard bits.

Assume a hypothetic tuner capable of demodulating B/PAL, G/PAL and I/PAL signals. The first enumerated standard is a set of B and G/PAL, switched automatically depending on the selected radio frequency in UHF or VHF band. Enumeration gives a "PAL-B/G" or "PAL-I" choice. Similar a Composite input may collapse standards, enumerating "PAL-B/G/H/I", "NTSC-M" and "SECAM-D/K".[1]

To query and select the standard used by the current video input or output applications call the *VIDIOC_G_STD* and *VIDIOC_S_STD* ioctl, respectively. The *received* standard can be sensed with the *ioctl VIDIOC_QUERYSTD, VIDIOC_SUBDEV_QUERYSTD* ioctl.

Note: The parameter of all these ioctls is a pointer to a *v4l2_std_id* type (a standard set), *not* an index into the standard enumeration. Drivers must implement all video standard ioctls when the device has one or more video inputs or outputs.

Special rules apply to devices such as USB cameras where the notion of video standards makes little sense. More generally for any capture or output device which is:

[1] Some users are already confused by technical terms PAL, NTSC and SECAM. There is no point asking them to distinguish between B, G, D, or K when the software or hardware can do that automatically.

- incapable of capturing fields or frames at the nominal rate of the video standard, or
- that does not support the video standard formats at all.

Here the driver shall set the std field of struct v4l2_input and struct v4l2_output to zero and the *VIDIOC_G_STD*, *VIDIOC_S_STD*, *ioctl VIDIOC_QUERYSTD*, *VIDIOC_SUBDEV_QUERYSTD* and *ioctl VIDIOC_ENUMSTD*, *VIDIOC_SUBDEV_ENUMSTD* ioctls shall return the ENOTTY error code or the EINVAL error code.

Applications can make use of the *Input capabilities* and *Output capabilities* flags to determine whether the video standard ioctls can be used with the given input or output.

Example: Information about the current video standard

```
v4l2_std_id std_id;
struct v4l2_standard standard;

if (-1 == ioctl(fd, VIDIOC_G_STD, &std_id)) {
    /* Note when VIDIOC_ENUMSTD always returns ENOTTY this
       is no video device or it falls under the USB exception,
       and VIDIOC_G_STD returning ENOTTY is no error. */

    perror("VIDIOC_G_STD");
    exit(EXIT_FAILURE);
}

memset(&standard, 0, sizeof(standard));
standard.index = 0;

while (0 == ioctl(fd, VIDIOC_ENUMSTD, &standard)) {
    if (standard.id & std_id) {
           printf("Current video standard: %s\\n", standard.name);
           exit(EXIT_SUCCESS);
    }

    standard.index++;
}

/* EINVAL indicates the end of the enumeration, which cannot be
   empty unless this device falls under the USB exception. */

if (errno == EINVAL || standard.index == 0) {
    perror("VIDIOC_ENUMSTD");
    exit(EXIT_FAILURE);
}
```

Example: Listing the video standards supported by the current input

```
struct v4l2_input input;
struct v4l2_standard standard;

memset(&input, 0, sizeof(input));

if (-1 == ioctl(fd, VIDIOC_G_INPUT, &input.index)) {
    perror("VIDIOC_G_INPUT");
    exit(EXIT_FAILURE);
}

if (-1 == ioctl(fd, VIDIOC_ENUMINPUT, &input)) {
    perror("VIDIOC_ENUM_INPUT");
    exit(EXIT_FAILURE);
}

printf("Current input %s supports:\\n", input.name);

memset(&standard, 0, sizeof(standard));
standard.index = 0;

while (0 == ioctl(fd, VIDIOC_ENUMSTD, &standard)) {
    if (standard.id & input.std)
        printf("%s\\n", standard.name);

    standard.index++;
}

/* EINVAL indicates the end of the enumeration, which cannot be
   empty unless this device falls under the USB exception. */

if (errno != EINVAL || standard.index == 0) {
    perror("VIDIOC_ENUMSTD");
    exit(EXIT_FAILURE);
}
```

Example: Selecting a new video standard

```
struct v4l2_input input;
v4l2_std_id std_id;

memset(&input, 0, sizeof(input));

if (-1 == ioctl(fd, VIDIOC_G_INPUT, &input.index)) {
    perror("VIDIOC_G_INPUT");
    exit(EXIT_FAILURE);
}
```

(continues on next page)

(continued from previous page)

```
if (-1 == ioctl(fd, VIDIOC_ENUMINPUT, &input)) {
    perror("VIDIOC_ENUM_INPUT");
    exit(EXIT_FAILURE);
}

if (0 == (input.std & V4L2_STD_PAL_BG)) {
    fprintf(stderr, "Oops. B/G PAL is not supported.\\n");
    exit(EXIT_FAILURE);
}

/* Note this is also supposed to work when only B
   or G/PAL is supported. */

std_id = V4L2_STD_PAL_BG;

if (-1 == ioctl(fd, VIDIOC_S_STD, &std_id)) {
    perror("VIDIOC_S_STD");
    exit(EXIT_FAILURE);
}
```

Digital Video (DV) Timings

The video standards discussed so far have been dealing with Analog TV and the corresponding video timings. Today there are many more different hardware interfaces such as High Definition TV interfaces (HDMI), VGA, DVI connectors etc., that carry video signals and there is a need to extend the API to select the video timings for these interfaces. Since it is not possible to extend the *v4l2_std_id* due to the limited bits available, a new set of ioctls was added to set/get video timings at the input and output.

These ioctls deal with the detailed digital video timings that define each video format. This includes parameters such as the active video width and height, signal polarities, frontporches, backporches, sync widths etc. The `linux/v4l2-dv-timings.h` header can be used to get the timings of the formats in the *CEA-861-E* and *VESA DMT* standards.

To enumerate and query the attributes of the DV timings supported by a device applications use the *ioctl VIDIOC_ENUM_DV_TIMINGS, VIDIOC_SUBDEV_ENUM_DV_TIMINGS* and *ioctl VIDIOC_DV_TIMINGS_CAP, VIDIOC_SUBDEV_DV_TIMINGS_CAP* ioctls. To set DV timings for the device applications use the *VIDIOC_S_DV_TIMINGS* ioctl and to get current DV timings they use the *VIDIOC_G_DV_TIMINGS* ioctl. To detect the DV timings as seen by the video receiver applications use the *ioctl VIDIOC_QUERY_DV_TIMINGS* ioctl.

Applications can make use of the *Input capabilities* and *Output capabilities* flags to determine whether the digital video ioctls can be used with the given input or output.

User Controls

Devices typically have a number of user-settable controls such as brightness, saturation and so on, which would be presented to the user on a graphical user interface. But, different devices will have different controls available, and furthermore, the range of possible values, and the default value will vary from device to device. The control ioctls provide the information and a mechanism to create a nice user interface for these controls that will work correctly with any device.

All controls are accessed using an ID value. V4L2 defines several IDs for specific purposes. Drivers can also implement their own custom controls using `V4L2_CID_PRIVATE_BASE`[1] and higher values. The pre-defined control IDs have the prefix `V4L2_CID_`, and are listed in *Control IDs*. The ID is used when querying the attributes of a control, and when getting or setting the current value.

Generally applications should present controls to the user without assumptions about their purpose. Each control comes with a name string the user is supposed to understand. When the purpose is non-intuitive the driver writer should provide a user manual, a user interface plug-in or a driver specific panel application. Predefined IDs were introduced to change a few controls programmatically, for example to mute a device during a channel switch.

Drivers may enumerate different controls after switching the current video input or output, tuner or modulator, or audio input or output. Different in the sense of other bounds, another default and current value, step size or other menu items. A control with a certain *custom* ID can also change name and type.

If a control is not applicable to the current configuration of the device (for example, it doesn't apply to the current video input) drivers set the `V4L2_CTRL_FLAG_INACTIVE` flag.

Control values are stored globally, they do not change when switching except to stay within the reported bounds. They also do not change e. g. when the device is opened or closed, when the tuner radio frequency is changed or generally never without application request.

V4L2 specifies an event mechanism to notify applications when controls change value (see *ioctl VIDIOC_SUBSCRIBE_EVENT, VIDIOC_UNSUBSCRIBE_EVENT*, event `V4L2_EVENT_CTRL`), panel applications might want to make use of that in order to always reflect the correct control value.

All controls use machine endianness.

[1] The use of `V4L2_CID_PRIVATE_BASE` is problematic because different drivers may use the same `V4L2_CID_PRIVATE_BASE` ID for different controls. This makes it hard to programmatically set such controls since the meaning of the control with that ID is driver dependent. In order to resolve this drivers use unique IDs and the `V4L2_CID_PRIVATE_BASE` IDs are mapped to those unique IDs by the kernel. Consider these `V4L2_CID_PRIVATE_BASE` IDs as aliases to the real IDs.

Many applications today still use the `V4L2_CID_PRIVATE_BASE` IDs instead of using *ioctls VIDIOC_QUERYCTRL, VIDIOC_QUERY_EXT_CTRL and VIDIOC_QUERYMENU* with the `V4L2_CTRL_FLAG_NEXT_CTRL` flag to enumerate all IDs, so support for `V4L2_CID_PRIVATE_BASE` is still around.

Control IDs

V4L2_CID_BASE
First predefined ID, equal to V4L2_CID_BRIGHTNESS.

V4L2_CID_USER_BASE
Synonym of V4L2_CID_BASE.

V4L2_CID_BRIGHTNESS (integer)
Picture brightness, or more precisely, the black level.

V4L2_CID_CONTRAST (integer)
Picture contrast or luma gain.

V4L2_CID_SATURATION (integer)
Picture color saturation or chroma gain.

V4L2_CID_HUE (integer)
Hue or color balance.

V4L2_CID_AUDIO_VOLUME (integer)
Overall audio volume. Note some drivers also provide an OSS or ALSA mixer interface.

V4L2_CID_AUDIO_BALANCE (integer)
Audio stereo balance. Minimum corresponds to all the way left, maximum to right.

V4L2_CID_AUDIO_BASS (integer)
Audio bass adjustment.

V4L2_CID_AUDIO_TREBLE (integer)
Audio treble adjustment.

V4L2_CID_AUDIO_MUTE (boolean)
Mute audio, i. e. set the volume to zero, however without affecting V4L2_CID_AUDIO_VOLUME. Like ALSA drivers, V4L2 drivers must mute at load time to avoid excessive noise. Actually the entire device should be reset to a low power consumption state.

V4L2_CID_AUDIO_LOUDNESS (boolean)
Loudness mode (bass boost).

V4L2_CID_BLACK_LEVEL (integer)
Another name for brightness (not a synonym of V4L2_CID_BRIGHTNESS). This control is deprecated and should not be used in new drivers and applications.

V4L2_CID_AUTO_WHITE_BALANCE (boolean)
Automatic white balance (cameras).

V4L2_CID_DO_WHITE_BALANCE (button)
This is an action control. When set (the value is ignored), the device will do a white balance and then hold the current setting. Contrast this with the boolean V4L2_CID_AUTO_WHITE_BALANCE, which, when activated, keeps adjusting the white balance.

V4L2_CID_RED_BALANCE (integer)
Red chroma balance.

V4L2_CID_BLUE_BALANCE (integer)
Blue chroma balance.

V4L2_CID_GAMMA (integer)
Gamma adjust.

V4L2_CID_WHITENESS (integer)
Whiteness for grey-scale devices. This is a synonym for V4L2_CID_GAMMA. This control is deprecated and should not be used in new drivers and applications.

V4L2_CID_EXPOSURE (integer)
Exposure (cameras). [Unit?]

V4L2_CID_AUTOGAIN (boolean)
Automatic gain/exposure control.

V4L2_CID_GAIN (integer)
Gain control.

Primarily used to control gain on e.g. TV tuners but also on webcams. Most devices control only digital gain with this control but on some this could include analogue gain as well. Devices that recognise the difference between digital and analogue gain use controls V4L2_CID_DIGITAL_GAIN and V4L2_CID_ANALOGUE_GAIN.

V4L2_CID_HFLIP (boolean)
Mirror the picture horizontally.

V4L2_CID_VFLIP (boolean)
Mirror the picture vertically.

V4L2_CID_POWER_LINE_FREQUENCY (enum)
Enables a power line frequency filter to avoid flicker. Possible values for enum v4l2_power_line_frequency are: V4L2_CID_POWER_LINE_FREQUENCY_DISABLED (0), V4L2_CID_POWER_LINE_FREQUENCY_50HZ (1), V4L2_CID_POWER_LINE_FREQUENCY_60HZ (2) and V4L2_CID_POWER_LINE_FREQUENCY_AUTO (3).

V4L2_CID_HUE_AUTO (boolean)
Enables automatic hue control by the device. The effect of setting V4L2_CID_HUE while automatic hue control is enabled is undefined, drivers should ignore such request.

V4L2_CID_WHITE_BALANCE_TEMPERATURE (integer)
This control specifies the white balance settings as a color temperature in Kelvin. A driver should have a minimum of 2800 (incandescent) to 6500 (daylight). For more information about color temperature see Wikipedia.

V4L2_CID_SHARPNESS (integer)
Adjusts the sharpness filters in a camera. The minimum value disables the filters, higher values give a sharper picture.

V4L2_CID_BACKLIGHT_COMPENSATION (integer)
Adjusts the backlight compensation in a camera. The minimum value disables backlight compensation.

V4L2_CID_CHROMA_AGC (boolean)
Chroma automatic gain control.

V4L2_CID_CHROMA_GAIN (integer)
Adjusts the Chroma gain control (for use when chroma AGC is disabled).

V4L2_CID_COLOR_KILLER (boolean)
Enable the color killer (i. e. force a black & white image in case of a weak video signal).

V4L2_CID_COLORFX (enum)
Selects a color effect. The following values are defined:

V4L2_COLORFX_NONE	Color effect is disabled.
V4L2_COLORFX_ANTIQUE	An aging (old photo) effect.
V4L2_COLORFX_ART_FREEZE	Frost color effect.
V4L2_COLORFX_AQUA	Water color, cool tone.
V4L2_COLORFX_BW	Black and white.
V4L2_COLORFX_EMBOSS	Emboss, the highlights and shadows replace light/dark boundaries and low contrast areas are set to a gray background.
V4L2_COLORFX_GRASS_GREEN	Grass green.
V4L2_COLORFX_NEGATIVE	Negative.
V4L2_COLORFX_SEPIA	Sepia tone.
V4L2_COLORFX_SKETCH	Sketch.
V4L2_COLORFX_SKIN_WHITEN	Skin whiten.
V4L2_COLORFX_SKY_BLUE	Sky blue.
V4L2_COLORFX_SOLARIZATION	Solarization, the image is partially reversed in tone, only color values above or below a certain threshold are inverted.
V4L2_COLORFX_SILHOUETTE	Silhouette (outline).
V4L2_COLORFX_VIVID	Vivid colors.
V4L2_COLORFX_SET_CBCR	The Cb and Cr chroma components are replaced by fixed coefficients determined by V4L2_CID_COLORFX_CBCR control.

V4L2_CID_COLORFX_CBCR (integer)
Determines the Cb and Cr coefficients for V4L2_COLORFX_SET_CBCR color effect. Bits [7:0] of the supplied 32 bit value are interpreted as Cr component, bits [15:8] as Cb component and bits [31:16] must be zero.

V4L2_CID_AUTOBRIGHTNESS (boolean)
Enable Automatic Brightness.

V4L2_CID_ROTATE (integer)
Rotates the image by specified angle. Common angles are 90, 270 and 180. Rotating the image to 90 and 270 will reverse the height and width of the display window. It is necessary to set the new height and width of the picture using the *VIDIOC_S_FMT* ioctl according to the rotation angle selected.

V4L2_CID_BG_COLOR (integer)
Sets the background color on the current output device. Background color needs to be specified in the RGB24 format. The supplied 32 bit value is interpreted as bits 0-7 Red color information, bits 8-15 Green color information, bits 16-23 Blue color information and bits 24-31 must be zero.

V4L2_CID_ILLUMINATORS_1 V4L2_CID_ILLUMINATORS_2 (boolean)

Switch on or off the illuminator 1 or 2 of the device (usually a microscope).

V4L2_CID_MIN_BUFFERS_FOR_CAPTURE (integer)
This is a read-only control that can be read by the application and used as a hint to determine the number of CAPTURE buffers to pass to REQBUFS. The value is the minimum number of CAPTURE buffers that is necessary for hardware to work.

V4L2_CID_MIN_BUFFERS_FOR_OUTPUT (integer)
This is a read-only control that can be read by the application and used as a hint to determine the number of OUTPUT buffers to pass to REQBUFS. The value is the minimum number of OUTPUT buffers that is necessary for hardware to work.

V4L2_CID_ALPHA_COMPONENT (integer)
Sets the alpha color component. When a capture device (or capture queue of a mem-to-mem device) produces a frame format that includes an alpha component (e.g. *packed RGB image formats*) and the alpha value is not defined by the device or the mem-to-mem input data this control lets you select the alpha component value of all pixels. When an output device (or output queue of a mem-to-mem device) consumes a frame format that doesn't include an alpha component and the device supports alpha channel processing this control lets you set the alpha component value of all pixels for further processing in the device.

V4L2_CID_LASTP1
End of the predefined control IDs (currently `V4L2_CID_ALPHA_COMPONENT` + 1).

V4L2_CID_PRIVATE_BASE
ID of the first custom (driver specific) control. Applications depending on particular custom controls should check the driver name and version, see *Querying Capabilities*.

Applications can enumerate the available controls with the *ioctls VIDIOC_QUERYCTRL, VIDIOC_QUERY_EXT_CTRL and VIDIOC_QUERYMENU* and *VIDIOC_QUERYMENU* ioctls, get and set a control value with the *VIDIOC_G_CTRL* and *VIDIOC_S_CTRL* ioctls. Drivers must implement `VIDIOC_QUERYCTRL`, `VIDIOC_G_CTRL` and `VIDIOC_S_CTRL` when the device has one or more controls, `VIDIOC_QUERYMENU` when it has one or more menu type controls.

Example: Enumerating all controls

```
struct v4l2_queryctrl queryctrl;
struct v4l2_querymenu querymenu;

static void enumerate_menu(__u32 id)
{
    printf("  Menu items:\\n");

    memset(&querymenu, 0, sizeof(querymenu));
    querymenu.id = id;
```

(continues on next page)

(continued from previous page)

```
    for (querymenu.index = queryctrl.minimum;
         querymenu.index <= queryctrl.maximum;
         querymenu.index++) {
        if (0 == ioctl(fd, VIDIOC_QUERYMENU, &querymenu)) {
            printf("  %s\\n", querymenu.name);
        }
    }
}

memset(&queryctrl, 0, sizeof(queryctrl));

queryctrl.id = V4L2_CTRL_FLAG_NEXT_CTRL;
while (0 == ioctl(fd, VIDIOC_QUERYCTRL, &queryctrl)) {
    if (!(queryctrl.flags & V4L2_CTRL_FLAG_DISABLED)) {
        printf("Control %s\\n", queryctrl.name);

        if (queryctrl.type == V4L2_CTRL_TYPE_MENU)
            enumerate_menu(queryctrl.id);
    }

    queryctrl.id |= V4L2_CTRL_FLAG_NEXT_CTRL;
}
if (errno != EINVAL) {
    perror("VIDIOC_QUERYCTRL");
    exit(EXIT_FAILURE);
}
```

Example: Enumerating all controls including compound controls

```
struct v4l2_query_ext_ctrl query_ext_ctrl;

memset(&query_ext_ctrl, 0, sizeof(query_ext_ctrl));

query_ext_ctrl.id = V4L2_CTRL_FLAG_NEXT_CTRL | V4L2_CTRL_FLAG_NEXT_
↪COMPOUND;
while (0 == ioctl(fd, VIDIOC_QUERY_EXT_CTRL, &query_ext_ctrl)) {
    if (!(query_ext_ctrl.flags & V4L2_CTRL_FLAG_DISABLED)) {
        printf("Control %s\\n", query_ext_ctrl.name);

        if (query_ext_ctrl.type == V4L2_CTRL_TYPE_MENU)
            enumerate_menu(query_ext_ctrl.id);
    }

    query_ext_ctrl.id |= V4L2_CTRL_FLAG_NEXT_CTRL | V4L2_CTRL_FLAG_
↪NEXT_COMPOUND;
}
if (errno != EINVAL) {
```

(continues on next page)

(continued from previous page)

```
    perror("VIDIOC_QUERY_EXT_CTRL");
    exit(EXIT_FAILURE);
}
```

Example: Enumerating all user controls (old style)

```
memset(&queryctrl, 0, sizeof(queryctrl));

for (queryctrl.id = V4L2_CID_BASE;
     queryctrl.id < V4L2_CID_LASTP1;
     queryctrl.id++) {
    if (0 == ioctl(fd, VIDIOC_QUERYCTRL, &queryctrl)) {
        if (queryctrl.flags & V4L2_CTRL_FLAG_DISABLED)
            continue;

        printf("Control %s\\n", queryctrl.name);

        if (queryctrl.type == V4L2_CTRL_TYPE_MENU)
            enumerate_menu(queryctrl.id);
    } else {
        if (errno == EINVAL)
            continue;

        perror("VIDIOC_QUERYCTRL");
        exit(EXIT_FAILURE);
    }
}

for (queryctrl.id = V4L2_CID_PRIVATE_BASE;;
     queryctrl.id++) {
    if (0 == ioctl(fd, VIDIOC_QUERYCTRL, &queryctrl)) {
        if (queryctrl.flags & V4L2_CTRL_FLAG_DISABLED)
            continue;

        printf("Control %s\\n", queryctrl.name);

        if (queryctrl.type == V4L2_CTRL_TYPE_MENU)
            enumerate_menu(queryctrl.id);
    } else {
        if (errno == EINVAL)
            break;

        perror("VIDIOC_QUERYCTRL");
        exit(EXIT_FAILURE);
    }
}
```

Example: Changing controls

```
struct v4l2_queryctrl queryctrl;
struct v4l2_control control;

memset(&queryctrl, 0, sizeof(queryctrl));
queryctrl.id = V4L2_CID_BRIGHTNESS;

if (-1 == ioctl(fd, VIDIOC_QUERYCTRL, &queryctrl)) {
    if (errno != EINVAL) {
        perror("VIDIOC_QUERYCTRL");
        exit(EXIT_FAILURE);
    } else {
        printf("V4L2_CID_BRIGHTNESS is not supportedn");
    }
} else if (queryctrl.flags & V4L2_CTRL_FLAG_DISABLED) {
    printf("V4L2_CID_BRIGHTNESS is not supportedn");
} else {
    memset(&control, 0, sizeof (control));
    control.id = V4L2_CID_BRIGHTNESS;
    control.value = queryctrl.default_value;

    if (-1 == ioctl(fd, VIDIOC_S_CTRL, &control)) {
        perror("VIDIOC_S_CTRL");
        exit(EXIT_FAILURE);
    }
}

memset(&control, 0, sizeof(control));
control.id = V4L2_CID_CONTRAST;

if (0 == ioctl(fd, VIDIOC_G_CTRL, &control)) {
    control.value += 1;

    /* The driver may clamp the value or return ERANGE, ignored␣
↪here */

    if (-1 == ioctl(fd, VIDIOC_S_CTRL, &control)
        && errno != ERANGE) {
        perror("VIDIOC_S_CTRL");
        exit(EXIT_FAILURE);
    }
/* Ignore if V4L2_CID_CONTRAST is unsupported */
} else if (errno != EINVAL) {
    perror("VIDIOC_G_CTRL");
    exit(EXIT_FAILURE);
}

control.id = V4L2_CID_AUDIO_MUTE;
control.value = 1; /* silence */
```

(continues on next page)

(continued from previous page)

```
/* Errors ignored */
ioctl(fd, VIDIOC_S_CTRL, &control);
```

Extended Controls API

Introduction

The control mechanism as originally designed was meant to be used for user settings (brightness, saturation, etc). However, it turned out to be a very useful model for implementing more complicated driver APIs where each driver implements only a subset of a larger API.

The MPEG encoding API was the driving force behind designing and implementing this extended control mechanism: the MPEG standard is quite large and the currently supported hardware MPEG encoders each only implement a subset of this standard. Further more, many parameters relating to how the video is encoded into an MPEG stream are specific to the MPEG encoding chip since the MPEG standard only defines the format of the resulting MPEG stream, not how the video is actually encoded into that format.

Unfortunately, the original control API lacked some features needed for these new uses and so it was extended into the (not terribly originally named) extended control API.

Even though the MPEG encoding API was the first effort to use the Extended Control API, nowadays there are also other classes of Extended Controls, such as Camera Controls and FM Transmitter Controls. The Extended Controls API as well as all Extended Controls classes are described in the following text.

The Extended Control API

Three new ioctls are available: *VIDIOC_G_EXT_CTRLS*, *VIDIOC_S_EXT_CTRLS* and *VIDIOC_TRY_EXT_CTRLS*. These ioctls act on arrays of controls (as opposed to the *VIDIOC_G_CTRL* and *VIDIOC_S_CTRL* ioctls that act on a single control). This is needed since it is often required to atomically change several controls at once.

Each of the new ioctls expects a pointer to a struct `v4l2_ext_controls`. This structure contains a pointer to the control array, a count of the number of controls in that array and a control class. Control classes are used to group similar controls into a single class. For example, control class `V4L2_CTRL_CLASS_USER` contains all user controls (i. e. all controls that can also be set using the old *VIDIOC_S_CTRL* ioctl). Control class `V4L2_CTRL_CLASS_MPEG` contains all controls relating to MPEG encoding, etc.

All controls in the control array must belong to the specified control class. An error is returned if this is not the case.

It is also possible to use an empty control array (`count` == 0) to check whether the specified control class is supported.

The control array is a struct `v4l2_ext_control` array. The struct `v4l2_ext_control` is very similar to struct `v4l2_control`, except for the fact that it also allows for 64-bit values and pointers to be passed.

Since the struct `v4l2_ext_control` supports pointers it is now also possible to have controls with compound types such as N-dimensional arrays and/or structures. You need to specify the `V4L2_CTRL_FLAG_NEXT_COMPOUND` when enumerating controls to actually be able to see such compound controls. In other words, these controls with compound types should only be used programmatically.

Since such compound controls need to expose more information about themselves than is possible with *VIDIOC_QUERYCTRL* the *VIDIOC_QUERY_EXT_CTRL* ioctl was added. In particular, this ioctl gives the dimensions of the N-dimensional array if this control consists of more than one element.

Note:

1. It is important to realize that due to the flexibility of controls it is necessary to check whether the control you want to set actually is supported in the driver and what the valid range of values is. So use *ioctls VIDIOC_QUERYCTRL, VIDIOC_QUERY_EXT_CTRL and VIDIOC_QUERYMENU* to check this.
2. It is possible that some of the menu indices in a control of type `V4L2_CTRL_TYPE_MENU` may not be supported (`VIDIOC_QUERYMENU` will return an error). A good example is the list of supported MPEG audio bitrates. Some drivers only support one or two bitrates, others support a wider range.

All controls use machine endianness.

Enumerating Extended Controls

The recommended way to enumerate over the extended controls is by using *ioctls VIDIOC_QUERYCTRL, VIDIOC_QUERY_EXT_CTRL and VIDIOC_QUERYMENU* in combination with the `V4L2_CTRL_FLAG_NEXT_CTRL` flag:

```
struct v4l2_queryctrl qctrl;

qctrl.id = V4L2_CTRL_FLAG_NEXT_CTRL;
while (0 == ioctl (fd, VIDIOC_QUERYCTRL, &qctrl)) {
    /* ... */
    qctrl.id |= V4L2_CTRL_FLAG_NEXT_CTRL;
}
```

The initial control ID is set to 0 ORed with the `V4L2_CTRL_FLAG_NEXT_CTRL` flag. The `VIDIOC_QUERYCTRL` ioctl will return the first control with a higher ID than the specified one. When no such controls are found an error is returned.

If you want to get all controls within a specific control class, then you can set the initial `qctrl.id` value to the control class and add an extra check to break out of the loop when a control of another control class is found:

```
qctrl.id = V4L2_CTRL_CLASS_MPEG | V4L2_CTRL_FLAG_NEXT_CTRL;
while (0 == ioctl(fd, VIDIOC_QUERYCTRL, &qctrl)) {
    if (V4L2_CTRL_ID2CLASS(qctrl.id) != V4L2_CTRL_CLASS_MPEG)
        break;
    /* ... */
    qctrl.id |= V4L2_CTRL_FLAG_NEXT_CTRL;
}
```

The 32-bit `qctrl.id` value is subdivided into three bit ranges: the top 4 bits are reserved for flags (e. g. `V4L2_CTRL_FLAG_NEXT_CTRL`) and are not actually part of the ID. The remaining 28 bits form the control ID, of which the most significant 12 bits define the control class and the least significant 16 bits identify the control within the control class. It is guaranteed that these last 16 bits are always non-zero for controls. The range of 0x1000 and up are reserved for driver-specific controls. The macro `V4L2_CTRL_ID2CLASS(id)` returns the control class ID based on a control ID.

If the driver does not support extended controls, then `VIDIOC_QUERYCTRL` will fail when used in combination with `V4L2_CTRL_FLAG_NEXT_CTRL`. In that case the old method of enumerating control should be used (see *Example: Enumerating all controls*). But if it is supported, then it is guaranteed to enumerate over all controls, including driver-private controls.

Creating Control Panels

It is possible to create control panels for a graphical user interface where the user can select the various controls. Basically you will have to iterate over all controls using the method described above. Each control class starts with a control of type `V4L2_CTRL_TYPE_CTRL_CLASS`. `VIDIOC_QUERYCTRL` will return the name of this control class which can be used as the title of a tab page within a control panel.

The flags field of struct *v4l2_queryctrl* also contains hints on the behavior of the control. See the *ioctls VIDIOC_QUERYCTRL, VIDIOC_QUERY_EXT_CTRL and VIDIOC_QUERYMENU* documentation for more details.

Camera Control Reference

The Camera class includes controls for mechanical (or equivalent digital) features of a device such as controllable lenses or sensors.

Camera Control IDs

`V4L2_CID_CAMERA_CLASS (class)`
The Camera class descriptor. Calling *ioctls VIDIOC_QUERYCTRL, VIDIOC_QUERY_EXT_CTRL and VIDIOC_QUERYMENU* for this control will return a description of this control class.

`V4L2_CID_EXPOSURE_AUTO`
(enum)

enum v4l2_exposure_auto_type -

Enables automatic adjustments of the exposure time and/or iris aperture. The effect of manual changes of the exposure time or iris aperture while these features are enabled is undefined, drivers should ignore such requests. Possible values are:

V4L2_EXPOSURE_AUTO	Automatic exposure time, automatic iris aperture.
V4L2_EXPOSURE_MANUAL	Manual exposure time, manual iris.
V4L2_EXPOSURE_SHUTTER_PRIORIT	Manual exposure time, auto iris.
V4L2_EXPOSURE_APERTURE_PRIORI	Auto exposure time, manual iris.

V4L2_CID_EXPOSURE_ABSOLUTE (integer)

Determines the exposure time of the camera sensor. The exposure time is limited by the frame interval. Drivers should interpret the values as 100 µs units, where the value 1 stands for 1/10000th of a second, 10000 for 1 second and 100000 for 10 seconds.

V4L2_CID_EXPOSURE_AUTO_PRIORITY (boolean)

When V4L2_CID_EXPOSURE_AUTO is set to AUTO or APERTURE_PRIORITY, this control determines if the device may dynamically vary the frame rate. By default this feature is disabled (0) and the frame rate must remain constant.

V4L2_CID_AUTO_EXPOSURE_BIAS (integer menu)

Determines the automatic exposure compensation, it is effective only when V4L2_CID_EXPOSURE_AUTO control is set to AUTO, SHUTTER_PRIORITY or APERTURE_PRIORITY. It is expressed in terms of EV, drivers should interpret the values as 0.001 EV units, where the value 1000 stands for +1 EV.

Increasing the exposure compensation value is equivalent to decreasing the exposure value (EV) and will increase the amount of light at the image sensor. The camera performs the exposure compensation by adjusting absolute exposure time and/or aperture.

V4L2_CID_EXPOSURE_METERING

(enum)

enum v4l2_exposure_metering -

Determines how the camera measures the amount of light available for the frame exposure. Possible values are:

V4L2_EXPOSURE_METERING_AVERAGE	Use the light information coming from the entire frame and average giving no weighting to any particular portion of the metered area.
V4L2_EXPOSURE_METERING_CENTER_WEIGHTED	Average the light information coming from the entire frame giving priority to the center of the metered area.
V4L2_EXPOSURE_METERING_SPOT	Measure only very small area at the center of the frame.
V4L2_EXPOSURE_METERING_MATRIX	A multi-zone metering. The light intensity is measured in several points of the frame and the results are combined. The algorithm of the zones selection and their significance in calculating the final value is device dependent.

V4L2_CID_PAN_RELATIVE (integer)
: This control turns the camera horizontally by the specified amount. The unit is undefined. A positive value moves the camera to the right (clockwise when viewed from above), a negative value to the left. A value of zero does not cause motion. This is a write-only control.

V4L2_CID_TILT_RELATIVE (integer)
: This control turns the camera vertically by the specified amount. The unit is undefined. A positive value moves the camera up, a negative value down. A value of zero does not cause motion. This is a write-only control.

V4L2_CID_PAN_RESET (button)
: When this control is set, the camera moves horizontally to the default position.

V4L2_CID_TILT_RESET (button)
: When this control is set, the camera moves vertically to the default position.

V4L2_CID_PAN_ABSOLUTE (integer)
: This control turns the camera horizontally to the specified position. Positive values move the camera to the right (clockwise when viewed from above), negative values to the left. Drivers should interpret the values as arc seconds, with valid values between -180 * 3600 and +180 * 3600 inclusive.

V4L2_CID_TILT_ABSOLUTE (integer)
: This control turns the camera vertically to the specified position. Positive values move the camera up, negative values down. Drivers should interpret the values as arc seconds, with valid values between -180 * 3600 and +180 * 3600 inclusive.

V4L2_CID_FOCUS_ABSOLUTE (integer)
: This control sets the focal point of the camera to the specified position. The unit is undefined. Positive values set the focus closer to the camera, negative values towards infinity.

V4L2_CID_FOCUS_RELATIVE (integer)
: This control moves the focal point of the camera by the specified amount. The unit is undefined. Positive values move the focus closer to the camera, negative values towards infinity. This is a write-only control.

V4L2_CID_FOCUS_AUTO (boolean)
Enables continuous automatic focus adjustments. The effect of manual focus adjustments while this feature is enabled is undefined, drivers should ignore such requests.

V4L2_CID_AUTO_FOCUS_START (button)
Starts single auto focus process. The effect of setting this control when V4L2_CID_FOCUS_AUTO is set to TRUE (1) is undefined, drivers should ignore such requests.

V4L2_CID_AUTO_FOCUS_STOP (button)
Aborts automatic focusing started with V4L2_CID_AUTO_FOCUS_START control. It is effective only when the continuous autofocus is disabled, that is when V4L2_CID_FOCUS_AUTO control is set to FALSE (0).

V4L2_CID_AUTO_FOCUS_STATUS (bitmask)
The automatic focus status. This is a read-only control.

Setting V4L2_LOCK_FOCUS lock bit of the V4L2_CID_3A_LOCK control may stop updates of the V4L2_CID_AUTO_FOCUS_STATUS control value.

V4L2_AUTO_FOCUS_STATUS_IDLE	Automatic focus is not active.
V4L2_AUTO_FOCUS_STATUS_BUSY	Automatic focusing is in progress.
V4L2_AUTO_FOCUS_STATUS_REACHED	Focus has been reached.
V4L2_AUTO_FOCUS_STATUS_FAILED	Automatic focus has failed, the driver will not transition from this state until another action is performed by an application.

V4L2_CID_AUTO_FOCUS_RANGE
(enum)

enum v4l2_auto_focus_range -
Determines auto focus distance range for which lens may be adjusted.

V4L2_AUTO_FOCUS_RANGE_AUTO	The camera automatically selects the focus range.
V4L2_AUTO_FOCUS_RANGE_NORMAL	Normal distance range, limited for best automatic focus performance.
V4L2_AUTO_FOCUS_RANGE_MACRO	Macro (close-up) auto focus. The camera will use its minimum possible distance for auto focus.
V4L2_AUTO_FOCUS_RANGE_INFINITY	The lens is set to focus on an object at infinite distance.

V4L2_CID_ZOOM_ABSOLUTE (integer)
Specify the objective lens focal length as an absolute value. The zoom unit is driver-specific and its value should be a positive integer.

V4L2_CID_ZOOM_RELATIVE (integer)
Specify the objective lens focal length relatively to the current value. Positive values move the zoom lens group towards the telephoto direction, negative values towards the wide-angle direction. The zoom unit is driver-specific. This is a write-only control.

V4L2_CID_ZOOM_CONTINUOUS (integer)
Move the objective lens group at the specified speed until it reaches physical

device limits or until an explicit request to stop the movement. A positive value moves the zoom lens group towards the telephoto direction. A value of zero stops the zoom lens group movement. A negative value moves the zoom lens group towards the wide-angle direction. The zoom speed unit is driver-specific.

V4L2_CID_IRIS_ABSOLUTE (integer)
This control sets the camera's aperture to the specified value. The unit is undefined. Larger values open the iris wider, smaller values close it.

V4L2_CID_IRIS_RELATIVE (integer)
This control modifies the camera's aperture by the specified amount. The unit is undefined. Positive values open the iris one step further, negative values close it one step further. This is a write-only control.

V4L2_CID_PRIVACY (boolean)
Prevent video from being acquired by the camera. When this control is set to TRUE (1), no image can be captured by the camera. Common means to enforce privacy are mechanical obturation of the sensor and firmware image processing, but the device is not restricted to these methods. Devices that implement the privacy control must support read access and may support write access.

V4L2_CID_BAND_STOP_FILTER (integer)
Switch the band-stop filter of a camera sensor on or off, or specify its strength. Such band-stop filters can be used, for example, to filter out the fluorescent light component.

V4L2_CID_AUTO_N_PRESET_WHITE_BALANCE
(enum)

enum v4l2_auto_n_preset_white_balance -
Sets white balance to automatic, manual or a preset. The presets determine color temperature of the light as a hint to the camera for white balance adjustments resulting in most accurate color representation. The following white balance presets are listed in order of increasing color temperature.

V4L2_WHITE_BALANCE_MANUAL	Manual white balance.
V4L2_WHITE_BALANCE_AUTO	Automatic white balance adjustments.
V4L2_WHITE_BALANCE_INCANDESCENT	White balance setting for incandescent (tungsten) lighting. It generally cools down the colors and corresponds approximately to 2500…3500 K color temperature range.
V4L2_WHITE_BALANCE_FLUORESCENT	White balance preset for fluorescent lighting. It corresponds approximately to 4000…5000 K color temperature.
V4L2_WHITE_BALANCE_FLUORESCENT_H	With this setting the camera will compensate for fluorescent H lighting.
V4L2_WHITE_BALANCE_HORIZON	White balance setting for horizon daylight. It corresponds approximately to 5000 K color temperature.
V4L2_WHITE_BALANCE_DAYLIGHT	White balance preset for daylight (with clear sky). It corresponds approximately to 5000…6500 K color temperature.
V4L2_WHITE_BALANCE_FLASH	With this setting the camera will compensate for the flash light. It slightly warms up the colors and corresponds roughly to 5000…5500 K color temperature.
V4L2_WHITE_BALANCE_CLOUDY	White balance preset for moderately overcast sky. This option corresponds approximately to 6500…8000 K color temperature range.
V4L2_WHITE_BALANCE_SHADE	White balance preset for shade or heavily overcast sky. It corresponds approximately to 9000…10000 K color temperature.

V4L2_CID_WIDE_DYNAMIC_RANGE (boolean)
Enables or disables the camera's wide dynamic range feature. This feature allows to obtain clear images in situations where intensity of the illumination varies significantly throughout the scene, i.e. there are simultaneously very dark and very bright areas. It is most commonly realized in cameras by combining two subsequent frames with different exposure times.[1]

V4L2_CID_IMAGE_STABILIZATION (boolean)
Enables or disables image stabilization.

V4L2_CID_ISO_SENSITIVITY (integer menu)
Determines ISO equivalent of an image sensor indicating the sensor's sensitivity to light. The numbers are expressed in arithmetic scale, as per *ISO 12232:2006* standard, where doubling the sensor sensitivity is represented by doubling the numerical ISO value. Applications should interpret the values as standard ISO values multiplied by 1000, e.g. control value 800 stands for ISO 0.8. Drivers will usually support only a subset of standard ISO values. The effect of setting this control while the `V4L2_CID_ISO_SENSITIVITY_AUTO` control is set to a value other than `V4L2_CID_ISO_SENSITIVITY_MANUAL` is undefined, drivers should ignore such requests.

V4L2_CID_ISO_SENSITIVITY_AUTO
(enum)

[1] This control may be changed to a menu control in the future, if more options are required.

enum v4l2_iso_sensitivity_type -

Enables or disables automatic ISO sensitivity adjustments.

`V4L2_CID_ISO_SENSITIVITY_MANUAL`	Manual ISO sensitivity.
`V4L2_CID_ISO_SENSITIVITY_AUTO`	Automatic ISO sensitivity adjustments.

`V4L2_CID_SCENE_MODE`

(enum)

enum v4l2_scene_mode -

This control allows to select scene programs as the camera automatic modes optimized for common shooting scenes. Within these modes the camera determines best exposure, aperture, focusing, light metering, white balance and equivalent sensitivity. The controls of those parameters are influenced by the scene mode control. An exact behavior in each mode is subject to the camera specification.

When the scene mode feature is not used, this control should be set to `V4L2_SCENE_MODE_NONE` to make sure the other possibly related controls are accessible. The following scene programs are defined:

V4L2_SCENE_MODE_NONE	The scene mode feature is disabled.
V4L2_SCENE_MODE_BACKLIGHT	Backlight. Compensates for dark shadows when light is coming from behind a subject, also by automatically turning on the flash.
V4L2_SCENE_MODE_BEACH_SNOW	Beach and snow. This mode compensates for all-white or bright scenes, which tend to look gray and low contrast, when camera's automatic exposure is based on an average scene brightness. To compensate, this mode automatically slightly overexposes the frames. The white balance may also be adjusted to compensate for the fact that reflected snow looks bluish rather than white.
V4L2_SCENE_MODE_CANDLELIGHT	Candle light. The camera generally raises the ISO sensitivity and lowers the shutter speed. This mode compensates for relatively close subject in the scene. The flash is disabled in order to preserve the ambiance of the light.
V4L2_SCENE_MODE_DAWN_DUSK	Dawn and dusk. Preserves the colors seen in low natural light before dusk and after down. The camera may turn off the flash, and automatically focus at infinity. It will usually boost saturation and lower the shutter speed.
V4L2_SCENE_MODE_FALL_COLORS	Fall colors. Increases saturation and adjusts white balance for color enhancement. Pictures of autumn leaves get saturated reds and yellows.
V4L2_SCENE_MODE_FIREWORKS	Fireworks. Long exposure times are used to capture the expanding burst of light from a firework. The camera may invoke image stabilization.
V4L2_SCENE_MODE_LANDSCAPE	Landscape. The camera may choose a small aperture to provide deep depth of field and long exposure duration to help capture detail in dim light conditions. The focus is fixed at infinity. Suitable for distant and wide scenery.
V4L2_SCENE_MODE_NIGHT	Night, also known as Night Landscape. Designed for low light conditions, it preserves detail in the dark areas without blowing out bright objects. The camera generally sets itself to a medium-to-high ISO sensitivity, with a relatively long exposure time, and turns flash off. As such, there will be increased image noise and the possibility of blurred image.
V4L2_SCENE_MODE_PARTY_INDOOR	Party and indoor. Designed to capture indoor scenes that are lit by indoor background lighting as well as the flash. The camera usually increases ISO sensitivity, and adjusts exposure for the low light conditions.
V4L2_SCENE_MODE_PORTRAIT	Portrait. The camera adjusts the aperture so that the depth of field is reduced, which helps to isolate the subject against a smooth background. Most cameras recognize the presence of faces in the scene and focus on them. The color hue is adjusted to enhance skin tones. The intensity of the flash is often reduced.
V4L2_SCENE_MODE_SPORTS	Sports. Significantly increases ISO and uses a fast shutter speed to freeze motion of rapidly-moving subjects. Increased image noise may be seen in this mode.
V4L2_SCENE_MODE_SUNSET	Sunset. Preserves deep hues seen in sunsets and sunrises. It bumps up the saturation.
V4L2_SCENE_MODE_TEXT	Text. It applies extra contrast and sharpness, it is typically a black-and-white mode optimized for readability. Automatic focus may be switched to close-up mode and this setting may also involve some lens-distortion correction.

V4L2_CID_3A_LOCK (bitmask)

This control locks or unlocks the automatic focus, exposure and white balance. The automatic adjustments can be paused independently by setting the corresponding lock bit to 1. The camera then retains the settings until the lock bit is cleared. The following lock bits are defined:

When a given algorithm is not enabled, drivers should ignore requests to lock it and should return no error. An example might be an application setting bit V4L2_LOCK_WHITE_BALANCE when the V4L2_CID_AUTO_WHITE_BALANCE control is set to FALSE. The value of this control may be changed by exposure, white balance or focus controls.

V4L2_LOCK_EXPOSURE	Automatic exposure adjustments lock.
V4L2_LOCK_WHITE_BALANCE	Automatic white balance adjustments lock.
V4L2_LOCK_FOCUS	Automatic focus lock.

V4L2_CID_PAN_SPEED (integer)

This control turns the camera horizontally at the specific speed. The unit is undefined. A positive value moves the camera to the right (clockwise when viewed from above), a negative value to the left. A value of zero stops the motion if one is in progress and has no effect otherwise.

V4L2_CID_TILT_SPEED (integer)

This control turns the camera vertically at the specified speed. The unit is undefined. A positive value moves the camera up, a negative value down. A value of zero stops the motion if one is in progress and has no effect otherwise.

V4L2_CID_CAMERA_ORIENTATION (menu)

This read-only control describes the camera orientation by reporting its mounting position on the device where the camera is installed. The control value is constant and not modifiable by software. This control is particularly meaningful for devices which have a well defined orientation, such as phones, laptops and portable devices since the control is expressed as a position relative to the device' s intended usage orientation. For example, a camera installed on the user-facing side of a phone, a tablet or a laptop device is said to be have V4L2_CAMERA_ORIENTATION_FRONT orientation, while a camera installed on the opposite side of the front one is said to be have V4L2_CAMERA_ORIENTATION_BACK orientation. Camera sensors not directly attached to the device, or attached in a way that allows them to move freely, such as webcams and digital cameras, are said to have the V4L2_CAMERA_ORIENTATION_EXTERNAL orientation.

V4L2_CAMERA_ORIENTATION_	The camera is oriented towards the user facing side of the device.
V4L2_CAMERA_ORIENTATION_	The camera is oriented towards the back facing side of the device.
V4L2_CAMERA_ORIENTATION_	The camera is not directly attached to the device and is freely movable.

V4L2_CID_CAMERA_SENSOR_ROTATION (integer)

This read-only control describes the rotation correction in degrees in the counter-clockwise direction to be applied to the captured images once captured to memory to compensate for the camera sensor mounting rotation.

For a precise definition of the sensor mounting rotation refer to the extensive description of the 'rotation' properties in the device tree bindings file 'video-interfaces.txt' .

A few examples are below reported, using a shark swimming from left to right in front of the user as the example scene to capture.

```
  0               X-axis
0 +-------------------------------------->
  !
  !
  !
  !           |\____)\___
  !           ) _____  __`<
  !           |/     )/
  !
  !
  !
  V
Y-axis
```

Example one - Webcam

Assuming you can bring your laptop with you while swimming with sharks, the camera module of the laptop is installed on the user facing part of a laptop screen casing, and is typically used for video calls. The captured images are meant to be displayed in landscape mode (width > height) on the laptop screen.

The camera is typically mounted upside-down to compensate the lens optical inversion effect. In this case the value of the V4L2_CID_CAMERA_SENSOR_ROTATION control is 0, no rotation is required to display images correctly to the user.

If the camera sensor is not mounted upside-down it is required to compensate the lens optical inversion effect and the value of the V4L2_CID_CAMERA_SENSOR_ROTATION control is 180 degrees, as images will result rotated when captured to memory.

```
+---------------------------------------+
!                                       !
!                                       !
!                                       !
!              __/(_____/|              !
!            >.___  ____ (              !
!                 \(    \|              !
!                                       !
!                                       !
!                                       !
+---------------------------------------+
```

A software rotation correction of 180 degrees has to be applied to correctly display the image on the user screen.

```
+---------------------------------------+
!                                       !
!                                       !
!                                       !
!              |\____)\___              !
!              ) _____  __`<            !
!              |/     )/                !
!                                       !
!                                       !
!                                       !
+---------------------------------------+
```

Example two - Phone camera

It is more handy to go and swim with sharks with only your mobile phone with you and take pictures with the camera that is installed on the back side of the device, facing away from the user. The captured images are meant to be displayed in portrait mode (height > width) to match the device screen orientation and the device usage orientation used when taking the picture.

The camera sensor is typically mounted with its pixel array longer side aligned to the device longer side, upside-down mounted to compensate for the lens optical inversion effect.

The images once captured to memory will be rotated and the value of the V4L2_CID_CAMERA_SENSOR_ROTATION will report a 90 degree rotation.

```
+--------------------------------------+
|                    _ _               |
|                   \   /              |
|                    | |               |
|                    | |               |
|                    |  >              |
|                   <  |               |
|                    | |               |
|                      .               |
|                     V                |
+--------------------------------------+
```

A correction of 90 degrees in counter-clockwise direction has to be applied to correctly display the image in portrait mode on the device screen.

```
+---------------------+
|                     |
|                     |
|                     |
|                     |
|                     |
|                     |
|    |\____)\___      |
|    ) _____  __`<    |
|    |/     )/        |
```

(continues on next page)

(continued from previous page)

```
|                    |
|                    |
|                    |
|                    |
|                    |
+--------------------+
```

Flash Control Reference

The V4L2 flash controls are intended to provide generic access to flash controller devices. Flash controller devices are typically used in digital cameras.

The interface can support both LED and xenon flash devices. As of writing this, there is no xenon flash driver using this interface.

Supported use cases

Unsynchronised LED flash (software strobe)

Unsynchronised LED flash is controlled directly by the host as the sensor. The flash must be enabled by the host before the exposure of the image starts and disabled once it ends. The host is fully responsible for the timing of the flash.

Example of such device: Nokia N900.

Synchronised LED flash (hardware strobe)

The synchronised LED flash is pre-programmed by the host (power and timeout) but controlled by the sensor through a strobe signal from the sensor to the flash.

The sensor controls the flash duration and timing. This information typically must be made available to the sensor.

LED flash as torch

LED flash may be used as torch in conjunction with another use case involving camera or individually.

Flash Control IDs

V4L2_CID_FLASH_CLASS (class)
The FLASH class descriptor.

V4L2_CID_FLASH_LED_MODE (menu)
Defines the mode of the flash LED, the high-power white LED attached to the flash controller. Setting this control may not be possible in presence of some faults. See V4L2_CID_FLASH_FAULT.

V4L2_FLASH_LED_MODE_NON	Off.
V4L2_FLASH_LED_MODE_FLA	Flash mode.
V4L2_FLASH_LED_MODE_TOR	Torch mode. See V4L2_CID_FLASH_TORCH_INTENSITY.

V4L2_CID_FLASH_STROBE_SOURCE (menu)
Defines the source of the flash LED strobe.

V4L2_FLASH_STROBE_SOURCE_SOFTWARE	The flash strobe is triggered by using the V4L2_CID_FLASH_STROBE control.
V4L2_FLASH_STROBE_SOURCE_EXTERNAL	The flash strobe is triggered by an external source. Typically this is a sensor, which makes it possible to synchronise the flash strobe start to exposure start.

V4L2_CID_FLASH_STROBE (button)
Strobe flash. Valid when V4L2_CID_FLASH_LED_MODE is set to V4L2_FLASH_LED_MODE_FLASH and V4L2_CID_FLASH_STROBE_SOURCE is set to V4L2_FLASH_STROBE_SOURCE_SOFTWA Setting this control may not be possible in presence of some faults. See V4L2_CID_FLASH_FAULT.

V4L2_CID_FLASH_STROBE_STOP (button)
Stop flash strobe immediately.

V4L2_CID_FLASH_STROBE_STATUS (boolean)
Strobe status: whether the flash is strobing at the moment or not. This is a read-only control.

V4L2_CID_FLASH_TIMEOUT (integer)
Hardware timeout for flash. The flash strobe is stopped after this period of time has passed from the start of the strobe.

V4L2_CID_FLASH_INTENSITY (integer)
Intensity of the flash strobe when the flash LED is in flash mode (V4L2_FLASH_LED_MODE_FLASH). The unit should be milliamps (mA) if possible.

V4L2_CID_FLASH_TORCH_INTENSITY (integer)
Intensity of the flash LED in torch mode (V4L2_FLASH_LED_MODE_TORCH). The unit should be milliamps (mA) if possible. Setting this control may not be possible in presence of some faults. See V4L2_CID_FLASH_FAULT.

V4L2_CID_FLASH_INDICATOR_INTENSITY (integer)
Intensity of the indicator LED. The indicator LED may be fully independent

of the flash LED. The unit should be microamps (uA) if possible.

V4L2_CID_FLASH_FAULT (bitmask)
Faults related to the flash. The faults tell about specific problems in the flash chip itself or the LEDs attached to it. Faults may prevent further use of some of the flash controls. In particular, V4L2_CID_FLASH_LED_MODE is set to V4L2_FLASH_LED_MODE_NONE if the fault affects the flash LED. Exactly which faults have such an effect is chip dependent. Reading the faults resets the control and returns the chip to a usable state if possible.

V4L2_FLASH_FAULT_OVER_VOLTAGE	Flash controller voltage to the flash LED has exceeded the limit specific to the flash controller.
V4L2_FLASH_FAULT_TIMEOUT	The flash strobe was still on when the timeout set by the user —V4L2_CID_FLASH_TIMEOUT control —has expired. Not all flash controllers may set this in all such conditions.
V4L2_FLASH_FAULT_OVER_TEMPERATURE	The flash controller has overheated.
V4L2_FLASH_FAULT_SHORT_CIRCUIT	The short circuit protection of the flash controller has been triggered.
V4L2_FLASH_FAULT_OVER_CURRENT	Current in the LED power supply has exceeded the limit specific to the flash controller.
V4L2_FLASH_FAULT_INDICATOR	The flash controller has detected a short or open circuit condition on the indicator LED.
V4L2_FLASH_FAULT_UNDER_VOLTAGE	Flash controller voltage to the flash LED has been below the minimum limit specific to the flash controller.
V4L2_FLASH_FAULT_INPUT_VOLTAGE	The input voltage of the flash controller is below the limit under which strobing the flash at full current will not be possible.The condition persists until this flag is no longer set.
V4L2_FLASH_FAULT_LED_OVER_TEMPERATURE	The temperature of the LED has exceeded its allowed upper limit.

V4L2_CID_FLASH_CHARGE (boolean)
Enable or disable charging of the xenon flash capacitor.

V4L2_CID_FLASH_READY (boolean)
Is the flash ready to strobe? Xenon flashes require their capacitors charged before strobing. LED flashes often require a cooldown period after strobe during which another strobe will not be possible. This is a read-only control.

Image Source Control Reference

The Image Source control class is intended for low-level control of image source devices such as image sensors. The devices feature an analogue to digital converter and a bus transmitter to transmit the image data out of the device.

Image Source Control IDs

V4L2_CID_IMAGE_SOURCE_CLASS (class)
The IMAGE_SOURCE class descriptor.

V4L2_CID_VBLANK (integer)
Vertical blanking. The idle period after every frame during which no image data is produced. The unit of vertical blanking is a line. Every line has length of the image width plus horizontal blanking at the pixel rate defined by `V4L2_CID_PIXEL_RATE` control in the same sub-device.

V4L2_CID_HBLANK (integer)
Horizontal blanking. The idle period after every line of image data during which no image data is produced. The unit of horizontal blanking is pixels.

V4L2_CID_ANALOGUE_GAIN (integer)
Analogue gain is gain affecting all colour components in the pixel matrix. The gain operation is performed in the analogue domain before A/D conversion.

V4L2_CID_TEST_PATTERN_RED (integer)
Test pattern red colour component.

V4L2_CID_TEST_PATTERN_GREENR (integer)
Test pattern green (next to red) colour component.

V4L2_CID_TEST_PATTERN_BLUE (integer)
Test pattern blue colour component.

V4L2_CID_TEST_PATTERN_GREENB (integer)
Test pattern green (next to blue) colour component.

V4L2_CID_UNIT_CELL_SIZE (struct)
This control returns the unit cell size in nanometers. The struct `v4l2_area` provides the width and the height in separate fields to take into consideration asymmetric pixels. This control does not take into consideration any possible hardware binning. The unit cell consists of the whole area of the pixel, sensitive and non-sensitive. This control is required for automatic calibration of sensors/cameras.

Image Process Control Reference

The Image Process control class is intended for low-level control of image processing functions. Unlike `V4L2_CID_IMAGE_SOURCE_CLASS`, the controls in this class affect processing the image, and do not control capturing of it.

Image Process Control IDs

`V4L2_CID_IMAGE_PROC_CLASS (class)`
: The IMAGE_PROC class descriptor.

`V4L2_CID_LINK_FREQ (integer menu)`
: Data bus frequency. Together with the media bus pixel code, bus type (clock cycles per sample), the data bus frequency defines the pixel rate (`V4L2_CID_PIXEL_RATE`) in the pixel array (or possibly elsewhere, if the device is not an image sensor). The frame rate can be calculated from the pixel clock, image width and height and horizontal and vertical blanking. While the pixel rate control may be defined elsewhere than in the subdev containing the pixel array, the frame rate cannot be obtained from that information. This is because only on the pixel array it can be assumed that the vertical and horizontal blanking information is exact: no other blanking is allowed in the pixel array. The selection of frame rate is performed by selecting the desired horizontal and vertical blanking. The unit of this control is Hz.

`V4L2_CID_PIXEL_RATE (64-bit integer)`
: Pixel rate in the source pads of the subdev. This control is read-only and its unit is pixels / second.

`V4L2_CID_TEST_PATTERN (menu)`
: Some capture/display/sensor devices have the capability to generate test pattern images. These hardware specific test patterns can be used to test if a device is working properly.

`V4L2_CID_DEINTERLACING_MODE (menu)`
: The video deinterlacing mode (such as Bob, Weave, ⋯). The menu items are driver specific and are documented in uapi-v4l-drivers.

`V4L2_CID_DIGITAL_GAIN (integer)`
: Digital gain is the value by which all colour components are multiplied by. Typically the digital gain applied is the control value divided by e.g. 0x100, meaning that to get no digital gain the control value needs to be 0x100. The no-gain configuration is also typically the default.

Codec Control Reference

Below all controls within the Codec control class are described. First the generic controls, then controls specific for certain hardware.

Note: These controls are applicable to all codecs and not just MPEG. The defines are prefixed with V4L2_CID_MPEG/V4L2_MPEG as the controls were originally made for MPEG codecs and later extended to cover all encoding formats.

Generic Codec Controls

Codec Control IDs

`V4L2_CID_MPEG_CLASS (class)`
The Codec class descriptor. Calling *ioctls VIDIOC_QUERYCTRL, VIDIOC_QUERY_EXT_CTRL and VIDIOC_QUERYMENU* for this control will return a description of this control class. This description can be used as the caption of a Tab page in a GUI, for example.

`V4L2_CID_MPEG_STREAM_TYPE`
(enum)

enum v4l2_mpeg_stream_type -
The MPEG-1, -2 or -4 output stream type. One cannot assume anything here. Each hardware MPEG encoder tends to support different subsets of the available MPEG stream types. This control is specific to multiplexed MPEG streams. The currently defined stream types are:

`V4L2_MPEG_STREAM_TYPE_MPEG2_PS`	MPEG-2 program stream
`V4L2_MPEG_STREAM_TYPE_MPEG2_TS`	MPEG-2 transport stream
`V4L2_MPEG_STREAM_TYPE_MPEG1_SS`	MPEG-1 system stream
`V4L2_MPEG_STREAM_TYPE_MPEG2_DVD`	MPEG-2 DVD-compatible stream
`V4L2_MPEG_STREAM_TYPE_MPEG1_VCD`	MPEG-1 VCD-compatible stream
`V4L2_MPEG_STREAM_TYPE_MPEG2_SVCD`	MPEG-2 SVCD-compatible stream

`V4L2_CID_MPEG_STREAM_PID_PMT (integer)`
Program Map Table Packet ID for the MPEG transport stream (default 16)

`V4L2_CID_MPEG_STREAM_PID_AUDIO (integer)`
Audio Packet ID for the MPEG transport stream (default 256)

`V4L2_CID_MPEG_STREAM_PID_VIDEO (integer)`
Video Packet ID for the MPEG transport stream (default 260)

`V4L2_CID_MPEG_STREAM_PID_PCR (integer)`
Packet ID for the MPEG transport stream carrying PCR fields (default 259)

`V4L2_CID_MPEG_STREAM_PES_ID_AUDIO (integer)`
Audio ID for MPEG PES

`V4L2_CID_MPEG_STREAM_PES_ID_VIDEO (integer)`
Video ID for MPEG PES

V4L2_CID_MPEG_STREAM_VBI_FMT
(enum)

enum v4l2_mpeg_stream_vbi_fmt -
Some cards can embed VBI data (e. g. Closed Caption, Teletext) into the MPEG stream. This control selects whether VBI data should be embedded, and if so, what embedding method should be used. The list of possible VBI formats depends on the driver. The currently defined VBI format types are:

V4L2_MPEG_STREAM_VBI_FMT_NONE	No VBI in the MPEG stream
V4L2_MPEG_STREAM_VBI_FMT_IVTV	VBI in private packets, IVTV format (documented in the kernel sources in the file Documentation/userspace-api/media/drivers/cx2341x-uapi.rst)

V4L2_CID_MPEG_AUDIO_SAMPLING_FREQ
(enum)

enum v4l2_mpeg_audio_sampling_freq -
MPEG Audio sampling frequency. Possible values are:

V4L2_MPEG_AUDIO_SAMPLING_FREQ_44100	44.1 kHz
V4L2_MPEG_AUDIO_SAMPLING_FREQ_48000	48 kHz
V4L2_MPEG_AUDIO_SAMPLING_FREQ_32000	32 kHz

V4L2_CID_MPEG_AUDIO_ENCODING
(enum)

enum v4l2_mpeg_audio_encoding -
MPEG Audio encoding. This control is specific to multiplexed MPEG streams. Possible values are:

V4L2_MPEG_AUDIO_ENCODING_LAYER_1	MPEG-1/2 Layer I encoding
V4L2_MPEG_AUDIO_ENCODING_LAYER_2	MPEG-1/2 Layer II encoding
V4L2_MPEG_AUDIO_ENCODING_LAYER_3	MPEG-1/2 Layer III encoding
V4L2_MPEG_AUDIO_ENCODING_AAC	MPEG-2/4 AAC (Advanced Audio Coding)
V4L2_MPEG_AUDIO_ENCODING_AC3	AC-3 aka ATSC A/52 encoding

V4L2_CID_MPEG_AUDIO_L1_BITRATE
(enum)

enum v4l2_mpeg_audio_l1_bitrate -
MPEG-1/2 Layer I bitrate. Possible values are:

V4L2_MPEG_AUDIO_L1_BITRATE_32K	32 kbit/s
V4L2_MPEG_AUDIO_L1_BITRATE_64K	64 kbit/s
V4L2_MPEG_AUDIO_L1_BITRATE_96K	96 kbit/s
V4L2_MPEG_AUDIO_L1_BITRATE_128K	128 kbit/s
V4L2_MPEG_AUDIO_L1_BITRATE_160K	160 kbit/s
V4L2_MPEG_AUDIO_L1_BITRATE_192K	192 kbit/s
V4L2_MPEG_AUDIO_L1_BITRATE_224K	224 kbit/s
V4L2_MPEG_AUDIO_L1_BITRATE_256K	256 kbit/s
V4L2_MPEG_AUDIO_L1_BITRATE_288K	288 kbit/s
V4L2_MPEG_AUDIO_L1_BITRATE_320K	320 kbit/s
V4L2_MPEG_AUDIO_L1_BITRATE_352K	352 kbit/s
V4L2_MPEG_AUDIO_L1_BITRATE_384K	384 kbit/s
V4L2_MPEG_AUDIO_L1_BITRATE_416K	416 kbit/s
V4L2_MPEG_AUDIO_L1_BITRATE_448K	448 kbit/s

V4L2_CID_MPEG_AUDIO_L2_BITRATE
(enum)

enum v4l2_mpeg_audio_l2_bitrate -
MPEG-1/2 Layer II bitrate. Possible values are:

V4L2_MPEG_AUDIO_L2_BITRATE_32K	32 kbit/s
V4L2_MPEG_AUDIO_L2_BITRATE_48K	48 kbit/s
V4L2_MPEG_AUDIO_L2_BITRATE_56K	56 kbit/s
V4L2_MPEG_AUDIO_L2_BITRATE_64K	64 kbit/s
V4L2_MPEG_AUDIO_L2_BITRATE_80K	80 kbit/s
V4L2_MPEG_AUDIO_L2_BITRATE_96K	96 kbit/s
V4L2_MPEG_AUDIO_L2_BITRATE_112K	112 kbit/s
V4L2_MPEG_AUDIO_L2_BITRATE_128K	128 kbit/s
V4L2_MPEG_AUDIO_L2_BITRATE_160K	160 kbit/s
V4L2_MPEG_AUDIO_L2_BITRATE_192K	192 kbit/s
V4L2_MPEG_AUDIO_L2_BITRATE_224K	224 kbit/s
V4L2_MPEG_AUDIO_L2_BITRATE_256K	256 kbit/s
V4L2_MPEG_AUDIO_L2_BITRATE_320K	320 kbit/s
V4L2_MPEG_AUDIO_L2_BITRATE_384K	384 kbit/s

V4L2_CID_MPEG_AUDIO_L3_BITRATE
(enum)

enum v4l2_mpeg_audio_l3_bitrate -
MPEG-1/2 Layer III bitrate. Possible values are:

V4L2_MPEG_AUDIO_L3_BITRATE_32K	32 kbit/s
V4L2_MPEG_AUDIO_L3_BITRATE_40K	40 kbit/s
V4L2_MPEG_AUDIO_L3_BITRATE_48K	48 kbit/s
V4L2_MPEG_AUDIO_L3_BITRATE_56K	56 kbit/s
V4L2_MPEG_AUDIO_L3_BITRATE_64K	64 kbit/s
V4L2_MPEG_AUDIO_L3_BITRATE_80K	80 kbit/s
V4L2_MPEG_AUDIO_L3_BITRATE_96K	96 kbit/s
V4L2_MPEG_AUDIO_L3_BITRATE_112K	112 kbit/s
V4L2_MPEG_AUDIO_L3_BITRATE_128K	128 kbit/s
V4L2_MPEG_AUDIO_L3_BITRATE_160K	160 kbit/s
V4L2_MPEG_AUDIO_L3_BITRATE_192K	192 kbit/s
V4L2_MPEG_AUDIO_L3_BITRATE_224K	224 kbit/s
V4L2_MPEG_AUDIO_L3_BITRATE_256K	256 kbit/s
V4L2_MPEG_AUDIO_L3_BITRATE_320K	320 kbit/s

V4L2_CID_MPEG_AUDIO_AAC_BITRATE (integer)
AAC bitrate in bits per second.

V4L2_CID_MPEG_AUDIO_AC3_BITRATE
(enum)

enum v4l2_mpeg_audio_ac3_bitrate -
AC-3 bitrate. Possible values are:

V4L2_MPEG_AUDIO_AC3_BITRATE_32K	32 kbit/s
V4L2_MPEG_AUDIO_AC3_BITRATE_40K	40 kbit/s
V4L2_MPEG_AUDIO_AC3_BITRATE_48K	48 kbit/s
V4L2_MPEG_AUDIO_AC3_BITRATE_56K	56 kbit/s
V4L2_MPEG_AUDIO_AC3_BITRATE_64K	64 kbit/s
V4L2_MPEG_AUDIO_AC3_BITRATE_80K	80 kbit/s
V4L2_MPEG_AUDIO_AC3_BITRATE_96K	96 kbit/s
V4L2_MPEG_AUDIO_AC3_BITRATE_112K	112 kbit/s
V4L2_MPEG_AUDIO_AC3_BITRATE_128K	128 kbit/s
V4L2_MPEG_AUDIO_AC3_BITRATE_160K	160 kbit/s
V4L2_MPEG_AUDIO_AC3_BITRATE_192K	192 kbit/s
V4L2_MPEG_AUDIO_AC3_BITRATE_224K	224 kbit/s
V4L2_MPEG_AUDIO_AC3_BITRATE_256K	256 kbit/s
V4L2_MPEG_AUDIO_AC3_BITRATE_320K	320 kbit/s
V4L2_MPEG_AUDIO_AC3_BITRATE_384K	384 kbit/s
V4L2_MPEG_AUDIO_AC3_BITRATE_448K	448 kbit/s
V4L2_MPEG_AUDIO_AC3_BITRATE_512K	512 kbit/s
V4L2_MPEG_AUDIO_AC3_BITRATE_576K	576 kbit/s
V4L2_MPEG_AUDIO_AC3_BITRATE_640K	640 kbit/s

V4L2_CID_MPEG_AUDIO_MODE
(enum)

enum v4l2_mpeg_audio_mode -
MPEG Audio mode. Possible values are:

V4L2_MPEG_AUDIO_MODE_STEREO	Stereo
V4L2_MPEG_AUDIO_MODE_JOINT_STEREO	Joint Stereo
V4L2_MPEG_AUDIO_MODE_DUAL	Bilingual
V4L2_MPEG_AUDIO_MODE_MONO	Mono

V4L2_CID_MPEG_AUDIO_MODE_EXTENSION
(enum)

enum v4l2_mpeg_audio_mode_extension -
Joint Stereo audio mode extension. In Layer I and II they indicate which subbands are in intensity stereo. All other subbands are coded in stereo. Layer III is not (yet) supported. Possible values are:

V4L2_MPEG_AUDIO_MODE_EXTENSION_BOUND_4	Subbands 4-31 in intensity stereo
V4L2_MPEG_AUDIO_MODE_EXTENSION_BOUND_8	Subbands 8-31 in intensity stereo
V4L2_MPEG_AUDIO_MODE_EXTENSION_BOUND_1	Subbands 12-31 in intensity stereo
V4L2_MPEG_AUDIO_MODE_EXTENSION_BOUND_1	Subbands 16-31 in intensity stereo

V4L2_CID_MPEG_AUDIO_EMPHASIS
(enum)

enum v4l2_mpeg_audio_emphasis -
Audio Emphasis. Possible values are:

V4L2_MPEG_AUDIO_EMPHASIS_NONE	None
V4L2_MPEG_AUDIO_EMPHASIS_50_DIV_15_uS	50/15 microsecond emphasis
V4L2_MPEG_AUDIO_EMPHASIS_CCITT_J17	CCITT J.17

V4L2_CID_MPEG_AUDIO_CRC
(enum)

enum v4l2_mpeg_audio_crc -
CRC method. Possible values are:

V4L2_MPEG_AUDIO_CRC_NONE	None
V4L2_MPEG_AUDIO_CRC_CRC16	16 bit parity check

V4L2_CID_MPEG_AUDIO_MUTE (boolean)
Mutes the audio when capturing. This is not done by muting audio hardware, which can still produce a slight hiss, but in the encoder itself, guaranteeing a fixed and reproducible audio bitstream. 0 = unmuted, 1 = muted.

V4L2_CID_MPEG_AUDIO_DEC_PLAYBACK
(enum)

enum v4l2_mpeg_audio_dec_playback -
Determines how monolingual audio should be played back. Possible values are:

V4L2_MPEG_AUDIO_DEC_PLAYBACK_AUTO	Automatically determines the best playback mode.
V4L2_MPEG_AUDIO_DEC_PLAYBACK_STEREO	Stereo playback.
V4L2_MPEG_AUDIO_DEC_PLAYBACK_LEFT	Left channel playback.
V4L2_MPEG_AUDIO_DEC_PLAYBACK_RIGHT	Right channel playback.
V4L2_MPEG_AUDIO_DEC_PLAYBACK_MONO	Mono playback.
V4L2_MPEG_AUDIO_DEC_PLAYBACK_SWAPPED_STEREO	Stereo playback with swapped left and right channels.

V4L2_CID_MPEG_AUDIO_DEC_MULTILINGUAL_PLAYBACK
(enum)

enum v4l2_mpeg_audio_dec_playback -
Determines how multilingual audio should be played back.

V4L2_CID_MPEG_VIDEO_ENCODING
(enum)

enum v4l2_mpeg_video_encoding -
MPEG Video encoding method. This control is specific to multiplexed MPEG streams. Possible values are:

V4L2_MPEG_VIDEO_ENCODING_MPEG_1	MPEG-1 Video encoding
V4L2_MPEG_VIDEO_ENCODING_MPEG_2	MPEG-2 Video encoding
V4L2_MPEG_VIDEO_ENCODING_MPEG_4_AVC	MPEG-4 AVC (H.264) Video encoding

V4L2_CID_MPEG_VIDEO_ASPECT
(enum)

enum v4l2_mpeg_video_aspect -
Video aspect. Possible values are:

V4L2_MPEG_VIDEO_ASPECT_1x1
V4L2_MPEG_VIDEO_ASPECT_4x3
V4L2_MPEG_VIDEO_ASPECT_16x9
V4L2_MPEG_VIDEO_ASPECT_221x100

V4L2_CID_MPEG_VIDEO_B_FRAMES (integer)
Number of B-Frames (default 2)

V4L2_CID_MPEG_VIDEO_GOP_SIZE (integer)
GOP size (default 12)

V4L2_CID_MPEG_VIDEO_GOP_CLOSURE (boolean)
GOP closure (default 1)

V4L2_CID_MPEG_VIDEO_PULLDOWN (boolean)
Enable 3:2 pulldown (default 0)

V4L2_CID_MPEG_VIDEO_BITRATE_MODE
(enum)

enum v4l2_mpeg_video_bitrate_mode -
Video bitrate mode. Possible values are:

V4L2_MPEG_VIDEO_BITRATE_MODE_VBR	Variable bitrate
V4L2_MPEG_VIDEO_BITRATE_MODE_CBR	Constant bitrate
V4L2_MPEG_VIDEO_BITRATE_MODE_CQ	Constant quality

V4L2_CID_MPEG_VIDEO_BITRATE (integer)
Video bitrate in bits per second.

V4L2_CID_MPEG_VIDEO_BITRATE_PEAK (integer)
Peak video bitrate in bits per second. Must be larger or equal to the average video bitrate. It is ignored if the video bitrate mode is set to constant bitrate.

V4L2_CID_MPEG_VIDEO_CONSTANT_QUALITY (integer)
Constant quality level control. This control is applicable when V4L2_CID_MPEG_VIDEO_BITRATE_MODE value is V4L2_MPEG_VIDEO_BITRATE_MODE_CQ. Valid range is 1 to 100 where 1 indicates lowest quality and 100 indicates highest quality. Encoder will decide the appropriate quantization parameter and bitrate to produce requested frame quality.

V4L2_CID_MPEG_VIDEO_FRAME_SKIP_MODE (enum)

enum v4l2_mpeg_video_frame_skip_mode -
Indicates in what conditions the encoder should skip frames. If encoding a frame would cause the encoded stream to be larger then a chosen data limit then the frame will be skipped. Possible values are:

V4L2_MPEG_FRAME_SKIP_MODE_DISABLED	Frame skip mode is disabled.
V4L2_MPEG_FRAME_SKIP_MODE_LEVEL_LIMIT	Frame skip mode enabled and buffer limit is set by the chosen level and is defined by the standard.
V4L2_MPEG_FRAME_SKIP_MODE_BUF_LIMIT	Frame skip mode enabled and buffer limit is set by the *VBV (MPEG1/2/4)* or *CPB (H264) buffer size* control.

V4L2_CID_MPEG_VIDEO_TEMPORAL_DECIMATION (integer)
For every captured frame, skip this many subsequent frames (default 0).

V4L2_CID_MPEG_VIDEO_MUTE (boolean)
"Mutes" the video to a fixed color when capturing. This is useful for testing, to produce a fixed video bitstream. 0 = unmuted, 1 = muted.

V4L2_CID_MPEG_VIDEO_MUTE_YUV (integer)
Sets the "mute" color of the video. The supplied 32-bit integer is interpreted as follows (bit 0 = least significant bit):

Bit 0:7	V chrominance information
Bit 8:15	U chrominance information
Bit 16:23	Y luminance information
Bit 24:31	Must be zero.

V4L2_CID_MPEG_VIDEO_DEC_PTS (integer64)
This read-only control returns the 33-bit video Presentation Time Stamp as defined in ITU T-REC-H.222.0 and ISO/IEC 13818-1 of the currently displayed frame. This is the same PTS as is used in *ioctl VIDIOC_DECODER_CMD, VIDIOC_TRY_DECODER_CMD*.

V4L2_CID_MPEG_VIDEO_DEC_FRAME (integer64)
This read-only control returns the frame counter of the frame that is currently displayed (decoded). This value is reset to 0 whenever the decoder is started.

V4L2_CID_MPEG_VIDEO_DECODER_SLICE_INTERFACE (boolean)
If enabled the decoder expects to receive a single slice per buffer, otherwise the decoder expects a single frame in per buffer. Applicable to the decoder, all codecs.

V4L2_CID_MPEG_VIDEO_H264_VUI_SAR_ENABLE (boolean)
Enable writing sample aspect ratio in the Video Usability Information. Applicable to the H264 encoder.

V4L2_CID_MPEG_VIDEO_H264_VUI_SAR_IDC
(enum)

enum v4l2_mpeg_video_h264_vui_sar_idc -
VUI sample aspect ratio indicator for H.264 encoding. The value is defined in the table E-1 in the standard. Applicable to the H264 encoder.

V4L2_MPEG_VIDEO_H264_VUI_SAR_IDC_UNSPECIFIED	Unspecified
V4L2_MPEG_VIDEO_H264_VUI_SAR_IDC_1x1	1x1
V4L2_MPEG_VIDEO_H264_VUI_SAR_IDC_12x11	12x11
V4L2_MPEG_VIDEO_H264_VUI_SAR_IDC_10x11	10x11
V4L2_MPEG_VIDEO_H264_VUI_SAR_IDC_16x11	16x11
V4L2_MPEG_VIDEO_H264_VUI_SAR_IDC_40x33	40x33
V4L2_MPEG_VIDEO_H264_VUI_SAR_IDC_24x11	24x11
V4L2_MPEG_VIDEO_H264_VUI_SAR_IDC_20x11	20x11
V4L2_MPEG_VIDEO_H264_VUI_SAR_IDC_32x11	32x11
V4L2_MPEG_VIDEO_H264_VUI_SAR_IDC_80x33	80x33
V4L2_MPEG_VIDEO_H264_VUI_SAR_IDC_18x11	18x11
V4L2_MPEG_VIDEO_H264_VUI_SAR_IDC_15x11	15x11
V4L2_MPEG_VIDEO_H264_VUI_SAR_IDC_64x33	64x33
V4L2_MPEG_VIDEO_H264_VUI_SAR_IDC_160x99	160x99
V4L2_MPEG_VIDEO_H264_VUI_SAR_IDC_4x3	4x3
V4L2_MPEG_VIDEO_H264_VUI_SAR_IDC_3x2	3x2
V4L2_MPEG_VIDEO_H264_VUI_SAR_IDC_2x1	2x1
V4L2_MPEG_VIDEO_H264_VUI_SAR_IDC_EXTENDED	Extended SAR

V4L2_CID_MPEG_VIDEO_H264_VUI_EXT_SAR_WIDTH (integer)
Extended sample aspect ratio width for H.264 VUI encoding. Applicable to the H264 encoder.

V4L2_CID_MPEG_VIDEO_H264_VUI_EXT_SAR_HEIGHT (integer)
Extended sample aspect ratio height for H.264 VUI encoding. Applicable to the H264 encoder.

V4L2_CID_MPEG_VIDEO_H264_LEVEL

(enum)

enum v4l2_mpeg_video_h264_level -
The level information for the H264 video elementary stream. Applicable to the H264 encoder. Possible values are:

V4L2_MPEG_VIDEO_H264_LEVEL_1_0	Level 1.0
V4L2_MPEG_VIDEO_H264_LEVEL_1B	Level 1B
V4L2_MPEG_VIDEO_H264_LEVEL_1_1	Level 1.1
V4L2_MPEG_VIDEO_H264_LEVEL_1_2	Level 1.2
V4L2_MPEG_VIDEO_H264_LEVEL_1_3	Level 1.3
V4L2_MPEG_VIDEO_H264_LEVEL_2_0	Level 2.0
V4L2_MPEG_VIDEO_H264_LEVEL_2_1	Level 2.1
V4L2_MPEG_VIDEO_H264_LEVEL_2_2	Level 2.2
V4L2_MPEG_VIDEO_H264_LEVEL_3_0	Level 3.0
V4L2_MPEG_VIDEO_H264_LEVEL_3_1	Level 3.1
V4L2_MPEG_VIDEO_H264_LEVEL_3_2	Level 3.2
V4L2_MPEG_VIDEO_H264_LEVEL_4_0	Level 4.0
V4L2_MPEG_VIDEO_H264_LEVEL_4_1	Level 4.1
V4L2_MPEG_VIDEO_H264_LEVEL_4_2	Level 4.2
V4L2_MPEG_VIDEO_H264_LEVEL_5_0	Level 5.0
V4L2_MPEG_VIDEO_H264_LEVEL_5_1	Level 5.1
V4L2_MPEG_VIDEO_H264_LEVEL_5_2	Level 5.2
V4L2_MPEG_VIDEO_H264_LEVEL_6_0	Level 6.0
V4L2_MPEG_VIDEO_H264_LEVEL_6_1	Level 6.1
V4L2_MPEG_VIDEO_H264_LEVEL_6_2	Level 6.2

V4L2_CID_MPEG_VIDEO_MPEG2_LEVEL
(enum)

enum v4l2_mpeg_video_mpeg2_level -
The level information for the MPEG2 elementary stream. Applicable to MPEG2 codecs. Possible values are:

V4L2_MPEG_VIDEO_MPEG2_LEVEL_LOW	Low Level (LL)
V4L2_MPEG_VIDEO_MPEG2_LEVEL_MAIN	Main Level (ML)
V4L2_MPEG_VIDEO_MPEG2_LEVEL_HIGH_1440	High-1440 Level (H-14)
V4L2_MPEG_VIDEO_MPEG2_LEVEL_HIGH	High Level (HL)

V4L2_CID_MPEG_VIDEO_MPEG4_LEVEL
(enum)

enum v4l2_mpeg_video_mpeg4_level -
The level information for the MPEG4 elementary stream. Applicable to the MPEG4 encoder. Possible values are:

V4L2_MPEG_VIDEO_MPEG4_LEVEL_0	Level 0
V4L2_MPEG_VIDEO_MPEG4_LEVEL_0B	Level 0b
V4L2_MPEG_VIDEO_MPEG4_LEVEL_1	Level 1
V4L2_MPEG_VIDEO_MPEG4_LEVEL_2	Level 2
V4L2_MPEG_VIDEO_MPEG4_LEVEL_3	Level 3
V4L2_MPEG_VIDEO_MPEG4_LEVEL_3B	Level 3b
V4L2_MPEG_VIDEO_MPEG4_LEVEL_4	Level 4
V4L2_MPEG_VIDEO_MPEG4_LEVEL_5	Level 5

V4L2_CID_MPEG_VIDEO_H264_PROFILE
(enum)

enum v4l2_mpeg_video_h264_profile -
The profile information for H264. Applicable to the H264 encoder. Possible values are:

V4L2_MPEG_VIDEO_H264_PROFILE_BASELINE	Baseline profile
V4L2_MPEG_VIDEO_H264_PROFILE_CONSTRAINED_BASE	Constrained Baseline profile
V4L2_MPEG_VIDEO_H264_PROFILE_MAIN	Main profile
V4L2_MPEG_VIDEO_H264_PROFILE_EXTENDED	Extended profile
V4L2_MPEG_VIDEO_H264_PROFILE_HIGH	High profile
V4L2_MPEG_VIDEO_H264_PROFILE_HIGH_10	High 10 profile
V4L2_MPEG_VIDEO_H264_PROFILE_HIGH_422	High 422 profile
V4L2_MPEG_VIDEO_H264_PROFILE_HIGH_444_PREDICT	High 444 Predictive profile
V4L2_MPEG_VIDEO_H264_PROFILE_HIGH_10_INTRA	High 10 Intra profile
V4L2_MPEG_VIDEO_H264_PROFILE_HIGH_422_INTRA	High 422 Intra profile
V4L2_MPEG_VIDEO_H264_PROFILE_HIGH_444_INTRA	High 444 Intra profile
V4L2_MPEG_VIDEO_H264_PROFILE_CAVLC_444_INTRA	CAVLC 444 Intra profile
V4L2_MPEG_VIDEO_H264_PROFILE_SCALABLE_BASELIN	Scalable Baseline profile
V4L2_MPEG_VIDEO_H264_PROFILE_SCALABLE_HIGH	Scalable High profile
V4L2_MPEG_VIDEO_H264_PROFILE_SCALABLE_HIGH_IN	Scalable High Intra profile
V4L2_MPEG_VIDEO_H264_PROFILE_STEREO_HIGH	Stereo High profile
V4L2_MPEG_VIDEO_H264_PROFILE_MULTIVIEW_HIGH	Multiview High profile
V4L2_MPEG_VIDEO_H264_PROFILE_CONSTRAINED_HIGH	Constrained High profile

V4L2_CID_MPEG_VIDEO_MPEG2_PROFILE
(enum)

enum v4l2_mpeg_video_mpeg2_profile -
The profile information for MPEG2. Applicable to MPEG2 codecs. Possible values are:

V4L2_MPEG_VIDEO_MPEG2_PROFILE_SIMPLE	Simple profile (SP)
V4L2_MPEG_VIDEO_MPEG2_PROFILE_MAIN	Main profile (MP)
V4L2_MPEG_VIDEO_MPEG2_PROFILE_SNR_SCALABLE	SNR Scalable profile (SNR)
V4L2_MPEG_VIDEO_MPEG2_PROFILE_SPATIALLY_SCAI	Spatially Scalable profile (Spt)
V4L2_MPEG_VIDEO_MPEG2_PROFILE_HIGH	High profile (HP)
V4L2_MPEG_VIDEO_MPEG2_PROFILE_MULTIVIEW	Multi-view profile (MVP)

V4L2_CID_MPEG_VIDEO_MPEG4_PROFILE
(enum)

enum v4l2_mpeg_video_mpeg4_profile -
The profile information for MPEG4. Applicable to the MPEG4 encoder. Possible values are:

V4L2_MPEG_VIDEO_MPEG4_PROFILE_SIMPLE	Simple profile
V4L2_MPEG_VIDEO_MPEG4_PROFILE_ADVANCED_SIMPLE	Advanced Simple profile
V4L2_MPEG_VIDEO_MPEG4_PROFILE_CORE	Core profile
V4L2_MPEG_VIDEO_MPEG4_PROFILE_SIMPLE_SCALABLE	Simple Scalable profile
V4L2_MPEG_VIDEO_MPEG4_PROFILE_ADVANCED_CODING_EFI	

V4L2_CID_MPEG_VIDEO_MAX_REF_PIC (integer)
The maximum number of reference pictures used for encoding. Applicable to the encoder.

V4L2_CID_MPEG_VIDEO_MULTI_SLICE_MODE
(enum)

enum v4l2_mpeg_video_multi_slice_mode -
Determines how the encoder should handle division of frame into slices. Applicable to the encoder. Possible values are:

V4L2_MPEG_VIDEO_MULTI_SLICE_MODE_SINGLE	Single slice per frame.
V4L2_MPEG_VIDEO_MULTI_SLICE_MODE_MAX_MB	Multiple slices with set maximum number of macroblocks per slice.
V4L2_MPEG_VIDEO_MULTI_SLICE_MODE_MAX_BYTES	Multiple slice with set maximum size in bytes per slice.

V4L2_CID_MPEG_VIDEO_MULTI_SLICE_MAX_MB (integer)
The maximum number of macroblocks in a slice. Used when V4L2_CID_MPEG_VIDEO_MULTI_SLICE_MODE is set to V4L2_MPEG_VIDEO_MULTI_SLICE_MODE_MAX_MB. Applicable to the encoder.

V4L2_CID_MPEG_VIDEO_MULTI_SLICE_MAX_BYTES (integer)
The maximum size of a slice in bytes. Used when V4L2_CID_MPEG_VIDEO_MULTI_SLICE_MODE is set to V4L2_MPEG_VIDEO_MULTI_SLICE_MODE_MAX_BYTES. Applicable to the encoder.

V4L2_CID_MPEG_VIDEO_H264_LOOP_FILTER_MODE
(enum)

enum v4l2_mpeg_video_h264_loop_filter_mode -
Loop filter mode for H264 encoder. Possible values are:

V4L2_MPEG_VIDEO_H264_LOOP_FILTER_MODE_ENABLED	Loop filter is enabled.
V4L2_MPEG_VIDEO_H264_LOOP_FILTER_MODE_DISABLED	Loop filter is disabled.
V4L2_MPEG_VIDEO_H264_LOOP_FILTER_MODE_DISABLED_AT_SLICE_BOUNDARY	Loop filter is disabled at the slice boundary.

V4L2_CID_MPEG_VIDEO_H264_LOOP_FILTER_ALPHA (integer)
Loop filter alpha coefficient, defined in the H264 standard. This value corresponds to the slice_alpha_c0_offset_div2 slice header field, and should be in the range of -6 to +6, inclusive. The actual alpha offset FilterOffsetA is twice this value. Applicable to the H264 encoder.

V4L2_CID_MPEG_VIDEO_H264_LOOP_FILTER_BETA (integer)
Loop filter beta coefficient, defined in the H264 standard. This corresponds to the slice_beta_offset_div2 slice header field, and should be in the range of -6 to +6, inclusive. The actual beta offset FilterOffsetB is twice this value. Applicable to the H264 encoder.

V4L2_CID_MPEG_VIDEO_H264_ENTROPY_MODE
(enum)

enum v4l2_mpeg_video_h264_entropy_mode -
Entropy coding mode for H264 - CABAC/CAVALC. Applicable to the H264 encoder. Possible values are:

V4L2_MPEG_VIDEO_H264_ENTROPY_MODE_CAVLC	Use CAVLC entropy coding.
V4L2_MPEG_VIDEO_H264_ENTROPY_MODE_CABAC	Use CABAC entropy coding.

V4L2_CID_MPEG_VIDEO_H264_8X8_TRANSFORM (boolean)
Enable 8X8 transform for H264. Applicable to the H264 encoder.

V4L2_CID_MPEG_VIDEO_H264_CONSTRAINED_INTRA_PREDICTION (boolean)
Enable constrained intra prediction for H264. Applicable to the H264 encoder.

V4L2_CID_MPEG_VIDEO_H264_CHROMA_QP_INDEX_OFFSET (integer)
Specify the offset that should be added to the luma quantization parameter to determine the chroma quantization parameter. Applicable to the H264 encoder.

V4L2_CID_MPEG_VIDEO_CYCLIC_INTRA_REFRESH_MB (integer)
Cyclic intra macroblock refresh. This is the number of continuous macroblocks refreshed every frame. Each frame a successive set of macroblocks is refreshed until the cycle completes and starts from the top of the frame. Applicable to H264, H263 and MPEG4 encoder.

V4L2_CID_MPEG_VIDEO_FRAME_RC_ENABLE (boolean)
Frame level rate control enable. If this control is disabled then the quantization parameter for each frame type is constant and set with appropriate

controls (e.g. V4L2_CID_MPEG_VIDEO_H263_I_FRAME_QP). If frame rate control is enabled then quantization parameter is adjusted to meet the chosen bitrate. Minimum and maximum value for the quantization parameter can be set with appropriate controls (e.g. V4L2_CID_MPEG_VIDEO_H263_MIN_QP). Applicable to encoders.

V4L2_CID_MPEG_VIDEO_MB_RC_ENABLE (boolean)
Macroblock level rate control enable. Applicable to the MPEG4 and H264 encoders.

V4L2_CID_MPEG_VIDEO_MPEG4_QPEL (boolean)
Quarter pixel motion estimation for MPEG4. Applicable to the MPEG4 encoder.

V4L2_CID_MPEG_VIDEO_H263_I_FRAME_QP (integer)
Quantization parameter for an I frame for H263. Valid range: from 1 to 31.

V4L2_CID_MPEG_VIDEO_H263_MIN_QP (integer)
Minimum quantization parameter for H263. Valid range: from 1 to 31.

V4L2_CID_MPEG_VIDEO_H263_MAX_QP (integer)
Maximum quantization parameter for H263. Valid range: from 1 to 31.

V4L2_CID_MPEG_VIDEO_H263_P_FRAME_QP (integer)
Quantization parameter for an P frame for H263. Valid range: from 1 to 31.

V4L2_CID_MPEG_VIDEO_H263_B_FRAME_QP (integer)
Quantization parameter for an B frame for H263. Valid range: from 1 to 31.

V4L2_CID_MPEG_VIDEO_H264_I_FRAME_QP (integer)
Quantization parameter for an I frame for H264. Valid range: from 0 to 51.

V4L2_CID_MPEG_VIDEO_H264_MIN_QP (integer)
Minimum quantization parameter for H264. Valid range: from 0 to 51.

V4L2_CID_MPEG_VIDEO_H264_MAX_QP (integer)
Maximum quantization parameter for H264. Valid range: from 0 to 51.

V4L2_CID_MPEG_VIDEO_H264_P_FRAME_QP (integer)
Quantization parameter for an P frame for H264. Valid range: from 0 to 51.

V4L2_CID_MPEG_VIDEO_H264_B_FRAME_QP (integer)
Quantization parameter for an B frame for H264. Valid range: from 0 to 51.

V4L2_CID_MPEG_VIDEO_H264_I_FRAME_MIN_QP (integer)
Minimum quantization parameter for the H264 I frame to limit I frame quality to a range. Valid range: from 0 to 51. If V4L2_CID_MPEG_VIDEO_H264_MIN_QP is also set, the quantization parameter should be chosen to meet both requirements.

V4L2_CID_MPEG_VIDEO_H264_I_FRAME_MAX_QP (integer)
Maximum quantization parameter for the H264 I frame to limit I frame quality to a range. Valid range: from 0 to 51. If V4L2_CID_MPEG_VIDEO_H264_MAX_QP is also set, the quantization parameter should be chosen to meet both requirements.

V4L2_CID_MPEG_VIDEO_H264_P_FRAME_MIN_QP (integer)
Minimum quantization parameter for the H264 P frame to limit P frame quality to a range. Valid range: from 0 to 51. If

V4L2_CID_MPEG_VIDEO_H264_MIN_QP is also set, the quantization parameter should be chosen to meet both requirements.

V4L2_CID_MPEG_VIDEO_H264_P_FRAME_MAX_QP (integer)
Maximum quantization parameter for the H264 P frame to limit P frame quality to a range. Valid range: from 0 to 51. If V4L2_CID_MPEG_VIDEO_H264_MAX_QP is also set, the quantization parameter should be chosen to meet both requirements.

V4L2_CID_MPEG_VIDEO_MPEG4_I_FRAME_QP (integer)
Quantization parameter for an I frame for MPEG4. Valid range: from 1 to 31.

V4L2_CID_MPEG_VIDEO_MPEG4_MIN_QP (integer)
Minimum quantization parameter for MPEG4. Valid range: from 1 to 31.

V4L2_CID_MPEG_VIDEO_MPEG4_MAX_QP (integer)
Maximum quantization parameter for MPEG4. Valid range: from 1 to 31.

V4L2_CID_MPEG_VIDEO_MPEG4_P_FRAME_QP (integer)
Quantization parameter for an P frame for MPEG4. Valid range: from 1 to 31.

V4L2_CID_MPEG_VIDEO_MPEG4_B_FRAME_QP (integer)
Quantization parameter for an B frame for MPEG4. Valid range: from 1 to 31.

V4L2_CID_MPEG_VIDEO_VBV_SIZE (integer)
The Video Buffer Verifier size in kilobytes, it is used as a limitation of frame skip. The VBV is defined in the standard as a mean to verify that the produced stream will be successfully decoded. The standard describes it as "Part of a hypothetical decoder that is conceptually connected to the output of the encoder. Its purpose is to provide a constraint on the variability of the data rate that an encoder or editing process may produce.". Applicable to the MPEG1, MPEG2, MPEG4 encoders.

V4L2_CID_MPEG_VIDEO_VBV_DELAY (integer)
Sets the initial delay in milliseconds for VBV buffer control.

V4L2_CID_MPEG_VIDEO_MV_H_SEARCH_RANGE (integer)
Horizontal search range defines maximum horizontal search area in pixels to search and match for the present Macroblock (MB) in the reference picture. This V4L2 control macro is used to set horizontal search range for motion estimation module in video encoder.

V4L2_CID_MPEG_VIDEO_MV_V_SEARCH_RANGE (integer)
Vertical search range defines maximum vertical search area in pixels to search and match for the present Macroblock (MB) in the reference picture. This V4L2 control macro is used to set vertical search range for motion estimation module in video encoder.

V4L2_CID_MPEG_VIDEO_FORCE_KEY_FRAME (button)
Force a key frame for the next queued buffer. Applicable to encoders. This is a general, codec-agnostic keyframe control.

V4L2_CID_MPEG_VIDEO_H264_CPB_SIZE (integer)
The Coded Picture Buffer size in kilobytes, it is used as a limitation of frame skip. The CPB is defined in the H264 standard as a mean to verify that the

produced stream will be successfully decoded. Applicable to the H264 encoder.

V4L2_CID_MPEG_VIDEO_H264_I_PERIOD (integer)
Period between I-frames in the open GOP for H264. In case of an open GOP this is the period between two I-frames. The period between IDR (Instantaneous Decoding Refresh) frames is taken from the GOP_SIZE control. An IDR frame, which stands for Instantaneous Decoding Refresh is an I-frame after which no prior frames are referenced. This means that a stream can be restarted from an IDR frame without the need to store or decode any previous frames. Applicable to the H264 encoder.

V4L2_CID_MPEG_VIDEO_HEADER_MODE
(enum)

enum v4l2_mpeg_video_header_mode -
Determines whether the header is returned as the first buffer or is it returned together with the first frame. Applicable to encoders. Possible values are:

V4L2_MPEG_VIDEO_HEADER_MODE_SEPARATE	The stream header is returned separately in the first buffer.
V4L2_MPEG_VIDEO_HEADER_MODE_JOINED_WITH_1ST_FRAME	The stream header is returned together with the first encoded frame.

V4L2_CID_MPEG_VIDEO_REPEAT_SEQ_HEADER (boolean)
Repeat the video sequence headers. Repeating these headers makes random access to the video stream easier. Applicable to the MPEG1, 2 and 4 encoder.

V4L2_CID_MPEG_VIDEO_DECODER_MPEG4_DEBLOCK_FILTER (boolean)
Enabled the deblocking post processing filter for MPEG4 decoder. Applicable to the MPEG4 decoder.

V4L2_CID_MPEG_VIDEO_MPEG4_VOP_TIME_RES (integer)
vop_time_increment_resolution value for MPEG4. Applicable to the MPEG4 encoder.

V4L2_CID_MPEG_VIDEO_MPEG4_VOP_TIME_INC (integer)
vop_time_increment value for MPEG4. Applicable to the MPEG4 encoder.

V4L2_CID_MPEG_VIDEO_H264_SEI_FRAME_PACKING (boolean)
Enable generation of frame packing supplemental enhancement information in the encoded bitstream. The frame packing SEI message contains the arrangement of L and R planes for 3D viewing. Applicable to the H264 encoder.

V4L2_CID_MPEG_VIDEO_H264_SEI_FP_CURRENT_FRAME_0 (boolean)
Sets current frame as frame0 in frame packing SEI. Applicable to the H264 encoder.

V4L2_CID_MPEG_VIDEO_H264_SEI_FP_ARRANGEMENT_TYPE
(enum)

enum v4l2_mpeg_video_h264_sei_fp_arrangement_type -
Frame packing arrangement type for H264 SEI. Applicable to the H264 encoder. Possible values are:

V4L2_MPEG_VIDEO_H264_SEI_FP_ARRANGEMENT_TYPE_CHEKERBOARD	Pixels are alternatively from L and R.
V4L2_MPEG_VIDEO_H264_SEI_FP_ARRANGEMENT_TYPE_COLUMN	L and R are interlaced by column.
V4L2_MPEG_VIDEO_H264_SEI_FP_ARRANGEMENT_TYPE_ROW	L and R are interlaced by row.
V4L2_MPEG_VIDEO_H264_SEI_FP_ARRANGEMENT_TYPE_SIDE_BY_SIDE	L is on the left, R on the right.
V4L2_MPEG_VIDEO_H264_SEI_FP_ARRANGEMENT_TYPE_TOP_BOTTOM	L is on top, R on bottom.
V4L2_MPEG_VIDEO_H264_SEI_FP_ARRANGEMENT_TYPE_TEMPORAL	One view per frame.

V4L2_CID_MPEG_VIDEO_H264_FMO (boolean)
Enables flexible macroblock ordering in the encoded bitstream. It is a technique used for restructuring the ordering of macroblocks in pictures. Applicable to the H264 encoder.

V4L2_CID_MPEG_VIDEO_H264_FMO_MAP_TYPE
(enum)

enum v4l2_mpeg_video_h264_fmo_map_type -
When using FMO, the map type divides the image in different scan patterns of macroblocks. Applicable to the H264 encoder. Possible values are:

V4L2_MPEG_VIDEO_H264_FMO_MAP_TYPE_INTERLEAVED_SLICES	Slices are interleaved one after other with macroblocks in run length order.
V4L2_MPEG_VIDEO_H264_FMO_MAP_TYPE_SCATTERED_SLICES	Scatters the macroblocks based on a mathematical function known to both encoder and decoder.
V4L2_MPEG_VIDEO_H264_FMO_MAP_TYPE_FOREGROUND_WITH_LEFT_OVER	Macroblocks arranged in rectangular areas or regions of interest.
V4L2_MPEG_VIDEO_H264_FMO_MAP_TYPE_BOX_OUT	Slice groups grow in a cyclic way from centre to outwards.
V4L2_MPEG_VIDEO_H264_FMO_MAP_TYPE_RASTER_SCAN	Slice groups grow in raster scan pattern from left to right.
V4L2_MPEG_VIDEO_H264_FMO_MAP_TYPE_WIPE_SCAN	Slice groups grow in wipe scan pattern from top to bottom.
V4L2_MPEG_VIDEO_H264_FMO_MAP_TYPE_EXPLICIT	User defined map type.

V4L2_CID_MPEG_VIDEO_H264_FMO_SLICE_GROUP (integer)
Number of slice groups in FMO. Applicable to the H264 encoder.

V4L2_CID_MPEG_VIDEO_H264_FMO_CHANGE_DIRECTION
(enum)

enum v4l2_mpeg_video_h264_fmo_change_dir -
Specifies a direction of the slice group change for raster and wipe maps. Applicable to the H264 encoder. Possible values are:

V4L2_MPEG_VIDEO_H264_FMO_CHANGE_DIR_RIGH	Raster scan or wipe right.
V4L2_MPEG_VIDEO_H264_FMO_CHANGE_DIR_LEFT	Reverse raster scan or wipe left.

V4L2_CID_MPEG_VIDEO_H264_FMO_CHANGE_RATE (integer)
Specifies the size of the first slice group for raster and wipe map. Applicable to the H264 encoder.

V4L2_CID_MPEG_VIDEO_H264_FMO_RUN_LENGTH (integer)
Specifies the number of consecutive macroblocks for the interleaved map. Applicable to the H264 encoder.

V4L2_CID_MPEG_VIDEO_H264_ASO (boolean)
Enables arbitrary slice ordering in encoded bitstream. Applicable to the H264 encoder.

V4L2_CID_MPEG_VIDEO_H264_ASO_SLICE_ORDER (integer)
Specifies the slice order in ASO. Applicable to the H264 encoder. The supplied 32-bit integer is interpreted as follows (bit 0 = least significant bit):

Bit 0:15	Slice ID
Bit 16:32	Slice position or order

V4L2_CID_MPEG_VIDEO_H264_HIERARCHICAL_CODING (boolean)
Enables H264 hierarchical coding. Applicable to the H264 encoder.

V4L2_CID_MPEG_VIDEO_H264_HIERARCHICAL_CODING_TYPE
(enum)

enum v4l2_mpeg_video_h264_hierarchical_coding_type -
Specifies the hierarchical coding type. Applicable to the H264 encoder. Possible values are:

V4L2_MPEG_VIDEO_H264_HIERARCHICAL_CODING_B	Hierarchical B coding.
V4L2_MPEG_VIDEO_H264_HIERARCHICAL_CODING_P	Hierarchical P coding.

V4L2_CID_MPEG_VIDEO_H264_HIERARCHICAL_CODING_LAYER (integer)
Specifies the number of hierarchical coding layers. Applicable to the H264 encoder.

V4L2_CID_MPEG_VIDEO_H264_HIERARCHICAL_CODING_LAYER_QP (integer)
Specifies a user defined QP for each layer. Applicable to the H264 encoder. The supplied 32-bit integer is interpreted as follows (bit 0 = least significant bit):

Bit 0:15	QP value
Bit 16:32	Layer number

V4L2_CID_MPEG_VIDEO_H264_SPS (struct)
Specifies the sequence parameter set (as extracted from the bitstream) for the associated H264 slice data. This includes the necessary parameters for

configuring a stateless hardware decoding pipeline for H264. The bitstream parameters are defined according to *ITU-T Rec. H.264 Specification (04/2017 Edition)*, section 7.4.2.1.1 "Sequence Parameter Set Data Semantics". For further documentation, refer to the above specification, unless there is an explicit comment stating otherwise.

Note: This compound control is not yet part of the public kernel API and it is expected to change.

---

type **v4l2_ctrl_h264_sps**

Table 1: struct v4l2_ctrl_h264_sps

__u8	profile_idc	
__u8	constraint_set_flags	See *Sequence Parameter Set Constrain*
__u8	level_idc	
__u8	seq_parameter_set_id	
__u8	chroma_format_idc	
__u8	bit_depth_luma_minus8	
__u8	bit_depth_chroma_minus8	
__u8	log2_max_frame_num_minus4	
__u8	pic_order_cnt_type	
__u8	log2_max_pic_order_cnt_lsb_minus4	
__u8	max_num_ref_frames	
__u8	num_ref_frames_in_pic_order_cnt_cycle	
__s32	offset_for_ref_frame[255]	
__s32	offset_for_non_ref_pic	
__s32	offset_for_top_to_bottom_field	
__u16	pic_width_in_mbs_minus1	
__u16	pic_height_in_map_units_minus1	
__u32	flags	See *Sequence Parameter Set Flags*

Sequence Parameter Set Constraints Set Flags

V4L2_H264_SPS_CONSTRAINT_SET0_FLAG	0x00000001
V4L2_H264_SPS_CONSTRAINT_SET1_FLAG	0x00000002
V4L2_H264_SPS_CONSTRAINT_SET2_FLAG	0x00000004
V4L2_H264_SPS_CONSTRAINT_SET3_FLAG	0x00000008
V4L2_H264_SPS_CONSTRAINT_SET4_FLAG	0x00000010
V4L2_H264_SPS_CONSTRAINT_SET5_FLAG	0x00000020

Sequence Parameter Set Flags

V4L2_H264_SPS_FLAG_SEPARATE_COLOUR_PLANE	0x00000001
V4L2_H264_SPS_FLAG_QPPRIME_Y_ZERO_TRANSFORM_BYPASS	0x00000002
V4L2_H264_SPS_FLAG_DELTA_PIC_ORDER_ALWAYS_ZERO	0x00000004
V4L2_H264_SPS_FLAG_GAPS_IN_FRAME_NUM_VALUE_ALLOWED	0x00000008

continues on next page

Table 3 – continued from previous page

V4L2_H264_SPS_FLAG_FRAME_MBS_ONLY	0x00000010
V4L2_H264_SPS_FLAG_MB_ADAPTIVE_FRAME_FIELD	0x00000020
V4L2_H264_SPS_FLAG_DIRECT_8X8_INFERENCE	0x00000040

V4L2_CID_MPEG_VIDEO_H264_PPS (struct)

Specifies the picture parameter set (as extracted from the bitstream) for the associated H264 slice data. This includes the necessary parameters for configuring a stateless hardware decoding pipeline for H264. The bitstream parameters are defined according to *ITU-T Rec. H.264 Specification (04/2017 Edition)*, section 7.4.2.2 "Picture Parameter Set RBSP Semantics". For further documentation, refer to the above specification, unless there is an explicit comment stating otherwise.

Note: This compound control is not yet part of the public kernel API and it is expected to change.

type **v4l2_ctrl_h264_pps**

Table 4: struct v4l2_ctrl_h264_pps

__u8	pic_parameter_set_id	
__u8	seq_parameter_set_id	
__u8	num_slice_groups_minus1	
__u8	num_ref_idx_l0_default_active_minus1	
__u8	num_ref_idx_l1_default_active_minus1	
__u8	weighted_bipred_idc	
__s8	pic_init_qp_minus26	
__s8	pic_init_qs_minus26	
__s8	chroma_qp_index_offset	
__s8	second_chroma_qp_index_offset	
__u16	flags	See *Picture Parameter Set Flags*

Picture Parameter Set Flags

V4L2_H264_PPS_FLAG_ENTROPY_CODING_MODE	0x00000001	
V4L2_H264_PPS_FLAG_BOTTOM_FIELD_PIC_ORDER_IN_FRAME_PRESENT	0x00000002	
V4L2_H264_PPS_FLAG_WEIGHTED_PRED	0x00000004	
V4L2_H264_PPS_FLAG_DEBLOCKING_FILTER_CONTROL_PRESENT	0x00000008	
V4L2_H264_PPS_FLAG_CONSTRAINED_INTRA_PRED	0x00000010	
V4L2_H264_PPS_FLAG_REDUNDANT_PIC_CNT_PRESENT	0x00000020	
V4L2_H264_PPS_FLAG_TRANSFORM_8X8_MODE	0x00000040	
V4L2_H264_PPS_FLAG_SCALING_MATRIX_PRESENT	0x00000080	Indicate

V4L2_CID_MPEG_VIDEO_H264_SCALING_MATRIX (struct)

Specifies the scaling matrix (as extracted from the bitstream) for the associated H264 slice data. The bitstream parameters are defined according to *ITU-T Rec. H.264 Specification (04/2017 Edition)*, section 7.4.2.1.1.1 "Scaling List Semantics". For further documentation, refer to the above specification, unless there is an explicit comment stating otherwise.

Note: This compound control is not yet part of the public kernel API and it is expected to change.

type **v4l2_ctrl_h264_scaling_matrix**

__u8	scaling_list_4x4[6][16]	Scaling matrix after applying the inverse scanning proces
__u8	scaling_list_8x8[6][64]	Scaling matrix after applying the inverse scanning proces

V4L2_CID_MPEG_VIDEO_H264_SLICE_PARAMS (struct)

Specifies the slice parameters (as extracted from the bitstream) for the associated H264 slice data. This includes the necessary parameters for configuring a stateless hardware decoding pipeline for H264. The bitstream parameters are defined according to *ITU-T Rec. H.264 Specification (04/2017 Edition)*, section 7.4.3 "Slice Header Semantics". For further documentation, refer to the above specification, unless there is an explicit comment stating otherwise.

Note: This compound control is not yet part of the public kernel API and it is expected to change.

type **v4l2_ctrl_h264_slice_params**

Table 7: struct v4l2_

__u32	header_bit_size	Offset in bits to slice_data
__u32	first_mb_in_slice	
__u8	slice_type	
__u8	colour_plane_id	
__u8	redundant_pic_cnt	
__u8	cabac_init_idc	
__s8	slice_qp_delta	
__s8	slice_qs_delta	
__u8	disable_deblocking_filter_idc	
__s8	slice_alpha_c0_offset_div2	
__s8	slice_beta_offset_div2	
__u8	num_ref_idx_l0_active_minus1	If num_ref_idx_active_ove
__u8	num_ref_idx_l1_active_minus1	If num_ref_idx_active_ove
__u8	reserved	Applications and drivers r
struct *v4l2_h264_reference*	ref_pic_list0[32]	Reference picture list afte
struct *v4l2_h264_reference*	ref_pic_list1[32]	Reference picture list afte
__u32	flags	See *Slice Parameter Flags*

Slice Parameter Set Flags

V4L2_H264_SLICE_FLAG_DIRECT_SPATIAL_MV_PRED	0x00000001
V4L2_H264_SLICE_FLAG_SP_FOR_SWITCH	0x00000002

V4L2_CID_MPEG_VIDEO_H264_PRED_WEIGHTS (struct)

Prediction weight table defined according to *ITU-T Rec. H.264 Specification (04/2017 Edition)*, section 7.4.3.2 "Prediction Weight Table Semantics". The prediction weight table must be passed by applications under the conditions explained in section 7.3.3 "Slice header syntax".

Note: This compound control is not yet part of the public kernel API and it is expected to change.

type **v4l2_ctrl_h264_pred_weights**

Table 9: struct v4l2_ct

__u16	luma_log2_weight_denom	
__u16	chroma_log2_weight_denom	
struct *v4l2_h264_weight_factors*	weight_factors[2]	The weight factors at inde

type **v4l2_h264_weight_factors**

Table 10: struct v4l2_h264_weight_factors

__s16	luma_weight[32]
__s16	luma_offset[32]
__s16	chroma_weight[32][2]
__s16	chroma_offset[32][2]

`Picture Reference`

type **v4l2_h264_reference**

Table 11: struct v4l2_h264_reference

__u8	fields	Specifies how the picture is referenced. See *Reference Fields*
__u8	index	Index into the *v4l2_ctrl_h264_decode_params*.dpb array.

`Reference Fields`

V4L2_H264_TOP_FIELD_REF	0x1	The top field in field pair is used for short-term referen
V4L2_H264_BOTTOM_FIELD_REF	0x2	The bottom field in field pair is used for short-term refe
V4L2_H264_FRAME_REF	0x3	The frame (or the top/bottom fields, if it's a field pair)

V4L2_CID_MPEG_VIDEO_H264_DECODE_PARAMS (struct)

Specifies the decode parameters (as extracted from the bitstream) for the associated H264 slice data. This includes the necessary parameters for configuring a stateless hardware decoding pipeline for H264. The bitstream parameters are defined according to *ITU-T Rec. H.264 Specification (04/2017*

Edition). For further documentation, refer to the above specification, unless there is an explicit comment stating otherwise.

Note: This compound control is not yet part of the public kernel API and it is expected to change.

type **v4l2_ctrl_h264_decode_params**

struct *v4l2_h264_dpb_entry*	dpb[16]	
__u16	nal_ref_idc	NAL reference ID value co
__u16	frame_num	
__s32	top_field_order_cnt	Picture Order Count for th
__s32	bottom_field_order_cnt	Picture Order Count for th
__u16	idr_pic_id	
__u16	pic_order_cnt_lsb	
__s32	delta_pic_order_cnt_bottom	
__s32	delta_pic_order_cnt0	
__s32	delta_pic_order_cnt1	
__u32	dec_ref_pic_marking_bit_size	Size in bits of the dec_ref_p
__u32	pic_order_cnt_bit_size	Combined size in bits of th
__u32	slice_group_change_cycle	
__u32	reserved	Applications and drivers m
__u32	flags	See *Decode Parameters Fl*

Decode Parameters Flags

V4L2_H264_DECODE_PARAM_FLAG_IDR_PIC	0x00000001	That picture is an IDR picture
V4L2_H264_DECODE_PARAM_FLAG_FIELD_PIC	0x00000002	
V4L2_H264_DECODE_PARAM_FLAG_BOTTOM_FIELD	0x00000004	

type **v4l2_h264_dpb_entry**

__u64	reference_ts	Timestamp of the V4L2 capture buffer to use as referenc
__u32	pic_num	
__u16	frame_num	
__u8	fields	Specifies how the DPB entry is referenced. See *Referenc*
__u8	reserved[5]	Applications and drivers must set this to zero.
__s32	top_field_order_cnt	
__s32	bottom_field_order_cnt	
__u32	flags	See *DPB Entry Flags*

DPB Entries Flags

V4L2_H264_DPB_ENTRY_FLAG_VALID	0x00000001	The DPB entry is valid (non-empty) a
V4L2_H264_DPB_ENTRY_FLAG_ACTIVE	0x00000002	The DPB entry is used for reference.
V4L2_H264_DPB_ENTRY_FLAG_LONG_TERM	0x00000004	The DPB entry is used for long-term
V4L2_H264_DPB_ENTRY_FLAG_FIELD	0x00000008	The DPB entry is a single field or a c

V4L2_CID_MPEG_VIDEO_H264_DECODE_MODE (enum)

Specifies the decoding mode to use. Currently exposes slice-based and frame-based decoding but new modes might be added later on. This control is used as a modifier for V4L2_PIX_FMT_H264_SLICE pixel format. Applications that support V4L2_PIX_FMT_H264_SLICE are required to set this control in order to specify the decoding mode that is expected for the buffer. Drivers may expose a single or multiple decoding modes, depending on what they can support.

Note: This menu control is not yet part of the public kernel API and it is expected to change.

type **v4l2_mpeg_video_h264_decode_mode**

V4L2_MPEG_VIDEO_H264_DECODE_MODE_SLICE_BASED	0	Decoding is done at the slice granul
V4L2_MPEG_VIDEO_H264_DECODE_MODE_FRAME_BASED	1	Decoding is done at the frame gran

V4L2_CID_MPEG_VIDEO_H264_START_CODE (enum)

Specifies the H264 slice start code expected for each slice. This control is used as a modifier for V4L2_PIX_FMT_H264_SLICE pixel format. Applications that support V4L2_PIX_FMT_H264_SLICE are required to set this control in order to specify the start code that is expected for the buffer. Drivers may expose a single or multiple start codes, depending on what they can support.

Note: This menu control is not yet part of the public kernel API and it is expected to change.

type **v4l2_mpeg_video_h264_start_code**

V4L2_MPEG_VIDEO_H264_START_CODE_NONE	0	Selecting this value specifies that H264 sl
V4L2_MPEG_VIDEO_H264_START_CODE_ANNEX_B	1	Selecting this value specifies that H264 sl

V4L2_CID_MPEG_VIDEO_MPEG2_SLICE_PARAMS (struct)

Specifies the slice parameters (as extracted from the bitstream) for the associated MPEG-2 slice data. This includes the necessary parameters for configuring a stateless hardware decoding pipeline for MPEG-2. The bitstream parameters are defined according to *ISO 13818-2*.

Note: This compound control is not yet part of the public kernel API and it is expected to change.

type **v4l2_ctrl_mpeg2_slice_params**

Table 19: struct v4l2_ctrl_mpeg2_slice_params

__u32	bit_size	Size (in bits) of the current slice data.
__u32	data_bit_offset	Offset (in bits) to the video data in the current slice data.
struct *v4l2_mpeg2_sequence*	sequence	Structure with MPEG-2 sequence metadata, merging relevant fields from the sequence header and sequence extension parts of the bitstream.
struct *v4l2_mpeg2_picture*	picture	Structure with MPEG-2 picture metadata, merging relevant fields from the picture header and picture coding extension parts of the bitstream.
__u64	backward_ref_ts	Timestamp of the V4L2 capture buffer to use as backward reference, used with B-coded and P-coded frames. The timestamp refers to the timestamp field in struct v4l2_buffer. Use the v4l2_timeval_to_ns() function to convert the struct timeval in struct v4l2_buffer to a __u64.
__u64	forward_ref_ts	Timestamp for the V4L2 capture buffer to use as forward reference, used with B-coded frames. The timestamp refers to the timestamp field in struct v4l2_buffer. Use the v4l2_timeval_to_ns() function to convert the struct timeval in struct v4l2_buffer to a __u64.
__u32	quantiser_scale_code	Code used to determine the quantization scale to use for the IDCT.

type **v4l2_mpeg2_sequence**

Table 20: struct v4l2_mpeg2_sequence

__u16	horizontal_size	The width of the displayable part of the frame's luminance component.
__u16	vertical_size	The height of the displayable part of the frame's luminance component.
__u32	vbv_buffer_size	Used to calculate the required size of the video buffering verifier, defined (in bits) as: 16 * 1024 * vbv_buffer_size.
__u16	profile_and_level_indication	The current profile and level indication as extracted from the bitstream.
__u8	progressive_sequence	Indication that all the frames for the sequence are progressive instead of interlaced.
__u8	chroma_format	The chrominance sub-sampling format (1: 4:2:0, 2: 4:2:2, 3: 4:4:4).

type **v4l2_mpeg2_picture**

Table 21: struct v4l2_mpeg2_picture

__u8	picture_coding_type	Picture coding type for the frame covered by the current slice (V4L2_MPEG2_PICTURE_CODING_TYPE_I, V4L2_MPEG2_PICTURE_CODING_TYPE_P or V4L2_MPEG2_PICTURE_CODING_TYPE_B).
__u8	f_code[2][2]	Motion vector codes.
__u8	intra_dc_precision	Precision of Discrete Cosine transform (0: 8 bits precision, 1: 9 bits precision, 2: 10 bits precision, 3: 11 bits precision).
__u8	picture_structure	Picture structure (1: interlaced top field, 2: interlaced bottom field, 3: progressive frame).
__u8	top_field_first	If set to 1 and interlaced stream, top field is output first.
__u8	frame_pred_frame_dct	If set to 1, only frame-DCT and frame prediction are used.
__u8	concealment_motion_vectors	If set to 1, motion vectors are coded for intra macroblocks.
__u8	q_scale_type	This flag affects the inverse quantization process.
__u8	intra_vlc_format	This flag affects the decoding of transform coefficient data.
__u8	alternate_scan	This flag affects the decoding of transform coefficient data.
__u8	repeat_first_field	This flag affects the decoding process of progressive frames.
__u16	progressive_frame	Indicates whether the current frame is progressive.

V4L2_CID_MPEG_VIDEO_MPEG2_QUANTIZATION (struct)

Specifies quantization matrices (as extracted from the bitstream) for the associated MPEG-2 slice data.

Note: This compound control is not yet part of the public kernel API and it is expected to change.

type **v4l2_ctrl_mpeg2_quantization**

Table 22: struct v4l2_ctrl_mpeg2_quantization

__u8	load_intra_quantiser_matrix	One bit to indicate whether to load the intra_quantiser_matrix data.
__u8	load_non_intra_quantiser_matrix	One bit to indicate whether to load the non_intra_quantiser_matrix data.
__u8	load_chroma_intra_quantiser_matrix	One bit to indicate whether to load the chroma_intra_quantiser_matrix data, only relevant for non-4:2:0 YUV formats.
__u8	load_chroma_non_intra_quantiser_matrix	One bit to indicate whether to load the chroma_non_intra_quantiser_matrix data, only relevant for non-4:2:0 YUV formats.
__u8	intra_quantiser_matrix[64]	The quantization matrix coefficients for intra-coded frames, in zigzag scanning order. It is relevant for both luma and chroma components, although it can be superseded by the chroma-specific matrix for non-4:2:0 YUV formats.
__u8	non_intra_quantiser_matrix[64]	The quantization matrix coefficients for non-intra-coded frames, in zigzag scanning order. It is relevant for both luma and chroma components, although it can be superseded by the chroma-specific matrix for non-4:2:0 YUV formats.
__u8	chroma_intra_quantiser_matrix[64]	The quantization matrix coefficients for the chominance component of intra-coded frames, in zigzag scanning order. Only relevant for non-4:2:0 YUV formats.
__u8	chroma_non_intra_quantiser_matrix[64]	The quantization matrix coefficients for the chrominance component of non-intra-coded frames, in zigzag scanning order. Only relevant for non-4:2:0 YUV formats.

V4L2_CID_FWHT_I_FRAME_QP (integer)
Quantization parameter for an I frame for FWHT. Valid range: from 1 to 31.

V4L2_CID_FWHT_P_FRAME_QP (integer)
Quantization parameter for a P frame for FWHT. Valid range: from 1 to 31.

V4L2_CID_MPEG_VIDEO_VP8_FRAME_HEADER (struct)
Specifies the frame parameters for the associated VP8 parsed frame data. This includes the necessary parameters for configuring a stateless hardware decoding pipeline for VP8. The bitstream parameters are defined according to *VP8*.

Note: This compound control is not yet part of the public kernel API and it is expected to change.

type **v4l2_ctrl_vp8_frame_header**

Table 23: struct v4l2_ctrl_vp8_frame_header

struct *v4l2_vp8_segment_header*	segment_header	Structure with segment-based adjustments metadata.
struct *v4l2_vp8_loopfilter_hea*	loopfilter_header	Structure with loop filter level adjustments metadata.
struct *v4l2_vp8_quantization_h*	quant_header	Structure with VP8 dequantization indices metadata.
struct *v4l2_vp8_entropy_header*	entropy_header	Structure with VP8 entropy coder probabilities metadata.
struct *v4l2_vp8_entropy_coder_*	coder_state	Structure with VP8 entropy coder state.
__u16	width	The width of the frame. Must be set for all frames.
__u16	height	The height of the frame. Must be set for all frames.
__u8	horizontal_scale	Horizontal scaling factor.
__u8	vertical_scaling factor	Vertical scale.
__u8	version	Bitstream version.
__u8	prob_skip_false	Indicates the probability that the macroblock is not skipped.
__u8	prob_intra	Indicates the probability that a macroblock is intra-predicted.
__u8	prob_last	Indicates the probability that the last reference frame is used for inter-prediction
__u8	prob_gf	Indicates the probability that the golden reference frame is used for inter-prediction
__u8	num_dct_parts	Number of DCT coefficients partitions. Must be one of: 1, 2, 4, or 8.
__u32	first_part_size	Size of the first partition, i.e. the control partition.
__u32	first_part_header_bits	Size in bits of the first partition header portion.
__u32	dct_part_sizes[8]	DCT coefficients sizes.
__u64	last_frame_ts	Timestamp for the V4L2 capture buffer to use as last reference frame, used with inter-coded frames. The timestamp refers to the `timestamp` field in struct `v4l2_buffer`. Use the `v4l2_timeval_to_ns()` function to convert the struct `timeval` in struct `v4l2_buffer` to a __u64.
__u64	golden_frame_ts	Timestamp for the V4L2 capture buffer to use as last reference frame, used with inter-coded frames. The timestamp refers to the `timestamp` field in struct `v4l2_buffer`. Use the `v4l2_timeval_to_ns()` function to convert the struct `timeval` in struct `v4l2_buffer` to a __u64.
__u64	alt_frame_ts	Timestamp for the V4L2 capture buffer to use as alternate reference frame, used with inter-coded frames. The timestamp refers to the `timestamp` field in struct `v4l2_buffer`. Use the `v4l2_timeval_to_ns()` function to convert the struct `timeval` in struct `v4l2_buffer` to a __u64.
__u64	flags	See *Frame Header Flags*

Frame Header Flags

V4L2_VP8_FRAME_HEADER_FLAG_KEY_FRAME	0x01	Indicates if the frame is a key frame.
V4L2_VP8_FRAME_HEADER_FLAG_EXPERIMENTAL	0x02	Experimental bitstream.
V4L2_VP8_FRAME_HEADER_FLAG_SHOW_FRAME	0x04	Show frame flag, indicates if the frame is
V4L2_VP8_FRAME_HEADER_FLAG_MB_NO_SKIP_COEFF	0x08	Enable/disable skipping of macroblocks w
V4L2_VP8_FRAME_HEADER_FLAG_SIGN_BIAS_GOLDEN	0x10	Sign of motion vectors when the golden fr
V4L2_VP8_FRAME_HEADER_FLAG_SIGN_BIAS_ALT	0x20	Sign of motion vectors when the alt frame

type **v4l2_vp8_entropy_coder_state**

Table 25: struct v4l2_vp8_entropy_coder_state

__u8	range	
__u8	value	
__u8	bit_count	
__u8	padding	Applications and drivers must set this to zero.

type **v4l2_vp8_segment_header**

Table 26: struct v4l2_vp8_segment_header

__s8	quant_update[4]	Signed quantizer value update.
__s8	lf_update[4]	Signed loop filter level value update.
__u8	segment_probs[3]	Segment probabilities.
__u8	padding	Applications and drivers must set this to zero.
__u32	flags	See *Segment Header Flags*

Segment Header Flags

V4L2_VP8_SEGMENT_HEADER_FLAG_ENABLED	0x01	Enable/disable segment-based adju
V4L2_VP8_SEGMENT_HEADER_FLAG_UPDATE_MAP	0x02	Indicates if the macroblock segmen
V4L2_VP8_SEGMENT_HEADER_FLAG_UPDATE_FEATURE_DATA	0x04	Indicates if the segment feature dat
V4L2_VP8_SEGMENT_HEADER_FLAG_DELTA_VALUE_MODE	0x08	If is set, the segment feature data m

type **v4l2_vp8_loopfilter_header**

Table 28: struct v4l2_vp8_loopfilter_header

__s8	ref_frm_delta[4]	Reference adjustment (signed) delta value.
__s8	mb_mode_delta[4]	Macroblock prediction mode adjustment (signed) delta value.
__u8	sharpness_level	Sharpness level
__u8	level	Filter level
__u16	padding	Applications and drivers must set this to zero.
__u32	flags	See *Loopfilter Header Flags*

Loopfilter Header Flags

V4L2_VP8_LF_HEADER_ADJ_ENABLE	0x01	Enable/disable macroblock-level loop filter adjustment.
V4L2_VP8_LF_HEADER_DELTA_UPDATE	0x02	Indicates if the delta values used in an adjustment are u

con

Table 29 – continued from previous page

V4L2_VP8_LF_FILTER_TYPE_SIMPLE	0x04	If set, indicates the filter type is simple. If cleared, the f

type **v4l2_vp8_quantization_header**

Table 30: struct v4l2_vp8_quantization_header

__u8	y_ac_qi	Luma AC coefficient table index.
__s8	y_dc_delta	Luma DC delta vaue.
__s8	y2_dc_delta	Y2 block DC delta value.
__s8	y2_ac_delta	Y2 block AC delta value.
__s8	uv_dc_delta	Chroma DC delta value.
__s8	uv_ac_delta	Chroma AC delta value.
__u16	padding	Applications and drivers must set this to zero.

type **v4l2_vp8_entropy_header**

Table 31: struct v4l2_vp8_entropy_header

__u8	coeff_probs[4][8][3][11]	Coefficient update probabilities.
__u8	y_mode_probs[4]	Luma mode update probabilities.
__u8	uv_mode_probs[3]	Chroma mode update probabilities.
__u8	mv_probs[2][19]	MV decoding update probabilities.
__u8	padding[3]	Applications and drivers must set this to zero.

MFC 5.1 MPEG Controls

The following MPEG class controls deal with MPEG decoding and encoding settings that are specific to the Multi Format Codec 5.1 device present in the S5P family of SoCs by Samsung.

MFC 5.1 Control IDs

V4L2_CID_MPEG_MFC51_VIDEO_DECODER_H264_DISPLAY_DELAY_ENABLE (boolean)
: If the display delay is enabled then the decoder is forced to return a CAPTURE buffer (decoded frame) after processing a certain number of OUTPUT buffers. The delay can be set through V4L2_CID_MPEG_MFC51_VIDEO_DECODER_H264_DISPLAY_DELAY. This feature can be used for example for generating thumbnails of videos. Applicable to the H264 decoder.

V4L2_CID_MPEG_MFC51_VIDEO_DECODER_H264_DISPLAY_DELAY (integer)
: Display delay value for H264 decoder. The decoder is forced to return a decoded frame after the set 'display delay' number of frames. If this number is low it may result in frames returned out of display order, in addition the hardware may still be using the returned buffer as a reference picture for subsequent frames.

V4L2_CID_MPEG_MFC51_VIDEO_H264_NUM_REF_PIC_FOR_P (integer)
: The number of reference pictures used for encoding a P picture. Applicable to the H264 encoder.

V4L2_CID_MPEG_MFC51_VIDEO_PADDING (boolean)
Padding enable in the encoder - use a color instead of repeating border pixels. Applicable to encoders.

V4L2_CID_MPEG_MFC51_VIDEO_PADDING_YUV (integer)
Padding color in the encoder. Applicable to encoders. The supplied 32-bit integer is interpreted as follows (bit 0 = least significant bit):

Bit 0:7	V chrominance information
Bit 8:15	U chrominance information
Bit 16:23	Y luminance information
Bit 24:31	Must be zero.

V4L2_CID_MPEG_MFC51_VIDEO_RC_REACTION_COEFF (integer)
Reaction coefficient for MFC rate control. Applicable to encoders.

Note:

1. Valid only when the frame level RC is enabled.
2. For tight CBR, this field must be small (ex. 2 ~ 10). For VBR, this field must be large (ex. 100 ~ 1000).
3. It is not recommended to use the greater number than FRAME_RATE * (10^9 / BIT_RATE).

V4L2_CID_MPEG_MFC51_VIDEO_H264_ADAPTIVE_RC_DARK (boolean)
Adaptive rate control for dark region. Valid only when H.264 and macroblock level RC is enabled (`V4L2_CID_MPEG_VIDEO_MB_RC_ENABLE`). Applicable to the H264 encoder.

V4L2_CID_MPEG_MFC51_VIDEO_H264_ADAPTIVE_RC_SMOOTH (boolean)
Adaptive rate control for smooth region. Valid only when H.264 and macroblock level RC is enabled (`V4L2_CID_MPEG_VIDEO_MB_RC_ENABLE`). Applicable to the H264 encoder.

V4L2_CID_MPEG_MFC51_VIDEO_H264_ADAPTIVE_RC_STATIC (boolean)
Adaptive rate control for static region. Valid only when H.264 and macroblock level RC is enabled (`V4L2_CID_MPEG_VIDEO_MB_RC_ENABLE`). Applicable to the H264 encoder.

V4L2_CID_MPEG_MFC51_VIDEO_H264_ADAPTIVE_RC_ACTIVITY (boolean)
Adaptive rate control for activity region. Valid only when H.264 and macroblock level RC is enabled (`V4L2_CID_MPEG_VIDEO_MB_RC_ENABLE`). Applicable to the H264 encoder.

V4L2_CID_MPEG_MFC51_VIDEO_FRAME_SKIP_MODE
(enum)

Note: This control is deprecated. Use the standard `V4L2_CID_MPEG_VIDEO_FRAME_SKIP_MODE` control instead.

enum v4l2_mpeg_mfc51_video_frame_skip_mode -

Indicates in what conditions the encoder should skip frames. If encoding a frame would cause the encoded stream to be larger then a chosen data limit then the frame will be skipped. Possible values are:

`V4L2_MPEG_MFC51_FRAME_SKIP_MODE_DISABLED`	Frame skip mode is disabled.
`V4L2_MPEG_MFC51_FRAME_SKIP_MODE_LEVEL_LIMIT`	Frame skip mode enabled and buffer limit is set by the chosen level and is defined by the standard.
`V4L2_MPEG_MFC51_FRAME_SKIP_MODE_BUF_LIMIT`	Frame skip mode enabled and buffer limit is set by the VBV (MPEG1/2/4) or CPB (H264) buffer size control.

`V4L2_CID_MPEG_MFC51_VIDEO_RC_FIXED_TARGET_BIT (integer)`

Enable rate-control with fixed target bit. If this setting is enabled, then the rate control logic of the encoder will calculate the average bitrate for a GOP and keep it below or equal the set bitrate target. Otherwise the rate control logic calculates the overall average bitrate for the stream and keeps it below or equal to the set bitrate. In the first case the average bitrate for the whole stream will be smaller then the set bitrate. This is caused because the average is calculated for smaller number of frames, on the other hand enabling this setting will ensure that the stream will meet tight bandwidth constraints. Applicable to encoders.

`V4L2_CID_MPEG_MFC51_VIDEO_FORCE_FRAME_TYPE`

(enum)

enum v4l2_mpeg_mfc51_video_force_frame_type -

Force a frame type for the next queued buffer. Applicable to encoders. Possible values are:

`V4L2_MPEG_MFC51_FORCE_FRAME_TYPE_DISABLED`	Forcing a specific frame type disabled.
`V4L2_MPEG_MFC51_FORCE_FRAME_TYPE_I_FRAME`	Force an I-frame.
`V4L2_MPEG_MFC51_FORCE_FRAME_TYPE_NOT_CODED`	Force a non-coded frame.

`V4L2_CID_MPEG_VIDEO_FWHT_PARAMS (struct)`

Specifies the fwht parameters (as extracted from the bitstream) for the associated FWHT data. This includes the necessary parameters for configuring a stateless hardware decoding pipeline for FWHT.

Note: This compound control is not yet part of the public kernel API and it is expected to change.

type **`v4l2_ctrl_fwht_params`**

Table 32: struct v4l2_ctrl_fwht_params

__u64	backward_ref_ts	Timestamp of the V4L2 capture buffer to use as backward reference, used with P-coded frames. The timestamp refers to the timestamp field in struct v4l2_buffer. Use the v4l2_timeval_to_ns() function to convert the struct timeval in struct v4l2_buffer to a __u64.
__u32	version	The version of the codec
__u32	width	The width of the frame
__u32	height	The height of the frame
__u32	flags	The flags of the frame, see *FWHT Flags*.
__u32	colorspace	The colorspace of the frame, from enum *v4l2_colorspace*.
__u32	xfer_func	The transfer function, from enum *v4l2_xfer_func*.
__u32	ycbcr_enc	The Y'CbCr encoding, from enum *v4l2_ycbcr_encoding*.
__u32	quantization	The quantization range, from enum *v4l2_quantization*.

FWHT Flags

FWHT_FL_IS_INTERLACED	0x00000001	Set if this is an interlaced format
FWHT_FL_IS_BOTTOM_FIRST	0x00000002	Set if this is a bottom-first (NTSC) interlaced format
FWHT_FL_IS_ALTERNATE	0x00000004	Set if each 'frame' contains just one field
FWHT_FL_IS_BOTTOM_FIELD	0x00000008	If FWHT_FL_IS_ALTERNATE was set, then this is set if this 'frame' is the bottom field, else it is the top field.
FWHT_FL_LUMA_IS_UNCOMPRESSED	0x00000010	Set if the luma plane is uncompressed
FWHT_FL_CB_IS_UNCOMPRESSED	0x00000020	Set if the cb plane is uncompressed
FWHT_FL_CR_IS_UNCOMPRESSED	0x00000040	Set if the cr plane is uncompressed
FWHT_FL_CHROMA_FULL_HEIGHT	0x00000080	Set if the chroma plane has the same height as the luma plane, else the chroma plane is half the height of the luma plane
FWHT_FL_CHROMA_FULL_WIDTH	0x00000100	Set if the chroma plane has the same width as the luma plane, else the chroma plane is half the width of the luma plane
FWHT_FL_ALPHA_IS_UNCOMPRESSED	0x00000200	Set if the alpha plane is uncompressed
FWHT_FL_I_FRAME	0x00000400	Set if this is an I-frame
FWHT_FL_COMPONENTS_NUM_MSK	0x00070000	A 4-values flag - the number of components - 1
FWHT_FL_PIXENC_YUV	0x00080000	Set if the pixel encoding is YUV
FWHT_FL_PIXENC_RGB	0x00100000	Set if the pixel encoding is RGB
FWHT_FL_PIXENC_HSV	0x00180000	Set if the pixel encoding is HSV

CX2341x MPEG Controls

The following MPEG class controls deal with MPEG encoding settings that are specific to the Conexant CX23415 and CX23416 MPEG encoding chips.

CX2341x Control IDs

V4L2_CID_MPEG_CX2341X_VIDEO_SPATIAL_FILTER_MODE
(enum)

enum v4l2_mpeg_cx2341x_video_spatial_filter_mode -
Sets the Spatial Filter mode (default MANUAL). Possible values are:

V4L2_MPEG_CX2341X_VIDEO_SPATIAL_FILTER_MODE_	Choose the filter manually
V4L2_MPEG_CX2341X_VIDEO_SPATIAL_FILTER_MODE_	Choose the filter automatically

V4L2_CID_MPEG_CX2341X_VIDEO_SPATIAL_FILTER (integer (0-15))
The setting for the Spatial Filter. 0 = off, 15 = maximum. (Default is 0.)

V4L2_CID_MPEG_CX2341X_VIDEO_LUMA_SPATIAL_FILTER_TYPE
(enum)

enum v4l2_mpeg_cx2341x_video_luma_spatial_filter_type -
Select the algorithm to use for the Luma Spatial Filter (default 1D_HOR). Possible values:

V4L2_MPEG_CX2341X_VIDEO_LUMA_SPATIAL_FILTER_TYPE_OFF	No filter
V4L2_MPEG_CX2341X_VIDEO_LUMA_SPATIAL_FILTER_TYPE_1D_HOR	One-dimensional horizontal
V4L2_MPEG_CX2341X_VIDEO_LUMA_SPATIAL_FILTER_TYPE_1D_VERT	One-dimensional vertical
V4L2_MPEG_CX2341X_VIDEO_LUMA_SPATIAL_FILTER_TYPE_2D_HV_SEPARABLE	Two-dimensional separable
V4L2_MPEG_CX2341X_VIDEO_LUMA_SPATIAL_FILTER_TYPE_2D_SYM_NON_SEPARABLE	Two-dimensional symmetrical non-separable

V4L2_CID_MPEG_CX2341X_VIDEO_CHROMA_SPATIAL_FILTER_TYPE
(enum)

enum v4l2_mpeg_cx2341x_video_chroma_spatial_filter_type -
Select the algorithm for the Chroma Spatial Filter (default 1D_HOR). Possible values are:

V4L2_MPEG_CX2341X_VIDEO_CHROMA_SPATIAL_FILTER_TYPE_OFF	No filter
V4L2_MPEG_CX2341X_VIDEO_CHROMA_SPATIAL_FILTER_TYPE_1D_HOR	One-dimensional horizontal

V4L2_CID_MPEG_CX2341X_VIDEO_TEMPORAL_FILTER_MODE
(enum)

enum v4l2_mpeg_cx2341x_video_temporal_filter_mode -
Sets the Temporal Filter mode (default MANUAL). Possible values are:

V4L2_MPEG_CX2341X_VIDEO_TEMPORAL_FILTER_MODE	Choose the filter manually
V4L2_MPEG_CX2341X_VIDEO_TEMPORAL_FILTER_MODE	Choose the filter automatically

V4L2_CID_MPEG_CX2341X_VIDEO_TEMPORAL_FILTER (integer (0-31))
The setting for the Temporal Filter. 0 = off, 31 = maximum. (Default is 8 for full-scale capturing and 0 for scaled capturing.)

V4L2_CID_MPEG_CX2341X_VIDEO_MEDIAN_FILTER_TYPE
(enum)

enum v4l2_mpeg_cx2341x_video_median_filter_type -
Median Filter Type (default OFF). Possible values are:

V4L2_MPEG_CX2341X_VIDEO_MEDIAN_FILTER_TYPE_OF	No filter
V4L2_MPEG_CX2341X_VIDEO_MEDIAN_FILTER_TYPE_HO	Horizontal filter
V4L2_MPEG_CX2341X_VIDEO_MEDIAN_FILTER_TYPE_VE	Vertical filter
V4L2_MPEG_CX2341X_VIDEO_MEDIAN_FILTER_TYPE_HO	Horizontal and vertical filter
V4L2_MPEG_CX2341X_VIDEO_MEDIAN_FILTER_TYPE_DI	Diagonal filter

V4L2_CID_MPEG_CX2341X_VIDEO_LUMA_MEDIAN_FILTER_BOTTOM (integer (0-255))
Threshold above which the luminance median filter is enabled (default 0)

V4L2_CID_MPEG_CX2341X_VIDEO_LUMA_MEDIAN_FILTER_TOP (integer (0-255))
Threshold below which the luminance median filter is enabled (default 255)

V4L2_CID_MPEG_CX2341X_VIDEO_CHROMA_MEDIAN_FILTER_BOTTOM (integer (0-255))
Threshold above which the chroma median filter is enabled (default 0)

V4L2_CID_MPEG_CX2341X_VIDEO_CHROMA_MEDIAN_FILTER_TOP (integer (0-255))
Threshold below which the chroma median filter is enabled (default 255)

V4L2_CID_MPEG_CX2341X_STREAM_INSERT_NAV_PACKETS (boolean)
The CX2341X MPEG encoder can insert one empty MPEG-2 PES packet into the stream between every four video frames. The packet size is 2048 bytes, including the packet_start_code_prefix and stream_id fields. The stream_id is 0xBF (private stream 2). The payload consists of 0x00 bytes, to be filled in by the application. 0 = do not insert, 1 = insert packets.

VPX Control Reference

The VPX controls include controls for encoding parameters of VPx video codec.

VPX Control IDs

V4L2_CID_MPEG_VIDEO_VPX_NUM_PARTITIONS
(enum)

enum v4l2_vp8_num_partitions -
The number of token partitions to use in VP8 encoder. Possible values are:

V4L2_CID_MPEG_VIDEO_VPX_1_PARTITION	1 coefficient partition
V4L2_CID_MPEG_VIDEO_VPX_2_PARTITIONS	2 coefficient partitions
V4L2_CID_MPEG_VIDEO_VPX_4_PARTITIONS	4 coefficient partitions
V4L2_CID_MPEG_VIDEO_VPX_8_PARTITIONS	8 coefficient partitions

V4L2_CID_MPEG_VIDEO_VPX_IMD_DISABLE_4X4 (boolean)
Setting this prevents intra 4x4 mode in the intra mode decision.

V4L2_CID_MPEG_VIDEO_VPX_NUM_REF_FRAMES
(enum)

enum v4l2_vp8_num_ref_frames -
The number of reference pictures for encoding P frames. Possible values are:

V4L2_CID_MPEG_VIDEO_VPX_1_REF_FRAME	Last encoded frame will be searched
V4L2_CID_MPEG_VIDEO_VPX_2_REF_FRAME	Two frames will be searched among the last encoded frame, the golden frame and the alternate reference (altref) frame. The encoder implementation will decide which two are chosen.
V4L2_CID_MPEG_VIDEO_VPX_3_REF_FRAME	The last encoded frame, the golden frame and the altref frame will be searched.

V4L2_CID_MPEG_VIDEO_VPX_FILTER_LEVEL (integer)
Indicates the loop filter level. The adjustment of the loop filter level is done via a delta value against a baseline loop filter value.

V4L2_CID_MPEG_VIDEO_VPX_FILTER_SHARPNESS (integer)
This parameter affects the loop filter. Anything above zero weakens the deblocking effect on the loop filter.

V4L2_CID_MPEG_VIDEO_VPX_GOLDEN_FRAME_REF_PERIOD (integer)
Sets the refresh period for the golden frame. The period is defined in number of frames. For a value of 'n', every nth frame starting from the first key frame will be taken as a golden frame. For eg. for encoding sequence of 0, 1, 2, 3, 4, 5, 6, 7 where the golden frame refresh period is set as 4, the frames 0, 4, 8 etc will be taken as the golden frames as frame 0 is always a key frame.

V4L2_CID_MPEG_VIDEO_VPX_GOLDEN_FRAME_SEL
(enum)

enum v4l2_vp8_golden_frame_sel -

Selects the golden frame for encoding. Possible values are:

V4L2_CID_MPEG_VIDEO_VPX_GOLDEN_FRAME_USE_PREV	Use the (n-2)th frame as a golden frame, current frame index being 'n'.
V4L2_CID_MPEG_VIDEO_VPX_GOLDEN_FRAME_USE_REF_PERIOD	Use the previous specific frame indicated by V4L2_CID_MPEG_VIDEO_VPX_GOLDEN_FRAME_REF_PERIOD as a golden frame.

V4L2_CID_MPEG_VIDEO_VPX_MIN_QP (integer)

Minimum quantization parameter for VP8.

V4L2_CID_MPEG_VIDEO_VPX_MAX_QP (integer)

Maximum quantization parameter for VP8.

V4L2_CID_MPEG_VIDEO_VPX_I_FRAME_QP (integer)

Quantization parameter for an I frame for VP8.

V4L2_CID_MPEG_VIDEO_VPX_P_FRAME_QP (integer)

Quantization parameter for a P frame for VP8.

V4L2_CID_MPEG_VIDEO_VP8_PROFILE

(enum)

enum v4l2_mpeg_video_vp8_profile -

This control allows selecting the profile for VP8 encoder. This is also used to enumerate supported profiles by VP8 encoder or decoder. Possible values are:

V4L2_MPEG_VIDEO_VP8_PROFILE_0	Profile 0
V4L2_MPEG_VIDEO_VP8_PROFILE_1	Profile 1
V4L2_MPEG_VIDEO_VP8_PROFILE_2	Profile 2
V4L2_MPEG_VIDEO_VP8_PROFILE_3	Profile 3

V4L2_CID_MPEG_VIDEO_VP9_PROFILE

(enum)

enum v4l2_mpeg_video_vp9_profile -

This control allows selecting the profile for VP9 encoder. This is also used to enumerate supported profiles by VP9 encoder or decoder. Possible values are:

V4L2_MPEG_VIDEO_VP9_PROFILE_0	Profile 0
V4L2_MPEG_VIDEO_VP9_PROFILE_1	Profile 1
V4L2_MPEG_VIDEO_VP9_PROFILE_2	Profile 2
V4L2_MPEG_VIDEO_VP9_PROFILE_3	Profile 3

`V4L2_CID_MPEG_VIDEO_VP9_LEVEL (enum)`

enum v4l2_mpeg_video_vp9_level -

This control allows selecting the level for VP9 encoder. This is also used to enumerate supported levels by VP9 encoder or decoder. More information can be found at webmproject. Possible values are:

V4L2_MPEG_VIDEO_VP9_LEVEL_1_0	Level 1
V4L2_MPEG_VIDEO_VP9_LEVEL_1_1	Level 1.1
V4L2_MPEG_VIDEO_VP9_LEVEL_2_0	Level 2
V4L2_MPEG_VIDEO_VP9_LEVEL_2_1	Level 2.1
V4L2_MPEG_VIDEO_VP9_LEVEL_3_0	Level 3
V4L2_MPEG_VIDEO_VP9_LEVEL_3_1	Level 3.1
V4L2_MPEG_VIDEO_VP9_LEVEL_4_0	Level 4
V4L2_MPEG_VIDEO_VP9_LEVEL_4_1	Level 4.1
V4L2_MPEG_VIDEO_VP9_LEVEL_5_0	Level 5
V4L2_MPEG_VIDEO_VP9_LEVEL_5_1	Level 5.1
V4L2_MPEG_VIDEO_VP9_LEVEL_5_2	Level 5.2
V4L2_MPEG_VIDEO_VP9_LEVEL_6_0	Level 6
V4L2_MPEG_VIDEO_VP9_LEVEL_6_1	Level 6.1
V4L2_MPEG_VIDEO_VP9_LEVEL_6_2	Level 6.2

High Efficiency Video Coding (HEVC/H.265) Control Reference

The HEVC/H.265 controls include controls for encoding parameters of HEVC/H.265 video codec.

HEVC/H.265 Control IDs

V4L2_CID_MPEG_VIDEO_HEVC_MIN_QP (integer)
Minimum quantization parameter for HEVC. Valid range: from 0 to 51.

V4L2_CID_MPEG_VIDEO_HEVC_MAX_QP (integer)
Maximum quantization parameter for HEVC. Valid range: from 0 to 51.

V4L2_CID_MPEG_VIDEO_HEVC_I_FRAME_QP (integer)
Quantization parameter for an I frame for HEVC. Valid range: [V4L2_CID_MPEG_VIDEO_HEVC_MIN_QP, V4L2_CID_MPEG_VIDEO_HEVC_MAX_QP].

V4L2_CID_MPEG_VIDEO_HEVC_P_FRAME_QP (integer)
Quantization parameter for a P frame for HEVC. Valid range: [V4L2_CID_MPEG_VIDEO_HEVC_MIN_QP, V4L2_CID_MPEG_VIDEO_HEVC_MAX_QP].

V4L2_CID_MPEG_VIDEO_HEVC_B_FRAME_QP (integer)
Quantization parameter for a B frame for HEVC. Valid range: [V4L2_CID_MPEG_VIDEO_HEVC_MIN_QP, V4L2_CID_MPEG_VIDEO_HEVC_MAX_QP].

V4L2_CID_MPEG_VIDEO_HEVC_HIER_QP (boolean)
HIERARCHICAL_QP allows the host to specify the quantization parameter values for each temporal layer through HIERARCHICAL_QP_LAYER. This is valid only if HIERARCHICAL_CODING_LAYER is greater than 1. Setting the control value to 1 enables setting of the QP values for the layers.

V4L2_CID_MPEG_VIDEO_HEVC_HIER_CODING_TYPE
(enum)

enum v4l2_mpeg_video_hevc_hier_coding_type -
Selects the hierarchical coding type for encoding. Possible values are:

V4L2_MPEG_VIDEO_HEVC_HIERARCHICAL_CODING_B	Use the B frame for hierarchical coding.
V4L2_MPEG_VIDEO_HEVC_HIERARCHICAL_CODING_P	Use the P frame for hierarchical coding.

V4L2_CID_MPEG_VIDEO_HEVC_HIER_CODING_LAYER (integer)
Selects the hierarchical coding layer. In normal encoding (non-hierarchial coding), it should be zero. Possible values are [0, 6]. 0 indicates HIERARCHICAL CODING LAYER 0, 1 indicates HIERARCHICAL CODING LAYER 1 and so on.

V4L2_CID_MPEG_VIDEO_HEVC_HIER_CODING_L0_QP (integer)
Indicates quantization parameter for hierarchical coding layer 0. Valid range: [V4L2_CID_MPEG_VIDEO_HEVC_MIN_QP, V4L2_CID_MPEG_VIDEO_HEVC_MAX_QP].

V4L2_CID_MPEG_VIDEO_HEVC_HIER_CODING_L1_QP (integer)
Indicates quantization parameter for hierarchical coding layer 1. Valid range: [V4L2_CID_MPEG_VIDEO_HEVC_MIN_QP, V4L2_CID_MPEG_VIDEO_HEVC_MAX_QP].

V4L2_CID_MPEG_VIDEO_HEVC_HIER_CODING_L2_QP (integer)
Indicates quantization parameter for hierarchical coding layer 2. Valid range: [V4L2_CID_MPEG_VIDEO_HEVC_MIN_QP, V4L2_CID_MPEG_VIDEO_HEVC_MAX_QP].

V4L2_CID_MPEG_VIDEO_HEVC_HIER_CODING_L3_QP (integer)
Indicates quantization parameter for hierarchical coding layer 3. Valid range: [V4L2_CID_MPEG_VIDEO_HEVC_MIN_QP, V4L2_CID_MPEG_VIDEO_HEVC_MAX_QP].

V4L2_CID_MPEG_VIDEO_HEVC_HIER_CODING_L4_QP (integer)
Indicates quantization parameter for hierarchical coding layer 4. Valid range: [V4L2_CID_MPEG_VIDEO_HEVC_MIN_QP, V4L2_CID_MPEG_VIDEO_HEVC_MAX_QP].

V4L2_CID_MPEG_VIDEO_HEVC_HIER_CODING_L5_QP (integer)
Indicates quantization parameter for hierarchical coding layer 5. Valid range: [V4L2_CID_MPEG_VIDEO_HEVC_MIN_QP, V4L2_CID_MPEG_VIDEO_HEVC_MAX_QP].

V4L2_CID_MPEG_VIDEO_HEVC_HIER_CODING_L6_QP (integer)
Indicates quantization parameter for hierarchical coding layer 6. Valid range: [V4L2_CID_MPEG_VIDEO_HEVC_MIN_QP, V4L2_CID_MPEG_VIDEO_HEVC_MAX_QP].

V4L2_CID_MPEG_VIDEO_HEVC_PROFILE
(enum)

enum v4l2_mpeg_video_hevc_profile -
Select the desired profile for HEVC encoder.

V4L2_MPEG_VIDEO_HEVC_PROFILE_MAIN	Main profile.
V4L2_MPEG_VIDEO_HEVC_PROFILE_MAIN_STILL_PICTURE	Main still picture profile.
V4L2_MPEG_VIDEO_HEVC_PROFILE_MAIN_10	Main 10 profile.

V4L2_CID_MPEG_VIDEO_HEVC_LEVEL
(enum)

enum v4l2_mpeg_video_hevc_level -
Selects the desired level for HEVC encoder.

V4L2_MPEG_VIDEO_HEVC_LEVEL_1	Level 1.0
V4L2_MPEG_VIDEO_HEVC_LEVEL_2	Level 2.0
V4L2_MPEG_VIDEO_HEVC_LEVEL_2_1	Level 2.1
V4L2_MPEG_VIDEO_HEVC_LEVEL_3	Level 3.0
V4L2_MPEG_VIDEO_HEVC_LEVEL_3_1	Level 3.1
V4L2_MPEG_VIDEO_HEVC_LEVEL_4	Level 4.0
V4L2_MPEG_VIDEO_HEVC_LEVEL_4_1	Level 4.1
V4L2_MPEG_VIDEO_HEVC_LEVEL_5	Level 5.0
V4L2_MPEG_VIDEO_HEVC_LEVEL_5_1	Level 5.1
V4L2_MPEG_VIDEO_HEVC_LEVEL_5_2	Level 5.2
V4L2_MPEG_VIDEO_HEVC_LEVEL_6	Level 6.0
V4L2_MPEG_VIDEO_HEVC_LEVEL_6_1	Level 6.1
V4L2_MPEG_VIDEO_HEVC_LEVEL_6_2	Level 6.2

V4L2_CID_MPEG_VIDEO_HEVC_FRAME_RATE_RESOLUTION (integer)
Indicates the number of evenly spaced subintervals, called ticks, within one second. This is a 16 bit unsigned integer and has a maximum value up to 0xffff and a minimum value of 1.

V4L2_CID_MPEG_VIDEO_HEVC_TIER
(enum)

enum v4l2_mpeg_video_hevc_tier -
TIER_FLAG specifies tiers information of the HEVC encoded picture. Tier were made to deal with applications that differ in terms of maximum bit rate. Setting the flag to 0 selects HEVC tier as Main tier and setting this flag to 1 indicates High tier. High tier is for applications requiring high bit rates.

V4L2_MPEG_VIDEO_HEVC_TIER_MAIN	Main tier.
V4L2_MPEG_VIDEO_HEVC_TIER_HIGH	High tier.

V4L2_CID_MPEG_VIDEO_HEVC_MAX_PARTITION_DEPTH (integer)
Selects HEVC maximum coding unit depth.

V4L2_CID_MPEG_VIDEO_HEVC_LOOP_FILTER_MODE
(enum)

enum v4l2_mpeg_video_hevc_loop_filter_mode -
Loop filter mode for HEVC encoder. Possible values are:

V4L2_MPEG_VIDEO_HEVC_LOOP_FILTER_MODE_DISABLED	Loop filter is disabled.
V4L2_MPEG_VIDEO_HEVC_LOOP_FILTER_MODE_ENABLED	Loop filter is enabled.
V4L2_MPEG_VIDEO_HEVC_LOOP_FILTER_MODE_DISABLED_AT_SLICE_BOUNDARY	Loop filter is disabled at the slice boundary.

V4L2_CID_MPEG_VIDEO_HEVC_LF_BETA_OFFSET_DIV2 (integer)
Selects HEVC loop filter beta offset. The valid range is [-6, +6].

V4L2_CID_MPEG_VIDEO_HEVC_LF_TC_OFFSET_DIV2 (integer)
Selects HEVC loop filter tc offset. The valid range is [-6, +6].

V4L2_CID_MPEG_VIDEO_HEVC_REFRESH_TYPE
(enum)

enum v4l2_mpeg_video_hevc_hier_refresh_type -
Selects refresh type for HEVC encoder. Host has to specify the period into V4L2_CID_MPEG_VIDEO_HEVC_REFRESH_PERIOD.

V4L2_MPEG_VIDEO_HEVC_REFRESH_NONE	Use the B frame for hierarchical coding.
V4L2_MPEG_VIDEO_HEVC_REFRESH_CRA	Use CRA (Clean Random Access Unit) picture encoding.
V4L2_MPEG_VIDEO_HEVC_REFRESH_IDR	Use IDR (Instantaneous Decoding Refresh) picture encoding.

V4L2_CID_MPEG_VIDEO_HEVC_REFRESH_PERIOD (integer)
Selects the refresh period for HEVC encoder. This specifies the number of I pictures between two CRA/IDR pictures. This is valid only if REFRESH_TYPE is not 0.

V4L2_CID_MPEG_VIDEO_HEVC_LOSSLESS_CU (boolean)
Indicates HEVC lossless encoding. Setting it to 0 disables lossless encoding. Setting it to 1 enables lossless encoding.

V4L2_CID_MPEG_VIDEO_HEVC_CONST_INTRA_PRED (boolean)
Indicates constant intra prediction for HEVC encoder. Specifies the constrained intra prediction in which intra largest coding unit (LCU) prediction is performed by using residual data and decoded samples of neighboring intra LCU only. Setting the value to 1 enables constant intra prediction and setting the value to 0 disables constant intra prediction.

V4L2_CID_MPEG_VIDEO_HEVC_WAVEFRONT (boolean)
Indicates wavefront parallel processing for HEVC encoder. Setting it to 0 disables the feature and setting it to 1 enables the wavefront parallel processing.

V4L2_CID_MPEG_VIDEO_HEVC_GENERAL_PB (boolean)
Setting the value to 1 enables combination of P and B frame for HEVC encoder.

V4L2_CID_MPEG_VIDEO_HEVC_TEMPORAL_ID (boolean)
Indicates temporal identifier for HEVC encoder which is enabled by setting the value to 1.

V4L2_CID_MPEG_VIDEO_HEVC_STRONG_SMOOTHING (boolean)
Indicates bi-linear interpolation is conditionally used in the intra prediction filtering process in the CVS when set to 1. Indicates bi-linear interpolation is not used in the CVS when set to 0.

V4L2_CID_MPEG_VIDEO_HEVC_MAX_NUM_MERGE_MV_MINUS1 (integer)
Indicates maximum number of merge candidate motion vectors. Values are from 0 to 4.

V4L2_CID_MPEG_VIDEO_HEVC_TMV_PREDICTION (boolean)
Indicates temporal motion vector prediction for HEVC encoder. Setting it to 1 enables the prediction. Setting it to 0 disables the prediction.

V4L2_CID_MPEG_VIDEO_HEVC_WITHOUT_STARTCODE (boolean)
Specifies if HEVC generates a stream with a size of the length field instead of start code pattern. The size of the length field is configurable through the V4L2_CID_MPEG_VIDEO_HEVC_SIZE_OF_LENGTH_FIELD control. Setting

the value to 0 disables encoding without startcode pattern. Setting the value to 1 will enables encoding without startcode pattern.

V4L2_CID_MPEG_VIDEO_HEVC_SIZE_OF_LENGTH_FIELD (enum)

enum v4l2_mpeg_video_hevc_size_of_length_field -
Indicates the size of length field. This is valid when encoding WITHOUT_STARTCODE_ENABLE is enabled.

V4L2_MPEG_VIDEO_HEVC_SIZE_0	Generate start code pattern (Normal).
V4L2_MPEG_VIDEO_HEVC_SIZE_1	Generate size of length field instead of start code pattern and length is 1.
V4L2_MPEG_VIDEO_HEVC_SIZE_2	Generate size of length field instead of start code pattern and length is 2.
V4L2_MPEG_VIDEO_HEVC_SIZE_4	Generate size of length field instead of start code pattern and length is 4.

V4L2_CID_MPEG_VIDEO_HEVC_HIER_CODING_L0_BR (integer)
Indicates bit rate for hierarchical coding layer 0 for HEVC encoder.

V4L2_CID_MPEG_VIDEO_HEVC_HIER_CODING_L1_BR (integer)
Indicates bit rate for hierarchical coding layer 1 for HEVC encoder.

V4L2_CID_MPEG_VIDEO_HEVC_HIER_CODING_L2_BR (integer)
Indicates bit rate for hierarchical coding layer 2 for HEVC encoder.

V4L2_CID_MPEG_VIDEO_HEVC_HIER_CODING_L3_BR (integer)
Indicates bit rate for hierarchical coding layer 3 for HEVC encoder.

V4L2_CID_MPEG_VIDEO_HEVC_HIER_CODING_L4_BR (integer)
Indicates bit rate for hierarchical coding layer 4 for HEVC encoder.

V4L2_CID_MPEG_VIDEO_HEVC_HIER_CODING_L5_BR (integer)
Indicates bit rate for hierarchical coding layer 5 for HEVC encoder.

V4L2_CID_MPEG_VIDEO_HEVC_HIER_CODING_L6_BR (integer)
Indicates bit rate for hierarchical coding layer 6 for HEVC encoder.

V4L2_CID_MPEG_VIDEO_REF_NUMBER_FOR_PFRAMES (integer)
Selects number of P reference pictures required for HEVC encoder. P-Frame can use 1 or 2 frames for reference.

V4L2_CID_MPEG_VIDEO_PREPEND_SPSPPS_TO_IDR (integer)
Indicates whether to generate SPS and PPS at every IDR. Setting it to 0 disables generating SPS and PPS at every IDR. Setting it to one enables generating SPS and PPS at every IDR.

V4L2_CID_MPEG_VIDEO_HEVC_SPS (struct)
Specifies the Sequence Parameter Set fields (as extracted from the bitstream) for the associated HEVC slice data. These bitstream parameters are defined according to *ITU H.265/HEVC*. They are described in section 7.4.3.2 "Sequence parameter set RBSP semantics" of the specification.

type **v4l2_ctrl_hevc_sps**

Table 33: struct v4l2_ctrl_hevc_sps

__u16	pic_width_in_luma_samples	
__u16	pic_height_in_luma_samples	
__u8	bit_depth_luma_minus8	
__u8	bit_depth_chroma_minus8	
__u8	log2_max_pic_order_cnt_lsb_minus4	
__u8	sps_max_dec_pic_buffering_minus1	
__u8	sps_max_num_reorder_pics	
__u8	sps_max_latency_increase_plus1	
__u8	log2_min_luma_coding_block_size_minus3	
__u8	log2_diff_max_min_luma_coding_block_size	
__u8	log2_min_luma_transform_block_size_minus2	
__u8	log2_diff_max_min_luma_transform_block_size	
__u8	max_transform_hierarchy_depth_inter	
__u8	max_transform_hierarchy_depth_intra	
__u8	pcm_sample_bit_depth_luma_minus1	
__u8	pcm_sample_bit_depth_chroma_minus1	
__u8	log2_min_pcm_luma_coding_block_size_minus3	
__u8	log2_diff_max_min_pcm_luma_coding_block_size	
__u8	num_short_term_ref_pic_sets	
__u8	num_long_term_ref_pics_sps	
__u8	chroma_format_idc	
__u64	flags	See *Sequence Parameter Set F*

Sequence Parameter Set Flags

V4L2_HEVC_SPS_FLAG_SEPARATE_COLOUR_PLANE	0x00000001
V4L2_HEVC_SPS_FLAG_SCALING_LIST_ENABLED	0x00000002
V4L2_HEVC_SPS_FLAG_AMP_ENABLED	0x00000004
V4L2_HEVC_SPS_FLAG_SAMPLE_ADAPTIVE_OFFSET	0x00000008
V4L2_HEVC_SPS_FLAG_PCM_ENABLED	0x00000010
V4L2_HEVC_SPS_FLAG_PCM_LOOP_FILTER_DISABLED	0x00000020
V4L2_HEVC_SPS_FLAG_LONG_TERM_REF_PICS_PRESENT	0x00000040
V4L2_HEVC_SPS_FLAG_SPS_TEMPORAL_MVP_ENABLED	0x00000080
V4L2_HEVC_SPS_FLAG_STRONG_INTRA_SMOOTHING_ENABLED	0x00000100

V4L2_CID_MPEG_VIDEO_HEVC_PPS (struct)

Specifies the Picture Parameter Set fields (as extracted from the bitstream) for the associated HEVC slice data. These bitstream parameters are defined according to *ITU H.265/HEVC*. They are described in section 7.4.3.3 "Picture parameter set RBSP semantics" of the specification.

type **v4l2_ctrl_hevc_pps**

Table 35: struct v4l2_ctrl_hevc_pps

__u8	num_extra_slice_header_bits
__s8	init_qp_minus26
__u8	diff_cu_qp_delta_depth

continues on next page

Table 35 – continued from previous page

__s8	pps_cb_qp_offset	
__s8	pps_cr_qp_offset	
__u8	num_tile_columns_minus1	
__u8	num_tile_rows_minus1	
__u8	column_width_minus1[20]	
__u8	row_height_minus1[22]	
__s8	pps_beta_offset_div2	
__s8	pps_tc_offset_div2	
__u8	log2_parallel_merge_level_minus2	
__u8	padding[4]	Applications and drivers must set this to zero
__u64	flags	See *Picture Parameter Set Flags*

Picture Parameter Set Flags

V4L2_HEVC_PPS_FLAG_DEPENDENT_SLICE_SEGMENT_ENABLED	0x00000001
V4L2_HEVC_PPS_FLAG_OUTPUT_FLAG_PRESENT	0x00000002
V4L2_HEVC_PPS_FLAG_SIGN_DATA_HIDING_ENABLED	0x00000004
V4L2_HEVC_PPS_FLAG_CABAC_INIT_PRESENT	0x00000008
V4L2_HEVC_PPS_FLAG_CONSTRAINED_INTRA_PRED	0x00000010
V4L2_HEVC_PPS_FLAG_TRANSFORM_SKIP_ENABLED	0x00000020
V4L2_HEVC_PPS_FLAG_CU_QP_DELTA_ENABLED	0x00000040
V4L2_HEVC_PPS_FLAG_PPS_SLICE_CHROMA_QP_OFFSETS_PRESENT	0x00000080
V4L2_HEVC_PPS_FLAG_WEIGHTED_PRED	0x00000100
V4L2_HEVC_PPS_FLAG_WEIGHTED_BIPRED	0x00000200
V4L2_HEVC_PPS_FLAG_TRANSQUANT_BYPASS_ENABLED	0x00000400
V4L2_HEVC_PPS_FLAG_TILES_ENABLED	0x00000800
V4L2_HEVC_PPS_FLAG_ENTROPY_CODING_SYNC_ENABLED	0x00001000
V4L2_HEVC_PPS_FLAG_LOOP_FILTER_ACROSS_TILES_ENABLED	0x00002000
V4L2_HEVC_PPS_FLAG_PPS_LOOP_FILTER_ACROSS_SLICES_ENABLED	0x00004000
V4L2_HEVC_PPS_FLAG_DEBLOCKING_FILTER_OVERRIDE_ENABLED	0x00008000
V4L2_HEVC_PPS_FLAG_PPS_DISABLE_DEBLOCKING_FILTER	0x00010000
V4L2_HEVC_PPS_FLAG_LISTS_MODIFICATION_PRESENT	0x00020000
V4L2_HEVC_PPS_FLAG_SLICE_SEGMENT_HEADER_EXTENSION_PRESENT	0x00040000

V4L2_CID_MPEG_VIDEO_HEVC_SLICE_PARAMS (struct)

Specifies various slice-specific parameters, especially from the NAL unit header, general slice segment header and weighted prediction parameter parts of the bitstream. These bitstream parameters are defined according to *ITU H.265/HEVC*. They are described in section 7.4.7 "General slice segment header semantics" of the specification.

type **v4l2_ctrl_hevc_slice_params**

Table 37: struct v4l2_ctr

__u32	bit_size	Si
__u32	data_bit_offset	Of
__u8	nal_unit_type	

Table 37 – continued f

__u8	nuh_temporal_id_plus1	
__u8	slice_type	(V
__u8	colour_plane_id	
__u16	slice_pic_order_cnt	
__u8	num_ref_idx_l0_active_minus1	
__u8	num_ref_idx_l1_active_minus1	
__u8	collocated_ref_idx	
__u8	five_minus_max_num_merge_cand	
__s8	slice_qp_delta	
__s8	slice_cb_qp_offset	
__s8	slice_cr_qp_offset	
__s8	slice_act_y_qp_offset	
__s8	slice_act_cb_qp_offset	
__s8	slice_act_cr_qp_offset	
__s8	slice_beta_offset_div2	
__s8	slice_tc_offset_div2	
__u8	pic_struct	
__u8	num_active_dpb_entries	Th
__u8	ref_idx_l0[V4L2_HEVC_DPB_ENTRIES_NUM_MAX]	Th
__u8	ref_idx_l1[V4L2_HEVC_DPB_ENTRIES_NUM_MAX]	Th
__u8	num_rps_poc_st_curr_before	Th
__u8	num_rps_poc_st_curr_after	Th
__u8	num_rps_poc_lt_curr	Th
__u8	padding[7]	Ap
struct *v4l2_hevc_dpb_entry*	dpb[V4L2_HEVC_DPB_ENTRIES_NUM_MAX]	Th
struct *v4l2_hevc_pred_weight_table*	pred_weight_table	Th
__u64	flags	Se

Slice Parameters Flags

V4L2_HEVC_SLICE_PARAMS_FLAG_SLICE_SAO_LUMA	0x00000001
V4L2_HEVC_SLICE_PARAMS_FLAG_SLICE_SAO_CHROMA	0x00000002
V4L2_HEVC_SLICE_PARAMS_FLAG_SLICE_TEMPORAL_MVP_ENABLED	0x00000004
V4L2_HEVC_SLICE_PARAMS_FLAG_MVD_L1_ZERO	0x00000008
V4L2_HEVC_SLICE_PARAMS_FLAG_CABAC_INIT	0x00000010
V4L2_HEVC_SLICE_PARAMS_FLAG_COLLOCATED_FROM_L0	0x00000020
V4L2_HEVC_SLICE_PARAMS_FLAG_USE_INTEGER_MV	0x00000040
V4L2_HEVC_SLICE_PARAMS_FLAG_SLICE_DEBLOCKING_FILTER_DISABLED	0x00000080
V4L2_HEVC_SLICE_PARAMS_FLAG_SLICE_LOOP_FILTER_ACROSS_SLICES_ENABLED	0x00000100
V4L2_HEVC_SLICE_PARAMS_FLAG_DEPENDENT_SLICE_SEGMENT	0x00000200

type **v4l2_hevc_dpb_entry**

__u64	timestamp	Timestamp of the V4L2 capture buffer to use as reference, used
__u8	rps	The reference set for the reference frame (V4L2_HEVC_DPB_EN
__u8	field_pic	Whether the reference is a field picture or a frame.

__u16	pic_order_cnt[2]	The picture order count of the reference. Only the first element
__u8	padding[2]	Applications and drivers must set this to zero.

type **v4l2_hevc_pred_weight_table**

Table 40: struct v4l2_hevc_pred_weight_table

__u8	luma_log2_weight_denom	
__s8	delta_chroma_log2_weight_denom	
__s8	delta_luma_weight_l0[V4L2_HEVC_DPB_ENTRIES_NUM_MAX]	
__s8	luma_offset_l0[V4L2_HEVC_DPB_ENTRIES_NUM_MAX]	
__s8	delta_chroma_weight_l0[V4L2_HEVC_DPB_ENTRIES_NUM_MAX][2]	
__s8	chroma_offset_l0[V4L2_HEVC_DPB_ENTRIES_NUM_MAX][2]	
__s8	delta_luma_weight_l1[V4L2_HEVC_DPB_ENTRIES_NUM_MAX]	
__s8	luma_offset_l1[V4L2_HEVC_DPB_ENTRIES_NUM_MAX]	
__s8	delta_chroma_weight_l1[V4L2_HEVC_DPB_ENTRIES_NUM_MAX][2]	
__s8	chroma_offset_l1[V4L2_HEVC_DPB_ENTRIES_NUM_MAX][2]	
__u8	padding[6]	Applications and d

V4L2_CID_MPEG_VIDEO_HEVC_DECODE_MODE (enum)

Specifies the decoding mode to use. Currently exposes slice-based and frame-based decoding but new modes might be added later on. This control is used as a modifier for V4L2_PIX_FMT_HEVC_SLICE pixel format. Applications that support V4L2_PIX_FMT_HEVC_SLICE are required to set this control in order to specify the decoding mode that is expected for the buffer. Drivers may expose a single or multiple decoding modes, depending on what they can support.

Note: This menu control is not yet part of the public kernel API and it is expected to change.

type **v4l2_mpeg_video_hevc_decode_mode**

V4L2_MPEG_VIDEO_HEVC_DECODE_MODE_SLICE_BASED	0	Decoding is done at the slice granul
V4L2_MPEG_VIDEO_HEVC_DECODE_MODE_FRAME_BASED	1	Decoding is done at the frame gran

V4L2_CID_MPEG_VIDEO_HEVC_START_CODE (enum)

Specifies the HEVC slice start code expected for each slice. This control is used as a modifier for V4L2_PIX_FMT_HEVC_SLICE pixel format. Applications that support V4L2_PIX_FMT_HEVC_SLICE are required to set this control in order to specify the start code that is expected for the buffer. Drivers may expose a single or multiple start codes, depending on what they can support.

Note: This menu control is not yet part of the public kernel API and it is expected to change.

type **v4l2_mpeg_video_hevc_start_code**

V4L2_MPEG_VIDEO_HEVC_START_CODE_NONE	0	Selecting this value specifies that HEVC s
V4L2_MPEG_VIDEO_HEVC_START_CODE_ANNEX_B	1	Selecting this value specifies that HEVC s

JPEG Control Reference

The JPEG class includes controls for common features of JPEG encoders and decoders. Currently it includes features for codecs implementing progressive baseline DCT compression process with Huffman entrophy coding.

JPEG Control IDs

V4L2_CID_JPEG_CLASS (class)
The JPEG class descriptor. Calling *ioctls VIDIOC_QUERYCTRL, VIDIOC_QUERY_EXT_CTRL and VIDIOC_QUERYMENU* for this control will return a description of this control class.

V4L2_CID_JPEG_CHROMA_SUBSAMPLING (menu)
The chroma subsampling factors describe how each component of an input image is sampled, in respect to maximum sample rate in each spatial dimension. See *ITU-T.81*, clause A.1.1. for more details. The V4L2_CID_JPEG_CHROMA_SUBSAMPLING control determines how Cb and Cr components are downsampled after converting an input image from RGB to Y' CbCr color space.

V4L2_JPEG_CHROMA_SUBSAMPLING_444	No chroma subsampling, each pixel has Y, Cr and Cb values.
V4L2_JPEG_CHROMA_SUBSAMPLING_422	Horizontally subsample Cr, Cb components by a factor of 2.
V4L2_JPEG_CHROMA_SUBSAMPLING_420	Subsample Cr, Cb components horizontally and vertically by 2.
V4L2_JPEG_CHROMA_SUBSAMPLING_411	Horizontally subsample Cr, Cb components by a factor of 4.
V4L2_JPEG_CHROMA_SUBSAMPLING_410	Subsample Cr, Cb components horizontally by 4 and vertically by 2.
V4L2_JPEG_CHROMA_SUBSAMPLING_GRAY	Use only luminance component.

V4L2_CID_JPEG_RESTART_INTERVAL (integer)
The restart interval determines an interval of inserting RSTm markers (m = 0..7). The purpose of these markers is to additionally reinitialize the encoder process, in order to process blocks of an image independently. For the lossy compression processes the restart interval unit is MCU (Minimum Coded Unit) and its value is contained in DRI (Define Restart Interval) marker. If V4L2_CID_JPEG_RESTART_INTERVAL control is set to 0, DRI and RSTm markers will not be inserted.

V4L2_CID_JPEG_COMPRESSION_QUALITY (integer)
V4L2_CID_JPEG_COMPRESSION_QUALITY control determines trade-off between

image quality and size. It provides simpler method for applications to control image quality, without a need for direct reconfiguration of luminance and chrominance quantization tables. In cases where a driver uses quantization tables configured directly by an application, using interfaces defined elsewhere, `V4L2_CID_JPEG_COMPRESSION_QUALITY` control should be set by driver to 0.

The value range of this control is driver-specific. Only positive, non-zero values are meaningful. The recommended range is 1 - 100, where larger values correspond to better image quality.

`V4L2_CID_JPEG_ACTIVE_MARKER (bitmask)`
Specify which JPEG markers are included in compressed stream. This control is valid only for encoders.

`V4L2_JPEG_ACTIVE_MARKER_APP0`	Application data segment APP_0.
`V4L2_JPEG_ACTIVE_MARKER_APP1`	Application data segment APP_1.
`V4L2_JPEG_ACTIVE_MARKER_COM`	Comment segment.
`V4L2_JPEG_ACTIVE_MARKER_DQT`	Quantization tables segment.
`V4L2_JPEG_ACTIVE_MARKER_DHT`	Huffman tables segment.

For more details about JPEG specification, refer to *ITU-T.81*, *JFIF*, *W3C JPEG JFIF*.

Digital Video Control Reference

The Digital Video control class is intended to control receivers and transmitters for VGA, DVI (Digital Visual Interface), HDMI (*HDMI*) and DisplayPort (*DP*). These controls are generally expected to be private to the receiver or transmitter subdevice that implements them, so they are only exposed on the `/dev/v4l-subdev*` device node.

Note: Note that these devices can have multiple input or output pads which are hooked up to e.g. HDMI connectors. Even though the subdevice will receive or transmit video from/to only one of those pads, the other pads can still be active when it comes to EDID (Extended Display Identification Data, *EDID*) and HDCP (High-bandwidth Digital Content Protection System, *HDCP*) processing, allowing the device to do the fairly slow EDID/HDCP handling in advance. This allows for quick switching between connectors.

These pads appear in several of the controls in this section as bitmasks, one bit for each pad. Bit 0 corresponds to pad 0, bit 1 to pad 1, etc. The maximum value of the control is the set of valid pads.

Digital Video Control IDs

V4L2_CID_DV_CLASS (class)
The Digital Video class descriptor.

V4L2_CID_DV_TX_HOTPLUG (bitmask)
Many connectors have a hotplug pin which is high if EDID information is available from the source. This control shows the state of the hotplug pin as seen by the transmitter. Each bit corresponds to an output pad on the transmitter. If an output pad does not have an associated hotplug pin, then the bit for that pad will be 0. This read-only control is applicable to DVI-D, HDMI and DisplayPort connectors.

V4L2_CID_DV_TX_RXSENSE (bitmask)
Rx Sense is the detection of pull-ups on the TMDS clock lines. This normally means that the sink has left/entered standby (i.e. the transmitter can sense that the receiver is ready to receive video). Each bit corresponds to an output pad on the transmitter. If an output pad does not have an associated Rx Sense, then the bit for that pad will be 0. This read-only control is applicable to DVI-D and HDMI devices.

V4L2_CID_DV_TX_EDID_PRESENT (bitmask)
When the transmitter sees the hotplug signal from the receiver it will attempt to read the EDID. If set, then the transmitter has read at least the first block (= 128 bytes). Each bit corresponds to an output pad on the transmitter. If an output pad does not support EDIDs, then the bit for that pad will be 0. This read-only control is applicable to VGA, DVI-A/D, HDMI and DisplayPort connectors.

V4L2_CID_DV_TX_MODE
(enum)

enum v4l2_dv_tx_mode -
HDMI transmitters can transmit in DVI-D mode (just video) or in HDMI mode (video + audio + auxiliary data). This control selects which mode to use: V4L2_DV_TX_MODE_DVI_D or V4L2_DV_TX_MODE_HDMI. This control is applicable to HDMI connectors.

V4L2_CID_DV_TX_RGB_RANGE
(enum)

enum v4l2_dv_rgb_range -
Select the quantization range for RGB output. V4L2_DV_RANGE_AUTO follows the RGB quantization range specified in the standard for the video interface (ie. *CEA-861-E* for HDMI). V4L2_DV_RANGE_LIMITED and V4L2_DV_RANGE_FULL override the standard to be compatible with sinks that have not implemented the standard correctly (unfortunately quite common for HDMI and DVI-D). Full range allows all possible values to be used whereas limited range sets the range to (16 << (N-8)) - (235 << (N-8)) where N is the number of bits per component. This control is applicable to VGA, DVI-A/D, HDMI and DisplayPort connectors.

V4L2_CID_DV_TX_IT_CONTENT_TYPE
(enum)

enum v4l2_dv_it_content_type -
Configures the IT Content Type of the transmitted video. This information is sent over HDMI and DisplayPort connectors as part of the AVI InfoFrame. The term 'IT Content' is used for content that originates from a computer as opposed to content from a TV broadcast or an analog source. The enum v4l2_dv_it_content_type defines the possible content types:

`V4L2_DV_IT_CONTENT_TYPE_GRAPHICS`	Graphics content. Pixel data should be passed unfiltered and without analog reconstruction.
`V4L2_DV_IT_CONTENT_TYPE_PHOTO`	Photo content. The content is derived from digital still pictures. The content should be passed through with minimal scaling and picture enhancements.
`V4L2_DV_IT_CONTENT_TYPE_CINEMA`	Cinema content.
`V4L2_DV_IT_CONTENT_TYPE_GAME`	Game content. Audio and video latency should be minimized.
`V4L2_DV_IT_CONTENT_TYPE_NO_ITC`	No IT Content information is available and the ITC bit in the AVI InfoFrame is set to 0.

`V4L2_CID_DV_RX_POWER_PRESENT (bitmask)`
Detects whether the receiver receives power from the source (e.g. HDMI carries 5V on one of the pins). This is often used to power an eeprom which contains EDID information, such that the source can read the EDID even if the sink is in standby/power off. Each bit corresponds to an input pad on the receiver. If an input pad cannot detect whether power is present, then the bit for that pad will be 0. This read-only control is applicable to DVI-D, HDMI and DisplayPort connectors.

`V4L2_CID_DV_RX_RGB_RANGE`
(enum)

enum v4l2_dv_rgb_range -
Select the quantization range for RGB input. V4L2_DV_RANGE_AUTO follows the RGB quantization range specified in the standard for the video interface (ie. *CEA-861-E* for HDMI). V4L2_DV_RANGE_LIMITED and V4L2_DV_RANGE_FULL override the standard to be compatible with sources that have not implemented the standard correctly (unfortunately quite common for HDMI and DVI-D). Full range allows all possible values to be used whereas limited range sets the range to (16 << (N-8)) - (235 << (N-8)) where N is the number of bits per component. This control is applicable to VGA, DVI-A/D, HDMI and DisplayPort connectors.

`V4L2_CID_DV_RX_IT_CONTENT_TYPE`
(enum)

enum v4l2_dv_it_content_type -
Reads the IT Content Type of the received video. This information is sent over HDMI and DisplayPort connectors as part of the AVI InfoFrame. The term 'IT Content' is used for content that originates from a computer as opposed to content from a TV broadcast or an analog source. See `V4L2_CID_DV_TX_IT_CONTENT_TYPE` for the available content types.

RF Tuner Control Reference

The RF Tuner (RF_TUNER) class includes controls for common features of devices having RF tuner.

In this context, RF tuner is radio receiver circuit between antenna and demodulator. It receives radio frequency (RF) from the antenna and converts that received signal to lower intermediate frequency (IF) or baseband frequency (BB). Tuners that could do baseband output are often called Zero-IF tuners. Older tuners were typically simple PLL tuners inside a metal box, while newer ones are highly integrated chips without a metal box "silicon tuners". These controls are mostly applicable for new feature rich silicon tuners, just because older tuners does not have much adjustable features.

For more information about RF tuners see Tuner (radio) and RF front end from Wikipedia.

RF_TUNER Control IDs

V4L2_CID_RF_TUNER_CLASS (class)
: The RF_TUNER class descriptor. Calling *ioctls VIDIOC_QUERYCTRL, VIDIOC_QUERY_EXT_CTRL and VIDIOC_QUERYMENU* for this control will return a description of this control class.

V4L2_CID_RF_TUNER_BANDWIDTH_AUTO (boolean)
: Enables/disables tuner radio channel bandwidth configuration. In automatic mode bandwidth configuration is performed by the driver.

V4L2_CID_RF_TUNER_BANDWIDTH (integer)
: Filter(s) on tuner signal path are used to filter signal according to receiving party needs. Driver configures filters to fulfill desired bandwidth requirement. Used when V4L2_CID_RF_TUNER_BANDWIDTH_AUTO is not set. Unit is in Hz. The range and step are driver-specific.

V4L2_CID_RF_TUNER_LNA_GAIN_AUTO (boolean)
: Enables/disables LNA automatic gain control (AGC)

V4L2_CID_RF_TUNER_MIXER_GAIN_AUTO (boolean)
: Enables/disables mixer automatic gain control (AGC)

V4L2_CID_RF_TUNER_IF_GAIN_AUTO (boolean)
: Enables/disables IF automatic gain control (AGC)

V4L2_CID_RF_TUNER_RF_GAIN (integer)
: The RF amplifier is the very first amplifier on the receiver signal path, just right after the antenna input. The difference between the LNA gain and the RF gain in this document is that the LNA gain is integrated in the tuner chip while the RF gain is a separate chip. There may be both RF and LNA gain controls in the same device. The range and step are driver-specific.

V4L2_CID_RF_TUNER_LNA_GAIN (integer)
: LNA (low noise amplifier) gain is first gain stage on the RF tuner signal path. It is located very close to tuner antenna input. Used when `V4L2_CID_RF_TUNER_LNA_GAIN_AUTO` is not set. See

V4L2_CID_RF_TUNER_RF_GAIN to understand how RF gain and LNA gain differs from the each others. The range and step are driver-specific.

V4L2_CID_RF_TUNER_MIXER_GAIN (integer)
Mixer gain is second gain stage on the RF tuner signal path. It is located inside mixer block, where RF signal is down-converted by the mixer. Used when V4L2_CID_RF_TUNER_MIXER_GAIN_AUTO is not set. The range and step are driver-specific.

V4L2_CID_RF_TUNER_IF_GAIN (integer)
IF gain is last gain stage on the RF tuner signal path. It is located on output of RF tuner. It controls signal level of intermediate frequency output or baseband output. Used when V4L2_CID_RF_TUNER_IF_GAIN_AUTO is not set. The range and step are driver-specific.

V4L2_CID_RF_TUNER_PLL_LOCK (boolean)
Is synthesizer PLL locked? RF tuner is receiving given frequency when that control is set. This is a read-only control.

FM Transmitter Control Reference

The FM Transmitter (FM_TX) class includes controls for common features of FM transmissions capable devices. Currently this class includes parameters for audio compression, pilot tone generation, audio deviation limiter, RDS transmission and tuning power features.

FM_TX Control IDs

V4L2_CID_FM_TX_CLASS (class)
The FM_TX class descriptor. Calling *ioctls VIDIOC_QUERYCTRL, VIDIOC_QUERY_EXT_CTRL and VIDIOC_QUERYMENU* for this control will return a description of this control class.

V4L2_CID_RDS_TX_DEVIATION (integer)
Configures RDS signal frequency deviation level in Hz. The range and step are driver-specific.

V4L2_CID_RDS_TX_PI (integer)
Sets the RDS Programme Identification field for transmission.

V4L2_CID_RDS_TX_PTY (integer)
Sets the RDS Programme Type field for transmission. This encodes up to 31 pre-defined programme types.

V4L2_CID_RDS_TX_PS_NAME (string)
Sets the Programme Service name (PS_NAME) for transmission. It is intended for static display on a receiver. It is the primary aid to listeners in programme service identification and selection. In Annex E of *IEC 62106*, the RDS specification, there is a full description of the correct character encoding for Programme Service name strings. Also from RDS specification, PS is usually a single eight character text. However, it is also possible to find receivers which can scroll strings sized as 8 x N characters. So, this control

must be configured with steps of 8 characters. The result is it must always contain a string with size multiple of 8.

V4L2_CID_RDS_TX_RADIO_TEXT (string)
Sets the Radio Text info for transmission. It is a textual description of what is being broadcasted. RDS Radio Text can be applied when broadcaster wishes to transmit longer PS names, programme-related information or any other text. In these cases, RadioText should be used in addition to V4L2_CID_RDS_TX_PS_NAME. The encoding for Radio Text strings is also fully described in Annex E of *IEC 62106*. The length of Radio Text strings depends on which RDS Block is being used to transmit it, either 32 (2A block) or 64 (2B block). However, it is also possible to find receivers which can scroll strings sized as 32 x N or 64 x N characters. So, this control must be configured with steps of 32 or 64 characters. The result is it must always contain a string with size multiple of 32 or 64.

V4L2_CID_RDS_TX_MONO_STEREO (boolean)
Sets the Mono/Stereo bit of the Decoder Identification code. If set, then the audio was recorded as stereo.

V4L2_CID_RDS_TX_ARTIFICIAL_HEAD (boolean)
Sets the Artificial Head bit of the Decoder Identification code. If set, then the audio was recorded using an artificial head.

V4L2_CID_RDS_TX_COMPRESSED (boolean)
Sets the Compressed bit of the Decoder Identification code. If set, then the audio is compressed.

V4L2_CID_RDS_TX_DYNAMIC_PTY (boolean)
Sets the Dynamic PTY bit of the Decoder Identification code. If set, then the PTY code is dynamically switched.

V4L2_CID_RDS_TX_TRAFFIC_ANNOUNCEMENT (boolean)
If set, then a traffic announcement is in progress.

V4L2_CID_RDS_TX_TRAFFIC_PROGRAM (boolean)
If set, then the tuned programme carries traffic announcements.

V4L2_CID_RDS_TX_MUSIC_SPEECH (boolean)
If set, then this channel broadcasts music. If cleared, then it broadcasts speech. If the transmitter doesn't make this distinction, then it should be set.

V4L2_CID_RDS_TX_ALT_FREQS_ENABLE (boolean)
If set, then transmit alternate frequencies.

V4L2_CID_RDS_TX_ALT_FREQS (__u32 array)
The alternate frequencies in kHz units. The RDS standard allows for up to 25 frequencies to be defined. Drivers may support fewer frequencies so check the array size.

V4L2_CID_AUDIO_LIMITER_ENABLED (boolean)
Enables or disables the audio deviation limiter feature. The limiter is useful when trying to maximize the audio volume, minimize receiver-generated distortion and prevent overmodulation.

V4L2_CID_AUDIO_LIMITER_RELEASE_TIME (integer)

Sets the audio deviation limiter feature release time. Unit is in useconds. Step and range are driver-specific.

V4L2_CID_AUDIO_LIMITER_DEVIATION (integer)
Configures audio frequency deviation level in Hz. The range and step are driver-specific.

V4L2_CID_AUDIO_COMPRESSION_ENABLED (boolean)
Enables or disables the audio compression feature. This feature amplifies signals below the threshold by a fixed gain and compresses audio signals above the threshold by the ratio of Threshold/(Gain + Threshold).

V4L2_CID_AUDIO_COMPRESSION_GAIN (integer)
Sets the gain for audio compression feature. It is a dB value. The range and step are driver-specific.

V4L2_CID_AUDIO_COMPRESSION_THRESHOLD (integer)
Sets the threshold level for audio compression freature. It is a dB value. The range and step are driver-specific.

V4L2_CID_AUDIO_COMPRESSION_ATTACK_TIME (integer)
Sets the attack time for audio compression feature. It is a useconds value. The range and step are driver-specific.

V4L2_CID_AUDIO_COMPRESSION_RELEASE_TIME (integer)
Sets the release time for audio compression feature. It is a useconds value. The range and step are driver-specific.

V4L2_CID_PILOT_TONE_ENABLED (boolean)
Enables or disables the pilot tone generation feature.

V4L2_CID_PILOT_TONE_DEVIATION (integer)
Configures pilot tone frequency deviation level. Unit is in Hz. The range and step are driver-specific.

V4L2_CID_PILOT_TONE_FREQUENCY (integer)
Configures pilot tone frequency value. Unit is in Hz. The range and step are driver-specific.

V4L2_CID_TUNE_PREEMPHASIS
(enum)

enum v4l2_preemphasis -
Configures the pre-emphasis value for broadcasting. A pre-emphasis filter is applied to the broadcast to accentuate the high audio frequencies. Depending on the region, a time constant of either 50 or 75 useconds is used. The enum v4l2_preemphasis defines possible values for pre-emphasis. Here they are:

V4L2_PREEMPHASIS_DISABLED	No pre-emphasis is applied.
V4L2_PREEMPHASIS_50_uS	A pre-emphasis of 50 uS is used.
V4L2_PREEMPHASIS_75_uS	A pre-emphasis of 75 uS is used.

V4L2_CID_TUNE_POWER_LEVEL (integer)
Sets the output power level for signal transmission. Unit is in dBuV. Range and step are driver-specific.

V4L2_CID_TUNE_ANTENNA_CAPACITOR (integer)
This selects the value of antenna tuning capacitor manually or automatically if set to zero. Unit, range and step are driver-specific.

For more details about RDS specification, refer to *IEC 62106* document, from CENELEC.

FM Receiver Control Reference

The FM Receiver (FM_RX) class includes controls for common features of FM Reception capable devices.

FM_RX Control IDs

V4L2_CID_FM_RX_CLASS (class)
The FM_RX class descriptor. Calling *ioctls VIDIOC_QUERYCTRL, VIDIOC_QUERY_EXT_CTRL and VIDIOC_QUERYMENU* for this control will return a description of this control class.

V4L2_CID_RDS_RECEPTION (boolean)
Enables/disables RDS reception by the radio tuner

V4L2_CID_RDS_RX_PTY (integer)
Gets RDS Programme Type field. This encodes up to 31 pre-defined programme types.

V4L2_CID_RDS_RX_PS_NAME (string)
Gets the Programme Service name (PS_NAME). It is intended for static display on a receiver. It is the primary aid to listeners in programme service identification and selection. In Annex E of *IEC 62106*, the RDS specification, there is a full description of the correct character encoding for Programme Service name strings. Also from RDS specification, PS is usually a single eight character text. However, it is also possible to find receivers which can scroll strings sized as 8 x N characters. So, this control must be configured with steps of 8 characters. The result is it must always contain a string with size multiple of 8.

V4L2_CID_RDS_RX_RADIO_TEXT (string)
Gets the Radio Text info. It is a textual description of what is being broadcasted. RDS Radio Text can be applied when broadcaster wishes to transmit longer PS names, programme-related information or any other text. In these cases, RadioText can be used in addition to `V4L2_CID_RDS_RX_PS_NAME`. The encoding for Radio Text strings is also fully described in Annex E of *IEC 62106*. The length of Radio Text strings depends on which RDS Block is being used to transmit it, either 32 (2A block) or 64 (2B block). However, it is also possible to find receivers which can scroll strings sized as 32 x N or 64 x N characters. So, this control must be configured with steps of 32 or 64 characters. The result is it must always contain a string with size multiple of 32 or 64.

V4L2_CID_RDS_RX_TRAFFIC_ANNOUNCEMENT (boolean)
If set, then a traffic announcement is in progress.

V4L2_CID_RDS_RX_TRAFFIC_PROGRAM (boolean)
If set, then the tuned programme carries traffic announcements.

V4L2_CID_RDS_RX_MUSIC_SPEECH (boolean)
If set, then this channel broadcasts music. If cleared, then it broadcasts speech. If the transmitter doesn't make this distinction, then it will be set.

V4L2_CID_TUNE_DEEMPHASIS
(enum)

enum v4l2_deemphasis -
Configures the de-emphasis value for reception. A de-emphasis filter is applied to the broadcast to accentuate the high audio frequencies. Depending on the region, a time constant of either 50 or 75 useconds is used. The enum v4l2_deemphasis defines possible values for de-emphasis. Here they are:

V4L2_DEEMPHASIS_DISABLED	No de-emphasis is applied.
V4L2_DEEMPHASIS_50_uS	A de-emphasis of 50 uS is used.
V4L2_DEEMPHASIS_75_uS	A de-emphasis of 75 uS is used.

Detect Control Reference

The Detect class includes controls for common features of various motion or object detection capable devices.

Detect Control IDs

V4L2_CID_DETECT_CLASS (class)
The Detect class descriptor. Calling *ioctls VIDIOC_QUERYCTRL, VIDIOC_QUERY_EXT_CTRL and VIDIOC_QUERYMENU* for this control will return a description of this control class.

V4L2_CID_DETECT_MD_MODE (menu)
Sets the motion detection mode.

V4L2_DETECT_MD_MODE_DISABLED	Disable motion detection.
V4L2_DETECT_MD_MODE_GLOBAL	Use a single motion detection threshold.
V4L2_DETECT_MD_MODE_THRESHOLD_GRID	The image is divided into a grid, each cell with its own motion detection threshold. These thresholds are set through the V4L2_CID_DETECT_MD_THRESHOLD_GRID matrix control.
V4L2_DETECT_MD_MODE_REGION_GRID	The image is divided into a grid, each cell with its own region value that specifies which per-region motion detection thresholds should be used. Each region has its own thresholds. How these per-region thresholds are set up is driver-specific. The region values for the grid are set through the V4L2_CID_DETECT_MD_REGION_GRID matrix control.

V4L2_CID_DETECT_MD_GLOBAL_THRESHOLD (integer)
Sets the global motion detection threshold to be used with the V4L2_DETECT_MD_MODE_GLOBAL motion detection mode.

V4L2_CID_DETECT_MD_THRESHOLD_GRID (__u16 matrix)
Sets the motion detection thresholds for each cell in the grid. To be used with the V4L2_DETECT_MD_MODE_THRESHOLD_GRID motion detection mode. Matrix element (0, 0) represents the cell at the top-left of the grid.

V4L2_CID_DETECT_MD_REGION_GRID (__u8 matrix)
Sets the motion detection region value for each cell in the grid. To be used with the V4L2_DETECT_MD_MODE_REGION_GRID motion detection mode. Matrix element (0, 0) represents the cell at the top-left of the grid.

Guidelines for Video4Linux pixel format 4CCs

Guidelines for Video4Linux 4CC codes defined using v4l2_fourcc() are specified in this document. First of the characters defines the nature of the pixel format, compression and colour space. The interpretation of the other three characters depends on the first one.

Existing 4CCs may not obey these guidelines.

Raw bayer

The following first characters are used by raw bayer formats:

- B: raw bayer, uncompressed
- b: raw bayer, DPCM compressed
- a: A-law compressed
- u: u-law compressed

2nd character: pixel order

- B: BGGR
- G: GBRG
- g: GRBG
- R: RGGB

3rd character: uncompressed bits-per-pixel 0–9, A–

4th character: compressed bits-per-pixel 0–9, A–

Data Formats

Data Format Negotiation

Different devices exchange different kinds of data with applications, for example video images, raw or sliced VBI data, RDS datagrams. Even within one kind many different formats are possible, in particular there is an abundance of image formats. Although drivers must provide a default and the selection persists across closing and reopening a device, applications should always negotiate a data format before engaging in data exchange. Negotiation means the application asks for a particular format and the driver selects and reports the best the hardware can do to satisfy the request. Of course applications can also just query the current selection.

A single mechanism exists to negotiate all data formats using the aggregate struct `v4l2_format` and the *VIDIOC_G_FMT* and *VIDIOC_S_FMT* ioctls. Additionally the *VIDIOC_TRY_FMT* ioctl can be used to examine what the hardware *could* do, without actually selecting a new data format. The data formats supported by the V4L2 API are covered in the respective device section in *Interfaces*. For a closer look at image formats see *Image Formats*.

The *VIDIOC_S_FMT* ioctl is a major turning-point in the initialization sequence. Prior to this point multiple panel applications can access the same device concurrently to select the current input, change controls or modify other properties. The first *VIDIOC_S_FMT* assigns a logical stream (video data, VBI data etc.) exclusively to one file descriptor.

Exclusive means no other application, more precisely no other file descriptor, can grab this stream or change device properties inconsistent with the negotiated parameters. A video standard change for example, when the new standard uses a different number of scan lines, can invalidate the selected image format. Therefore only the file descriptor owning the stream can make invalidating changes. Accordingly multiple file descriptors which grabbed different logical streams prevent each other from interfering with their settings. When for example video overlay is about to start or already in progress, simultaneous video capturing may be restricted to the same cropping and image size.

When applications omit the *VIDIOC_S_FMT* ioctl its locking side effects are implied by the next step, the selection of an I/O method with the *ioctl VIDIOC_REQBUFS* ioctl or implicit with the first `read()` or `write()` call.

Generally only one logical stream can be assigned to a file descriptor, the exception being drivers permitting simultaneous video capturing and overlay using the same file descriptor for compatibility with V4L and earlier versions of V4L2. Switching the logical stream or returning into "panel mode" is possible by closing and reopening the device. Drivers *may* support a switch using *VIDIOC_S_FMT*.

All drivers exchanging data with applications must support the *VIDIOC_G_FMT* and *VIDIOC_S_FMT* ioctl. Implementation of the *VIDIOC_TRY_FMT* is highly recommended but optional.

Image Format Enumeration

Apart of the generic format negotiation functions a special ioctl to enumerate all image formats supported by video capture, overlay or output devices is available.[1]

The *ioctl VIDIOC_ENUM_FMT* ioctl must be supported by all drivers exchanging image data with applications.

Important: Drivers are not supposed to convert image formats in kernel space. They must enumerate only formats directly supported by the hardware. If necessary driver writers should publish an example conversion routine or library for integration into applications.

Single- and multi-planar APIs

Some devices require data for each input or output video frame to be placed in discontiguous memory buffers. In such cases, one video frame has to be addressed using more than one memory address, i.e. one pointer per "plane". A plane is a sub-buffer of the current frame. For examples of such formats see *Image Formats*.

Initially, V4L2 API did not support multi-planar buffers and a set of extensions has been introduced to handle them. Those extensions constitute what is being referred to as the "multi-planar API".

Some of the V4L2 API calls and structures are interpreted differently, depending on whether single- or multi-planar API is being used. An application can choose whether to use one or the other by passing a corresponding buffer type to its ioctl calls. Multi-planar versions of buffer types are suffixed with an `_MPLANE` string. For a list of available multi-planar buffer types see enum `v4l2_buf_type`.

Multi-planar formats

Multi-planar API introduces new multi-planar formats. Those formats use a separate set of FourCC codes. It is important to distinguish between the multi-planar API and a multi-planar format. Multi-planar API calls can handle all single-planar formats as well (as long as they are passed in multi-planar API structures), while the single-planar API cannot handle multi-planar formats.

[1] Enumerating formats an application has no a-priori knowledge of (otherwise it could explicitly ask for them and need not enumerate) seems useless, but there are applications serving as proxy between drivers and the actual video applications for which this is useful.

Calls that distinguish between single and multi-planar APIs

VIDIOC_QUERYCAP
: Two additional multi-planar capabilities are added. They can be set together with non-multi-planar ones for devices that handle both single- and multi-planar formats.

VIDIOC_G_FMT, *VIDIOC_S_FMT*, *VIDIOC_TRY_FMT*
: New structures for describing multi-planar formats are added: struct `v4l2_pix_format_mplane` and struct `v4l2_plane_pix_format`. Drivers may define new multi-planar formats, which have distinct FourCC codes from the existing single-planar ones.

VIDIOC_QBUF, *VIDIOC_DQBUF*, *VIDIOC_QUERYBUF*
: A new struct `v4l2_plane` structure for describing planes is added. Arrays of this structure are passed in the new `m.planes` field of struct `v4l2_buffer`.

VIDIOC_REQBUFS
: Will allocate multi-planar buffers as requested.

Cropping, composing and scaling – the SELECTION API

Introduction

Some video capture devices can sample a subsection of a picture and shrink or enlarge it to an image of arbitrary size. Next, the devices can insert the image into larger one. Some video output devices can crop part of an input image, scale it up or down and insert it at an arbitrary scan line and horizontal offset into a video signal. We call these abilities cropping, scaling and composing.

On a video *capture* device the source is a video signal, and the cropping target determine the area actually sampled. The sink is an image stored in a memory buffer. The composing area specifies which part of the buffer is actually written to by the hardware.

On a video *output* device the source is an image in a memory buffer, and the cropping target is a part of an image to be shown on a display. The sink is the display or the graphics screen. The application may select the part of display where the image should be displayed. The size and position of such a window is controlled by the compose target.

Rectangles for all cropping and composing targets are defined even if the device does supports neither cropping nor composing. Their size and position will be fixed in such a case. If the device does not support scaling then the cropping and composing rectangles have the same size.

Selection targets

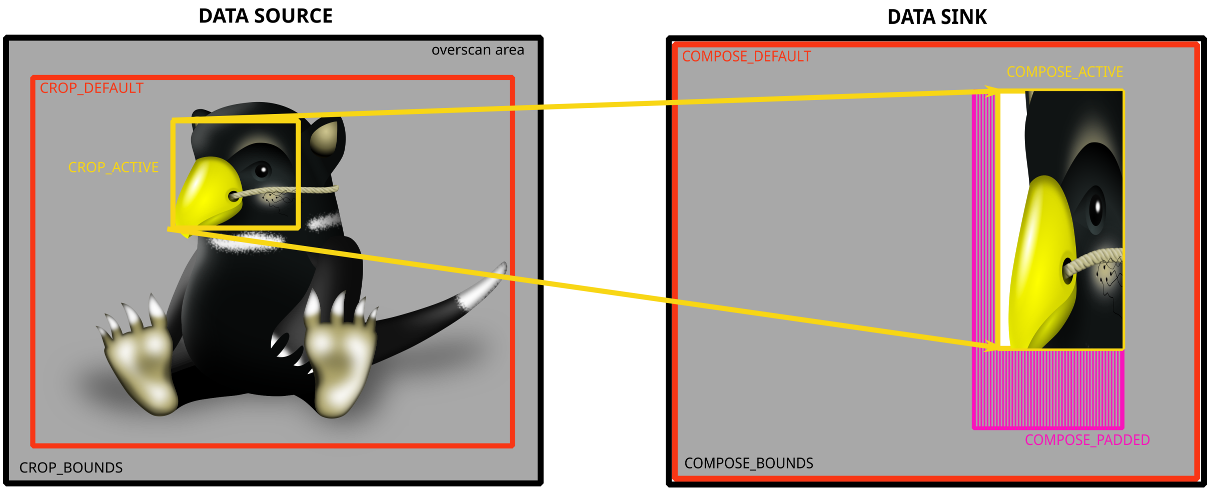

Fig. 2: Cropping and composing targets
Targets used by a cropping, composing and scaling process

See *Selection targets* for more information.

Configuration

Applications can use the *selection API* to select an area in a video signal or a buffer, and to query for default settings and hardware limits.

Video hardware can have various cropping, composing and scaling limitations. It may only scale up or down, support only discrete scaling factors, or have different scaling abilities in the horizontal and vertical directions. Also it may not support scaling at all. At the same time the cropping/composing rectangles may have to be aligned, and both the source and the sink may have arbitrary upper and lower size limits. Therefore, as usual, drivers are expected to adjust the requested parameters and return the actual values selected. An application can control the rounding behaviour using *constraint flags*.

Configuration of video capture

See figure *Cropping and composing targets* for examples of the selection targets available for a video capture device. It is recommended to configure the cropping targets before to the composing targets.

The range of coordinates of the top left corner, width and height of areas that can be sampled is given by the `V4L2_SEL_TGT_CROP_BOUNDS` target. It is recommended for the driver developers to put the top/left corner at position `(0,0)`. The rectangle's coordinates are expressed in pixels.

The top left corner, width and height of the source rectangle, that is the area actually sampled, is given by the `V4L2_SEL_TGT_CROP` target. It uses the same coordinate system as `V4L2_SEL_TGT_CROP_BOUNDS`. The active cropping area must lie completely inside the capture boundaries. The driver may further adjust the requested size and/or position according to hardware limitations.

Each capture device has a default source rectangle, given by the V4L2_SEL_TGT_CROP_DEFAULT target. This rectangle shall cover what the driver writer considers the complete picture. Drivers shall set the active crop rectangle to the default when the driver is first loaded, but not later.

The composing targets refer to a memory buffer. The limits of composing coordinates are obtained using V4L2_SEL_TGT_COMPOSE_BOUNDS. All coordinates are expressed in pixels. The rectangle's top/left corner must be located at position (0,0). The width and height are equal to the image size set by *VIDIOC_S_FMT*.

The part of a buffer into which the image is inserted by the hardware is controlled by the V4L2_SEL_TGT_COMPOSE target. The rectangle's coordinates are also expressed in the same coordinate system as the bounds rectangle. The composing rectangle must lie completely inside bounds rectangle. The driver must adjust the composing rectangle to fit to the bounding limits. Moreover, the driver can perform other adjustments according to hardware limitations. The application can control rounding behaviour using *constraint flags*.

For capture devices the default composing rectangle is queried using V4L2_SEL_TGT_COMPOSE_DEFAULT. It is usually equal to the bounding rectangle.

The part of a buffer that is modified by the hardware is given by V4L2_SEL_TGT_COMPOSE_PADDED. It contains all pixels defined using V4L2_SEL_TGT_COMPOSE plus all padding data modified by hardware during insertion process. All pixels outside this rectangle *must not* be changed by the hardware. The content of pixels that lie inside the padded area but outside active area is undefined. The application can use the padded and active rectangles to detect where the rubbish pixels are located and remove them if needed.

Configuration of video output

For output devices targets and ioctls are used similarly to the video capture case. The *composing* rectangle refers to the insertion of an image into a video signal. The cropping rectangles refer to a memory buffer. It is recommended to configure the composing targets before to the cropping targets.

The cropping targets refer to the memory buffer that contains an image to be inserted into a video signal or graphical screen. The limits of cropping coordinates are obtained using V4L2_SEL_TGT_CROP_BOUNDS. All coordinates are expressed in pixels. The top/left corner is always point (0,0). The width and height is equal to the image size specified using *VIDIOC_S_FMT* ioctl.

The top left corner, width and height of the source rectangle, that is the area from which image date are processed by the hardware, is given by the V4L2_SEL_TGT_CROP. Its coordinates are expressed in in the same coordinate system as the bounds rectangle. The active cropping area must lie completely inside the crop boundaries and the driver may further adjust the requested size and/or position according to hardware limitations.

For output devices the default cropping rectangle is queried using V4L2_SEL_TGT_CROP_DEFAULT. It is usually equal to the bounding rectangle.

The part of a video signal or graphics display where the image is inserted by the hardware is controlled by V4L2_SEL_TGT_COMPOSE target. The rectangle's coordinates are expressed in pixels. The composing rectangle must lie completely inside the bounds rectangle. The driver must adjust the area to fit to the bounding limits. Moreover, the driver can perform other adjustments according to hardware limitations.

The device has a default composing rectangle, given by the V4L2_SEL_TGT_COMPOSE_DEFAULT target. This rectangle shall cover what the driver writer considers the complete picture. It is recommended for the driver developers to put the top/left corner at position (0,0). Drivers shall set the active composing rectangle to the default one when the driver is first loaded.

The devices may introduce additional content to video signal other than an image from memory buffers. It includes borders around an image. However, such a padded area is driver-dependent feature not covered by this document. Driver developers are encouraged to keep padded rectangle equal to active one. The padded target is accessed by the V4L2_SEL_TGT_COMPOSE_PADDED identifier. It must contain all pixels from the V4L2_SEL_TGT_COMPOSE target.

Scaling control

An application can detect if scaling is performed by comparing the width and the height of rectangles obtained using V4L2_SEL_TGT_CROP and V4L2_SEL_TGT_COMPOSE targets. If these are not equal then the scaling is applied. The application can compute the scaling ratios using these values.

Comparison with old cropping API

The selection API was introduced to cope with deficiencies of the older *CROP API*, that was designed to control simple capture devices. Later the cropping API was adopted by video output drivers. The ioctls are used to select a part of the display were the video signal is inserted. It should be considered as an API abuse because the described operation is actually the composing. The selection API makes a clear distinction between composing and cropping operations by setting the appropriate targets.

The CROP API lacks any support for composing to and cropping from an image inside a memory buffer. The application could configure a capture device to fill only a part of an image by abusing V4L2 API. Cropping a smaller image from a larger one is achieved by setting the field bytesperline at struct *v4l2_pix_format*. Introducing an image offsets could be done by modifying field m_userptr at struct v4l2_buffer before calling *VIDIOC_QBUF*. Those operations should be avoided because they are not portable (endianness), and do not work for macroblock and Bayer formats and mmap buffers.

The selection API deals with configuration of buffer cropping/composing in a clear, intuitive and portable way. Next, with the selection API the concepts of the padded target and constraints flags are introduced. Finally, struct v4l2_crop and struct v4l2_cropcap have no reserved fields. Therefore there is no way to extend their functionality. The new struct v4l2_selection provides a lot of place for future extensions.

Driver developers are encouraged to implement only selection API. The former cropping API would be simulated using the new one.

Examples

(A video capture device is assumed; change V4L2_BUF_TYPE_VIDEO_CAPTURE for other devices; change target to V4L2_SEL_TGT_COMPOSE_* family to configure composing area)

Example: Resetting the cropping parameters

```
struct v4l2_selection sel = {
    .type = V4L2_BUF_TYPE_VIDEO_CAPTURE,
    .target = V4L2_SEL_TGT_CROP_DEFAULT,
};
ret = ioctl(fd, VIDIOC_G_SELECTION, &sel);
if (ret)
    exit(-1);
sel.target = V4L2_SEL_TGT_CROP;
ret = ioctl(fd, VIDIOC_S_SELECTION, &sel);
if (ret)
    exit(-1);
```

Setting a composing area on output of size of *at most* half of limit placed at a center of a display.

Example: Simple downscaling

```
struct v4l2_selection sel = {
    .type = V4L2_BUF_TYPE_VIDEO_OUTPUT,
    .target = V4L2_SEL_TGT_COMPOSE_BOUNDS,
};
struct v4l2_rect r;

ret = ioctl(fd, VIDIOC_G_SELECTION, &sel);
if (ret)
    exit(-1);
/* setting smaller compose rectangle */
r.width = sel.r.width / 2;
r.height = sel.r.height / 2;
r.left = sel.r.width / 4;
r.top = sel.r.height / 4;
sel.r = r;
sel.target = V4L2_SEL_TGT_COMPOSE;
sel.flags = V4L2_SEL_FLAG_LE;
ret = ioctl(fd, VIDIOC_S_SELECTION, &sel);
if (ret)
    exit(-1);
```

A video output device is assumed; change V4L2_BUF_TYPE_VIDEO_OUTPUT for other devices

Example: Querying for scaling factors

```
struct v4l2_selection compose = {
    .type = V4L2_BUF_TYPE_VIDEO_OUTPUT,
    .target = V4L2_SEL_TGT_COMPOSE,
};
struct v4l2_selection crop = {
    .type = V4L2_BUF_TYPE_VIDEO_OUTPUT,
    .target = V4L2_SEL_TGT_CROP,
};
double hscale, vscale;

ret = ioctl(fd, VIDIOC_G_SELECTION, &compose);
if (ret)
    exit(-1);
ret = ioctl(fd, VIDIOC_G_SELECTION, &crop);
if (ret)
    exit(-1);

/* computing scaling factors */
hscale = (double)compose.r.width / crop.r.width;
vscale = (double)compose.r.height / crop.r.height;
```

Image Cropping, Insertion and Scaling – the CROP API

Note: The CROP API is mostly superseded by the newer *SELECTION API*. The new API should be preferred in most cases, with the exception of pixel aspect ratio detection, which is implemented by *VIDIOC_CROPCAP* and has no equivalent in the SELECTION API. See *Comparison with old cropping API* for a comparison of the two APIs.

Some video capture devices can sample a subsection of the picture and shrink or enlarge it to an image of arbitrary size. We call these abilities cropping and scaling. Some video output devices can scale an image up or down and insert it at an arbitrary scan line and horizontal offset into a video signal.

Applications can use the following API to select an area in the video signal, query the default area and the hardware limits.

Note: Despite their name, the *VIDIOC_CROPCAP*, *VIDIOC_G_CROP* and *VIDIOC_S_CROP* ioctls apply to input as well as output devices.

Scaling requires a source and a target. On a video capture or overlay device the source is the video signal, and the cropping ioctls determine the area actually sampled. The target are images read by the application or overlaid onto the graphics screen. Their size (and position for an overlay) is negotiated with the *VIDIOC_G_FMT* and *VIDIOC_S_FMT* ioctls.

On a video output device the source are the images passed in by the application, and their size is again negotiated with the *VIDIOC_G_FMT* and *VIDIOC_S_FMT* ioctls, or may be encoded in a compressed video stream. The target is the video signal, and the cropping ioctls determine the area where the images are inserted.

Source and target rectangles are defined even if the device does not support scaling or the *VIDIOC_G_CROP* and *VIDIOC_S_CROP* ioctls. Their size (and position where applicable) will be fixed in this case.

Note: All capture and output devices that support the CROP or SELECTION API will also support the *VIDIOC_CROPCAP* ioctl.

Cropping Structures

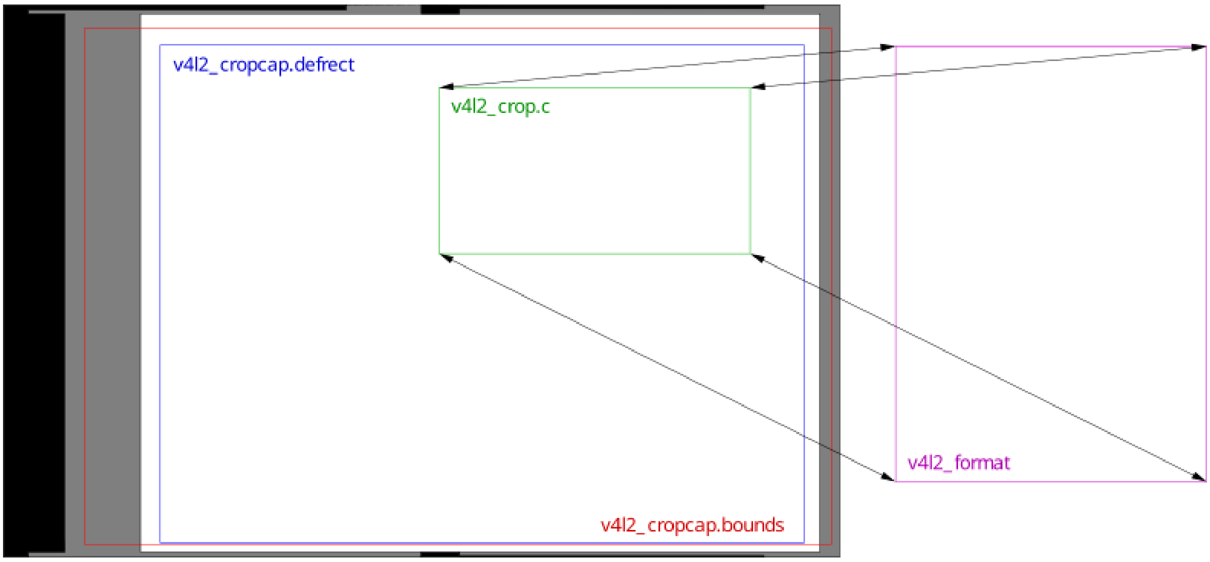

Fig. 3: Image Cropping, Insertion and Scaling
The cropping, insertion and scaling process

For capture devices the coordinates of the top left corner, width and height of the area which can be sampled is given by the `bounds` substructure of the struct `v4l2_cropcap` returned by the *VIDIOC_CROPCAP* ioctl. To support a wide range of hardware this specification does not define an origin or units. However by convention drivers should horizontally count unscaled samples relative to 0H (the leading edge of the horizontal sync pulse, see *Figure 4.1. Line synchronization*). Vertically ITU-R line numbers of the first field (see ITU R-525 line numbering for *525 lines* and for *625 lines*), multiplied by two if the driver can capture both fields.

The top left corner, width and height of the source rectangle, that is the area actually sampled, is given by struct `v4l2_crop` using the same coordinate system as struct `v4l2_cropcap`. Applications can use the *VIDIOC_G_CROP* and *VIDIOC_S_CROP* ioctls to get and set this rectangle. It must lie completely within the capture boundaries and the driver may further adjust the requested size and/or position according to hardware limitations.

Each capture device has a default source rectangle, given by the `defrect` substructure of struct `v4l2_cropcap`. The center of this rectangle shall align with the center of the active picture area of the video signal, and cover what the driver writer considers the complete picture. Drivers shall reset the source rectangle to the default when the driver is first loaded, but not later.

For output devices these structures and ioctls are used accordingly, defining the *target* rectangle where the images will be inserted into the video signal.

Scaling Adjustments

Video hardware can have various cropping, insertion and scaling limitations. It may only scale up or down, support only discrete scaling factors, or have different scaling abilities in horizontal and vertical direction. Also it may not support scaling at all. At the same time the struct `v4l2_crop` rectangle may have to be aligned, and both the source and target rectangles may have arbitrary upper and lower size limits. In particular the maximum `width` and `height` in struct `v4l2_crop` may be smaller than the struct `v4l2_cropcap`. `bounds` area. Therefore, as usual, drivers are expected to adjust the requested parameters and return the actual values selected.

Applications can change the source or the target rectangle first, as they may prefer a particular image size or a certain area in the video signal. If the driver has to adjust both to satisfy hardware limitations, the last requested rectangle shall take priority, and the driver should preferably adjust the opposite one. The *VIDIOC_TRY_FMT* ioctl however shall not change the driver state and therefore only adjust the requested rectangle.

Suppose scaling on a video capture device is restricted to a factor 1:1 or 2:1 in either direction and the target image size must be a multiple of 16 × 16 pixels. The source cropping rectangle is set to defaults, which are also the upper limit in this example, of 640 × 400 pixels at offset 0, 0. An application requests an image size of 300 × 225 pixels, assuming video will be scaled down from the "full picture" accordingly. The driver sets the image size to the closest possible values 304 × 224, then chooses the cropping rectangle closest to the requested size, that is 608 × 224 (224 × 2:1 would exceed the limit 400). The offset 0, 0 is still valid, thus unmodified. Given the default cropping rectangle reported by *VIDIOC_CROPCAP* the application can easily propose another offset to center the cropping rectangle.

Now the application may insist on covering an area using a picture aspect ratio closer to the original request, so it asks for a cropping rectangle of 608 × 456 pixels. The present scaling factors limit cropping to 640 × 384, so the driver returns the cropping size 608 × 384 and adjusts the image size to closest possible 304 × 192.

Examples

Source and target rectangles shall remain unchanged across closing and reopening a device, such that piping data into or out of a device will work without special preparations. More advanced applications should ensure the parameters are suitable before starting I/O.

Note: On the next two examples, a video capture device is assumed; change V4L2_BUF_TYPE_VIDEO_CAPTURE for other types of device.

Example: Resetting the cropping parameters

```
struct v4l2_cropcap cropcap;
struct v4l2_crop crop;

memset (&cropcap, 0, sizeof (cropcap));
cropcap.type = V4L2_BUF_TYPE_VIDEO_CAPTURE;

if (-1 == ioctl (fd, VIDIOC_CROPCAP, &cropcap)) {
    perror ("VIDIOC_CROPCAP");
    exit (EXIT_FAILURE);
}

memset (&crop, 0, sizeof (crop));
crop.type = V4L2_BUF_TYPE_VIDEO_CAPTURE;
crop.c = cropcap.defrect;

/* Ignore if cropping is not supported (EINVAL). */

if (-1 == ioctl (fd, VIDIOC_S_CROP, &crop)
    && errno != EINVAL) {
    perror ("VIDIOC_S_CROP");
    exit (EXIT_FAILURE);
}
```

Example: Simple downscaling

```
struct v4l2_cropcap cropcap;
struct v4l2_format format;

reset_cropping_parameters ();

/* Scale down to 1/4 size of full picture. */

memset (&format, 0, sizeof (format)); /* defaults */
```

(continues on next page)

(continued from previous page)

```
format.type = V4L2_BUF_TYPE_VIDEO_CAPTURE;

format.fmt.pix.width = cropcap.defrect.width >> 1;
format.fmt.pix.height = cropcap.defrect.height >> 1;
format.fmt.pix.pixelformat = V4L2_PIX_FMT_YUYV;

if (-1 == ioctl (fd, VIDIOC_S_FMT, &format)) {
    perror ("VIDIOC_S_FORMAT");
    exit (EXIT_FAILURE);
}

/* We could check the actual image size now, the actual scaling␣
↪factor
   or if the driver can scale at all. */
```

Example: Selecting an output area

Note: This example assumes an output device.

```
struct v4l2_cropcap cropcap;
struct v4l2_crop crop;

memset (&cropcap, 0, sizeof (cropcap));
cropcap.type = V4L2_BUF_TYPE_VIDEO_OUTPUT;

if (-1 == ioctl (fd, VIDIOC_CROPCAP;, &cropcap)) {
    perror ("VIDIOC_CROPCAP");
    exit (EXIT_FAILURE);
}

memset (&crop, 0, sizeof (crop));

crop.type = V4L2_BUF_TYPE_VIDEO_OUTPUT;
crop.c = cropcap.defrect;

/* Scale the width and height to 50 % of their original size
   and center the output. */

crop.c.width /= 2;
crop.c.height /= 2;
crop.c.left += crop.c.width / 2;
crop.c.top += crop.c.height / 2;

/* Ignore if cropping is not supported (EINVAL). */

if (-1 == ioctl (fd, VIDIOC_S_CROP, &crop)
```

(continues on next page)

(continued from previous page)

```
    && errno != EINVAL) {
    perror ("VIDIOC_S_CROP");
    exit (EXIT_FAILURE);
}
```

Example: Current scaling factor and pixel aspect

Note: This example assumes a video capture device.

```
struct v4l2_cropcap cropcap;
struct v4l2_crop crop;
struct v4l2_format format;
double hscale, vscale;
double aspect;
int dwidth, dheight;

memset (&cropcap, 0, sizeof (cropcap));
cropcap.type = V4L2_BUF_TYPE_VIDEO_CAPTURE;

if (-1 == ioctl (fd, VIDIOC_CROPCAP, &cropcap)) {
    perror ("VIDIOC_CROPCAP");
    exit (EXIT_FAILURE);
}

memset (&crop, 0, sizeof (crop));
crop.type = V4L2_BUF_TYPE_VIDEO_CAPTURE;

if (-1 == ioctl (fd, VIDIOC_G_CROP, &crop)) {
    if (errno != EINVAL) {
        perror ("VIDIOC_G_CROP");
        exit (EXIT_FAILURE);
    }

    /* Cropping not supported. */
    crop.c = cropcap.defrect;
}

memset (&format, 0, sizeof (format));
format.fmt.type = V4L2_BUF_TYPE_VIDEO_CAPTURE;

if (-1 == ioctl (fd, VIDIOC_G_FMT, &format)) {
    perror ("VIDIOC_G_FMT");
    exit (EXIT_FAILURE);
}

/* The scaling applied by the driver. */
```

(continues on next page)

(continued from previous page)

```
hscale = format.fmt.pix.width / (double) crop.c.width;
vscale = format.fmt.pix.height / (double) crop.c.height;

aspect = cropcap.pixelaspect.numerator /
     (double) cropcap.pixelaspect.denominator;
aspect = aspect * hscale / vscale;

/* Devices following ITU-R BT.601 do not capture
   square pixels. For playback on a computer monitor
   we should scale the images to this size. */

dwidth = format.fmt.pix.width / aspect;
dheight = format.fmt.pix.height;
```

Streaming Parameters

Streaming parameters are intended to optimize the video capture process as well as I/O. Presently applications can request a high quality capture mode with the *VIDIOC_S_PARM* ioctl.

The current video standard determines a nominal number of frames per second. If less than this number of frames is to be captured or output, applications can request frame skipping or duplicating on the driver side. This is especially useful when using the `read()` or `write()`, which are not augmented by timestamps or sequence counters, and to avoid unnecessary data copying.

Finally these ioctls can be used to determine the number of buffers used internally by a driver in read/write mode. For implications see the section discussing the `read()` function.

To get and set the streaming parameters applications call the *VIDIOC_G_PARM* and *VIDIOC_S_PARM* ioctl, respectively. They take a pointer to a struct `v4l2_streamparm`, which contains a union holding separate parameters for input and output devices.

These ioctls are optional, drivers need not implement them. If so, they return the `EINVAL` error code.

8.2.2 Image Formats

The V4L2 API was primarily designed for devices exchanging image data with applications. The struct `v4l2_pix_format` and struct `v4l2_pix_format_mplane` structures define the format and layout of an image in memory. The former is used with the single-planar API, while the latter is used with the multi-planar version (see *Single- and multi-planar APIs*). Image formats are negotiated with the *VIDIOC_S_FMT* ioctl. (The explanations here focus on video capturing and output, for overlay frame buffer formats see also *VIDIOC_G_FBUF*.)

Single-planar format structure

type **v4l2_pix_format**

Table 43: struct v4l2_pix_format

__u32	width	Image width in pixels.
__u32	height	Image height in pixels. If field is one of V4L2_FIELD_TOP, V4L2_FIELD_BOTTOM or V4L2_FIELD_ALTERNATE then height refers to the number of lines in the field, otherwise it refers to the number of lines in the frame (which is twice the field height for interlaced formats).
Applications set these fields to request an image size, drivers return the closest possible values. In case of planar formats the width and height applies to the largest plane. To avoid ambiguities drivers must return values rounded up to a multiple of the scale factor of any smaller planes. For example when the image format is YUV 4:2:0, width and height must be multiples of two. For compressed formats that contain the resolution information encoded inside the stream, when fed to a stateful mem2mem decoder, the fields may be zero to rely on the decoder to detect the right values. For more details see *Memory-to-Memory Stateful Video Decoder Interface* and format descriptions. For compressed formats on the CAPTURE side of a stateful mem2mem encoder, the fields must be zero, since the coded size is expected to be calculated internally by the encoder itself, based on the OUTPUT side. For more details see *Memory-to-Memory Stateful Video Encoder Interface* and format descriptions.		
__u32	pixelformat	The pixel format or type of compression, set by the application. This is a little endian *four character code*. V4L2 defines standard RGB formats in *RGB Formats*, YUV formats in *YUV Formats*, and reserved codes in *Reserved Image Formats*
__u32	field	Field order, from enum *v4l2_field*. Video images are typically interlaced. Applications can request to capture or output only the top or bottom field, or both fields interlaced or sequentially stored in one buffer or alternating in separate buffers. Drivers return the actual field order selected. For more details on fields see *Field Order*.
__u32	bytesperline	Distance in bytes between the leftmost pixels in two adjacent lines.

continues on next page

Table 43 – continued from previous page

Both applications and drivers can set this field to request padding bytes at the end of each line. Drivers however may ignore the value requested by the application, returning `width` times bytes per pixel or a larger value required by the hardware. That implies applications can just set this field to zero to get a reasonable default. Video hardware may access padding bytes, therefore they must reside in accessible memory. Consider cases where padding bytes after the last line of an image cross a system page boundary. Input devices may write padding bytes, the value is undefined. Output devices ignore the contents of padding bytes. When the image format is planar the `bytesperline` value applies to the first plane and is divided by the same factor as the `width` field for the other planes. For example the Cb and Cr planes of a YUV 4:2:0 image have half as many padding bytes following each line as the Y plane. To avoid ambiguities drivers must return a `bytesperline` value rounded up to a multiple of the scale factor. For compressed formats the `bytesperline` value makes no sense. Applications and drivers must set this to 0 in that case.		
`__u32`	`sizeimage`	Size in bytes of the buffer to hold a complete image, set by the driver. Usually this is `bytesperline` times `height`. When the image consists of variable length compressed data this is the number of bytes required by the codec to support the worst-case compression scenario. The driver will set the value for uncompressed images. Clients are allowed to set the sizeimage field for variable length compressed data flagged with `V4L2_FMT_FLAG_COMPRESSED` at *ioctl VIDIOC_ENUM_FMT*, but the driver may ignore it and set the value itself, or it may modify the provided value based on alignment requirements or minimum/maximum size requirements. If the client wants to leave this to the driver, then it should set sizeimage to 0.
`__u32`	`colorspace`	Image colorspace, from enum *`v4l2_colorspace`*. This information supplements the `pixelformat` and must be set by the driver for capture streams and by the application for output streams, see *Colorspaces*. If the application sets the flag `V4L2_PIX_FMT_FLAG_SET_CSC` then the application can set this field for a capture stream to request a specific colorspace for the captured image data. If the driver cannot handle requested conversion, it will return another supported colorspace. The driver indicates that colorspace conversion is supported by setting the flag V4L2_FMT_FLAG_CSC_COLORSPACE in the corresponding struct `v4l2_fmtdesc` during enumeration. See *Image Format Description Flags*.

continues on next page

Table 43 – continued from previous page

__u32	priv	This field indicates whether the remaining fields of the struct *v4l2_pix_format*, also called the extended fields are valid. When set to V4L2_PIX_FMT_PRIV_MAGIC, it indicates that the extended fields have been correctly initialized. When set to any other value it indicates that the extended fields contain undefined values. Applications that wish to use the pixel format extended fields must first ensure that the feature is supported by querying the device for the *V4L2_CAP_EXT_PIX_FORMAT* capability. If the capability isn't set the pixel format extended fields are not supported and using the extended fields will lead to undefined results. To use the extended fields, applications must set the priv field to V4L2_PIX_FMT_PRIV_MAGIC, initialize all the extended fields and zero the unused bytes of the struct v4l2_format raw_data field. When the priv field isn't set to V4L2_PIX_FMT_PRIV_MAGIC drivers must act as if all the extended fields were set to zero. On return drivers must set the priv field to V4L2_PIX_FMT_PRIV_MAGIC and all the extended fields to applicable values.
__u32	flags	Flags set by the application or driver, see *Format Flags*.
union {	(anonymous)	
__u32	ycbcr_enc	Y'CbCr encoding, from enum *v4l2_ycbcr_encoding*. This information supplements the colorspace and must be set by the driver for capture streams and by the application for output streams, see *Colorspaces*. If the application sets the flag V4L2_PIX_FMT_FLAG_SET_CSC then the application can set this field for a capture stream to request a specific Y'CbCr encoding for the captured image data. If the driver cannot handle requested conversion, it will return another supported encoding. This field is ignored for HSV pixelformats. The driver indicates that ycbcr_enc conversion is supported by setting the flag V4L2_FMT_FLAG_CSC_YCBCR_ENC in the corresponding struct v4l2_fmtdesc during enumeration. See *Image Format Description Flags*.

continues on next page

Table 43 – continued from previous page

__u32	hsv_enc	HSV encoding, from enum *v4l2_hsv_encoding*. This in formation supplements the colorspace and must be se by the driver for capture streams and by the applicatior for output streams, see *Colorspaces*. If the applicatior sets the flag V4L2_PIX_FMT_FLAG_SET_CSC then the ap plication can set this field for a capture stream to re quest a specific HSV encoding for the captured image data. If the driver cannot handle requested conversion it will return another supported encoding. This field is ignored for non-HSV pixelformats. The driver indicates that hsv_enc conversion is supported by setting the flag V4L2_FMT_FLAG_CSC_HSV_ENC in the corresponding struct v4l2_fmtdesc during enumeration. See *Image Format Description Flags*.
}		
__u32	quantizatior	Quantization range, from enum *v4l2_quantization* This information supplements the colorspace and mus be set by the driver for capture streams and by the ap plication for output streams, see *Colorspaces*. If the application sets the flag V4L2_PIX_FMT_FLAG_SET_CS then the application can set this field for a capture stream to request a specific quantization range fo the captured image data. If the driver cannot han dle requested conversion, it will return another sup ported quantization. The driver indicates that quan tization conversion is supported by setting the flag V4L2_FMT_FLAG_CSC_QUANTIZATION in the corre sponding struct v4l2_fmtdesc during enumeration. See *Image Format Description Flags*.
__u32	xfer_func	Transfer function, from enum *v4l2_xfer_func*. This in formation supplements the colorspace and must be se by the driver for capture streams and by the applicatior for output streams, see *Colorspaces*. If the applicatior sets the flag V4L2_PIX_FMT_FLAG_SET_CSC then the ap plication can set this field for a capture stream to reques a specific transfer function for the captured image data If the driver cannot handle requested conversion, it wil return another supported transfer function. The driver indicates that xfer_func conversion is supported by set ting the flag V4L2_FMT_FLAG_CSC_XFER_FUNC in the corresponding struct v4l2_fmtdesc during enumeration See *Image Format Description Flags*.

Table 44: Format Flags

V4L2_PIX_FMT_FLAG_PREMUL_ALPHA	0x00000001	The color values are premultiplied by the alpha channel value. For example, if a ligh blue pixel with 50% transparency was de scribed by RGBA values (128, 192, 255, 128) the same pixel described with premultiplied colors would be described by RGBA values (64, 96, 128, 128)
V4L2_PIX_FMT_FLAG_SET_CSC	0x00000002	Set by the application. It is only used fo capture and is ignored for output streams. I set, then request the device to do colorspace conversion from the received colorspace to the requested colorspace values. If the colorimetry field (`colorspace`, `xfer_func` `ycbcr_enc`, `hsv_enc` or `quantization` is set to `*_DEFAULT`, then that colorime try setting will remain unchanged from what was received. So in order to change the quantization, only the `quantization` field shall be set to non default value (`V4L2_QUANTIZATION_FULL_RANGE` o `V4L2_QUANTIZATION_LIM_RANGE`) and al other colorimetry fields shall be set to `*_DEFAULT`. To check which conversions are supported by the hardware for the current pixel format see *Image Format Description Flags*.

Multi-planar format structures

The struct *v4l2_plane_pix_format* structures define size and layout for each of the planes in a multi-planar format. The struct *v4l2_pix_format_mplane* structure contains information common to all planes (such as image width and height) and an array of struct *v4l2_plane_pix_format* structures, describing all planes of that format.

type **v4l2_plane_pix_format**

Table 45: struct v4l2_plane_pix_format

__u32	sizeimage	Maximum size in bytes required for image data in this plane, set by the driver. When the image consists of variable length compressed data this is the number of bytes required by the codec to support the worst-case compression scenario. The driver will set the value for uncompressed images. Clients are allowed to set the sizeimage field for variable length compressed data flagged with `V4L2_FMT_FLAG_COMPRESSED` at *ioctl VIDIOC_ENUM_FMT*, but the driver may ignore it and set the value itself, or it may modify the provided value based on alignment requirements or minimum/maximum size requirements. If the client wants to leave this to the driver, then it should set sizeimage to 0.
__u32	bytesperline	Distance in bytes between the leftmost pixels in two adjacent lines. See struct *v4l2_pix_format*.
__u16	reserved[6]	Reserved for future extensions. Should be zeroed by drivers and applications.

type **v4l2_pix_format_mplane**

Table 46: struct v4l2_pix_format_mplane

__u32	width	Image width in pixels. See struc *v4l2_pix_format*.
__u32	height	Image height in pixels. See struc *v4l2_pix_format*.
__u32	pixelformat	The pixel format. Both single- and multi planar four character codes can be used.
__u32	field	Field order, from enum *v4l2_field*. See struct *v4l2_pix_format*.
__u32	colorspace	Colorspace encoding, from enum *v4l2_colorspace*. See struc *v4l2_pix_format*.
struct *v4l2_plane_pix_format*	plane_fmt[VIDEO_MAX_PLANES]	An array of structures describing forma of each plane this pixel format consists of The number of valid entries in this array has to be put in the num_planes field.
__u8	num_planes	Number of planes (i.e. separate memory buffers) for this format and the number o valid entries in the plane_fmt array.
__u8	flags	Flags set by the application or driver, see *Format Flags*.
union {	(anonymous)	
__u8	ycbcr_enc	Y' CbCr encoding, from enum *v4l2_ycbcr_encoding*. See struc *v4l2_pix_format*.
__u8	hsv_enc	HSV encoding, from enum *v4l2_hsv_encoding*. See struc *v4l2_pix_format*.
}		
__u8	quantization	Quantization range, from enum *v4l2_quantization*. See struc *v4l2_pix_format*.
__u8	xfer_func	Transfer function, from enum *v4l2_xfer_func*. See struc *v4l2_pix_format*.
__u8	reserved[7]	Reserved for future extensions. Should be zeroed by drivers and applications.

Standard Image Formats

In order to exchange images between drivers and applications, it is necessary to have standard image data formats which both sides will interpret the same way. V4L2 includes several such formats, and this section is intended to be an unambiguous specification of the standard image data formats in V4L2.

V4L2 drivers are not limited to these formats, however. Driver-specific formats are possible. In that case the application may depend on a codec to convert images to one of the standard formats when needed. But the data can still be stored and retrieved in the proprietary format. For example, a device may support a proprietary compressed format. Applications can still capture and save the data in the compressed format, saving much disk space, and later use a codec to convert the images to the X Windows screen format when the video is to be displayed.

Even so, ultimately, some standard formats are needed, so the V4L2 specification would not be complete without well-defined standard formats.

The V4L2 standard formats are mainly uncompressed formats. The pixels are always arranged in memory from left to right, and from top to bottom. The first byte of data in the image buffer is always for the leftmost pixel of the topmost row. Following that is the pixel immediately to its right, and so on until the end of the top row of pixels. Following the rightmost pixel of the row there may be zero or more bytes of padding to guarantee that each row of pixel data has a certain alignment. Following the pad bytes, if any, is data for the leftmost pixel of the second row from the top, and so on. The last row has just as many pad bytes after it as the other rows.

In V4L2 each format has an identifier which looks like `PIX_FMT_XXX`, defined in the *videodev2.h* header file. These identifiers represent *four character (FourCC) codes* which are also listed below, however they are not the same as those used in the Windows world.

For some formats, data is stored in separate, discontiguous memory buffers. Those formats are identified by a separate set of FourCC codes and are referred to as "multi-planar formats". For example, a *YUV422* frame is normally stored in one memory buffer, but it can also be placed in two or three separate buffers, with Y component in one buffer and CbCr components in another in the 2-planar version or with each component in its own buffer in the 3-planar case. Those sub-buffers are referred to as "*planes*".

Indexed Format

In this format each pixel is represented by an 8 bit index into a 256 entry ARGB palette. It is intended for *Video Output Overlays* only. There are no ioctls to access the palette, this must be done with ioctls of the Linux framebuffer API.

Table 47: Indexed Image Format

Identifier	Code		Byte 0							
		Bit	7	6	5	4	3	2	1	0
`V4L2_PIX_FMT_PAL8`	'PAL8'		i_7	i_6	i_5	i_4	i_3	i_2	i_1	i_0

RGB Formats

Description

These formats are designed to match the pixel formats of typical PC graphics frame buffers. They occupy 8, 16, 24 or 32 bits per pixel. These are all packed-pixel formats, meaning all the data for a pixel lie next to each other in memory.

Table 48: RGB Image Formats

Identifier	Code	Byte 0 in memory								Byte 1								Byte 2								Byte 3							
		7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
`V4L2_PIX_FMT_RGB332`	'RGB1'	r_2	r_1	r_0	g_2	g_1	g_0	b_1	b_0																								
`V4L2_PIX_FMT_ARGB444`	'AR12'	g_3	g_2	g_1	g_0	b_3	b_2	b_1	b_0	a_3	a_2	a_1	a_0	r_3	r_2	r_1	r_0																
`V4L2_PIX_FMT_XRGB444`	'XR12'	g_3	g_2	g_1	g_0	b_3	b_2	b_1	b_0	-	-	-	-	r_3	r_2	r_1	r_0																

continues on next page

Table 48 – continued from previous page

Identifier	Code	Byte 0 in memory								Byte 1								Byte 2								Byte 3							
		7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
V4L2_PIX_FMT_RGBA444	'RA12'	b_3	b_2	b_1	b_0	a_3	a_2	a_1	a_0	r_3	r_2	r_1	r_0	g_3	g_2	g_1	g_0																
V4L2_PIX_FMT_RGBX444	'RX12'	b_3	b_2	b_1	b_0	-	-	-	-	r_3	r_2	r_1	r_0	g_3	g_2	g_1	g_0																
V4L2_PIX_FMT_ABGR444	'AB12'	g_3	g_2	g_1	g_0	r_3	r_2	r_1	r_0	a_3	a_2	a_1	a_0	b_3	b_2	b_1	b_0																
V4L2_PIX_FMT_XBGR444	'XB12'	g_3	g_2	g_1	g_0	r_3	r_2	r_1	r_0	-	-	-	-	b_3	b_2	b_1	b_0																
V4L2_PIX_FMT_BGRA444	'BA12'	r_3	r_2	r_1	r_0	a_3	a_2	a_1	a_0	b_3	b_2	b_1	b_0	g_3	g_2	g_1	g_0																
V4L2_PIX_FMT_BGRX444	'BX12'	r_3	r_2	r_1	r_0	-	-	-	-	b_3	b_2	b_1	b_0	g_3	g_2	g_1	g_0																
V4L2_PIX_FMT_ARGB555	'AR15'	g_2	g_1	g_0	b_4	b_3	b_2	b_1	b_0	a	r_4	r_3	r_2	r_1	r_0	g_4	g_3																
V4L2_PIX_FMT_XRGB555	'XR15'	g_2	g_1	g_0	b_4	b_3	b_2	b_1	b_0	-	r_4	r_3	r_2	r_1	r_0	g_4	g_3																
V4L2_PIX_FMT_RGBA555	'RA15'	g_1	g_0	b_4	b_3	b_2	b_1	b_0	a	r_4	r_3	r_2	r_1	r_0	g_4	g_3	g_2																
V4L2_PIX_FMT_RGBX555	'RX15'	g_1	g_0	b_4	b_3	b_2	b_1	b_0	-	r_4	r_3	r_2	r_1	r_0	g_4	g_3	g_2																
V4L2_PIX_FMT_ABGR555	'AB15'	g_2	g_1	g_0	r_4	r_3	r_2	r_1	r_0	a	b_4	b_3	b_2	b_1	b_0	g_4	g_3																
V4L2_PIX_FMT_XBGR555	'XB15'	g_2	g_1	g_0	r_4	r_3	r_2	r_1	r_0	-	b_4	b_3	b_2	b_1	b_0	g_4	g_3																
V4L2_PIX_FMT_BGRA555	'BA15'	g_1	g_0	r_4	r_3	r_2	r_1	r_0	a	b_4	b_3	b_2	b_1	b_0	g_4	g_3	g_2																
V4L2_PIX_FMT_BGRX555	'BX15'	g_1	g_0	r_4	r_3	r_2	r_1	r_0	-	b_4	b_3	b_2	b_1	b_0	g_4	g_3	g_2																
V4L2_PIX_FMT_RGB565	'RGBP'	g_2	g_1	g_0	b_4	b_3	b_2	b_1	b_0	r_4	r_3	r_2	r_1	r_0	g_5	g_4	g_3																
V4L2_PIX_FMT_ARGB555X	'AR15' | (1 << 31)	a	r_4	r_3	r_2	r_1	r_0	g_4	g_3	g_2	g_1	g_0	b_4	b_3	b_2	b_1	b_0																
V4L2_PIX_FMT_XRGB555X	'XR15' | (1 << 31)	-	r_4	r_3	r_2	r_1	r_0	g_4	g_3	g_2	g_1	g_0	b_4	b_3	b_2	b_1	b_0																
V4L2_PIX_FMT_RGB565X	'RGBR'	r_4	r_3	r_2	r_1	r_0	g_5	g_4	g_3	g_2	g_1	g_0	b_4	b_3	b_2	b_1	b_0																
V4L2_PIX_FMT_BGR24	'BGR3'	b_7	b_6	b_5	b_4	b_3	b_2	b_1	b_0	g_7	g_6	g_5	g_4	g_3	g_2	g_1	g_0	r_7	r_6	r_5	r_4	r_3	r_2	r_1	r_0								
V4L2_PIX_FMT_RGB24	'RGB3'	r_7	r_6	r_5	r_4	r_3	r_2	r_1	r_0	g_7	g_6	g_5	g_4	g_3	g_2	g_1	g_0	b_7	b_6	b_5	b_4	b_3	b_2	b_1	b_0								
V4L2_PIX_FMT_BGR666	'BGRH'	b_5	b_4	b_3	b_2	b_1	b_0	g_5	g_4	g_3	g_2	g_1	g_0	r_5	r_4	r_3	r_2	r_1	r_0	-	-	-	-	-	-	-	-	-	-	-	-	-	-
V4L2_PIX_FMT_ABGR32	'AR24'	b_7	b_6	b_5	b_4	b_3	b_2	b_1	b_0	g_7	g_6	g_5	g_4	g_3	g_2	g_1	g_0	r_7	r_6	r_5	r_4	r_3	r_2	r_1	r_0	a_7	a_6	a_5	a_4	a_3	a_2	a_1	a_0
V4L2_PIX_FMT_XBGR32	'XR24'	b_7	b_6	b_5	b_4	b_3	b_2	b_1	b_0	g_7	g_6	g_5	g_4	g_3	g_2	g_1	g_0	r_7	r_6	r_5	r_4	r_3	r_2	r_1	r_0	-	-	-	-	-	-	-	-
V4L2_PIX_FMT_BGRA32	'RA24'	a_7	a_6	a_5	a_4	a_3	a_2	a_1	a_0	b_7	b_6	b_5	b_4	b_3	b_2	b_1	b_0	g_7	g_6	g_5	g_4	g_3	g_2	g_1	g_0	r_7	r_6	r_5	r_4	r_3	r_2	r_1	r_0
V4L2_PIX_FMT_BGRX32	'RX24'	-	-	-	-	-	-	-	-	b_7	b_6	b_5	b_4	b_3	b_2	b_1	b_0	g_7	g_6	g_5	g_4	g_3	g_2	g_1	g_0	r_7	r_6	r_5	r_4	r_3	r_2	r_1	r_0
V4L2_PIX_FMT_RGBA32	'AB24'	r_7	r_6	r_5	r_4	r_3	r_2	r_1	r_0	g_7	g_6	g_5	g_4	g_3	g_2	g_1	g_0	b_7	b_6	b_5	b_4	b_3	b_2	b_1	b_0	a_7	a_6	a_5	a_4	a_3	a_2	a_1	a_0
V4L2_PIX_FMT_RGBX32	'XB24'	r_7	r_6	r_5	r_4	r_3	r_2	r_1	r_0	g_7	g_6	g_5	g_4	g_3	g_2	g_1	g_0	b_7	b_6	b_5	b_4	b_3	b_2	b_1	b_0	-	-	-	-	-	-	-	-
V4L2_PIX_FMT_ARGB32	'BA24'	a_7	a_6	a_5	a_4	a_3	a_2	a_1	a_0	r_7	r_6	r_5	r_4	r_3	r_2	r_1	r_0	g_7	g_6	g_5	g_4	g_3	g_2	g_1	g_0	b_7	b_6	b_5	b_4	b_3	b_2	b_1	b_0
V4L2_PIX_FMT_XRGB32	'BX24'	-	-	-	-	-	-	-	-	r_7	r_6	r_5	r_4	r_3	r_2	r_1	r_0	g_7	g_6	g_5	g_4	g_3	g_2	g_1	g_0	b_7	b_6	b_5	b_4	b_3	b_2	b_1	b_0

Note: Bit 7 is the most significant bit.

The usage and value of the alpha bits (a) in the ARGB and ABGR formats (collectively referred to as alpha formats) depend on the device type and hardware operation. *Capture* devices (including capture queues of mem-to-mem devices) fill the alpha component in memory. When the device outputs an alpha channel the alpha component will have a meaningful value. Otherwise, when the device doesn't output an alpha channel but can set the alpha bit to a user-configurable value, the *V4L2_CID_ALPHA_COMPONENT* control is used to specify that alpha value, and the alpha component of all pixels will be set to the value specified by that control. Otherwise a corresponding format without an alpha component (XRGB or XBGR) must be used instead of an alpha format.

Output devices (including output queues of mem-to-mem devices and *video output overlay* devices) read the alpha component from memory. When the device processes the alpha channel the alpha component must be filled with meaningful values by applications. Otherwise a corresponding format without an alpha component (XRGB or XBGR) must be used instead of an alpha format.

The XRGB and XBGR formats contain undefined bits (-). Applications, devices and drivers must ignore those bits, for both *Video Capture Interface* and *Video Output Interface* devices.

Byte Order. Each cell is one byte.

Table 49: RGB byte order

start + 0:	B_{00}	G_{00}	R_{00}	B_{01}	G_{01}	R_{01}	B_{02}	G_{02}	R_{02}	B_{03}	G_{03}	R_{03}
start + 12:	B_{10}	G_{10}	R_{10}	B_{11}	G_{11}	R_{11}	B_{12}	G_{12}	R_{12}	B_{13}	G_{13}	R_{13}
start + 24:	B_{20}	G_{20}	R_{20}	B_{21}	G_{21}	R_{21}	B_{22}	G_{22}	R_{22}	B_{23}	G_{23}	R_{23}
start + 36:	B_{30}	G_{30}	R_{30}	B_{31}	G_{31}	R_{31}	B_{32}	G_{32}	R_{32}	B_{33}	G_{33}	R_{33}

Formats defined in *Deprecated Packed RGB Image Formats* are deprecated and must not be used by new drivers. They are documented here for reference. The meaning of their alpha bits `(a)` are ill-defined and interpreted as in either the corresponding ARGB or XRGB format, depending on the driver.

Table 50: Deprecated Packed RGB Image Formats

Identifier	Code	Byte 0 in memory								Byte 1								Byte 2								Byte 3							
		7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
`V4L2_PIX_FMT_RGB444`	'R444'	g_3	g_2	g_1	g_0	b_3	b_2	b_1	b_0	a_3	a_2	a_1	a_0	r_3	r_2	r_1	r_0																
`V4L2_PIX_FMT_RGB555`	'RGBO'	g_2	g_1	g_0	b_4	b_3	b_2	b_1	b_0	a	r_4	r_3	r_2	r_1	r_0	g_4	g_3																
`V4L2_PIX_FMT_RGB555X`	'RGBQ'	a	r_4	r_3	r_2	r_1	r_0	g_4	g_3	g_2	g_1	g_0	b_4	b_3	b_2	b_1	b_0																
`V4L2_PIX_FMT_BGR32`	'BGR4'	b_7	b_6	b_5	b_4	b_3	b_2	b_1	b_0	g_7	g_6	g_5	g_4	g_3	g_2	g_1	g_0	r_7	r_6	r_5	r_4	r_3	r_2	r_1	r_0	a_7	a_6	a_5	a_4	a_3	a_2	a_1	a_0
`V4L2_PIX_FMT_RGB32`	'RGB4'	a_7	a_6	a_5	a_4	a_3	a_2	a_1	a_0	r_7	r_6	r_5	r_4	r_3	r_2	r_1	r_0	g_7	g_6	g_5	g_4	g_3	g_2	g_1	g_0	b_7	b_6	b_5	b_4	b_3	b_2	b_1	b_0

A test utility to determine which RGB formats a driver actually supports is available from the LinuxTV v4l-dvb repository. See https://linuxtv.org/repo/ for access instructions.

Raw Bayer Formats

Description

The raw Bayer formats are used by image sensors before much if any processing is performed on the image. The formats contain green, red and blue components, with alternating lines of red and green, and blue and green pixels in different orders. See also the Wikipedia article on Bayer filter.

V4L2_PIX_FMT_SRGGB8 ('RGGB'), V4L2_PIX_FMT_SGRBG8 ('GRBG'), V4L2_PIX_FMT_SGBRG8 ('GBRG'), V4L2_PIX_FMT_SBGGR8 ('BA81'),

8-bit Bayer formats

Description

These four pixel formats are raw sRGB / Bayer formats with 8 bits per sample. Each sample is stored in a byte. Each n-pixel row contains n/2 green samples and n/2 blue or red samples, with alternating red and blue rows. They are conventionally described as GRGR⋯BGBG⋯, RGRG⋯GBGB⋯, etc. Below is an example of a small V4L2_PIX_FMT_SBGGR8 image:

Byte Order. Each cell is one byte.

start + 0:	B_{00}	G_{01}	B_{02}	G_{03}
start + 4:	G_{10}	R_{11}	G_{12}	R_{13}
start + 8:	B_{20}	G_{21}	B_{22}	G_{23}
start + 12:	G_{30}	R_{31}	G_{32}	R_{33}

V4L2_PIX_FMT_SRGGB10 ('RG10'), V4L2_PIX_FMT_SGRBG10 ('BA10'), V4L2_PIX_FMT_SGBRG10 ('GB10'), V4L2_PIX_FMT_SBGGR10 ('BG10'),

V4L2_PIX_FMT_SGRBG10 V4L2_PIX_FMT_SGBRG10 V4L2_PIX_FMT_SBGGR10 10-bit Bayer formats expanded to 16 bits

Description

These four pixel formats are raw sRGB / Bayer formats with 10 bits per sample. Each sample is stored in a 16-bit word, with 6 unused high bits filled with zeros. Each n-pixel row contains n/2 green samples and n/2 blue or red samples, with alternating red and blue rows. Bytes are stored in memory in little endian order. They are conventionally described as GRGR⋯BGBG⋯, RGRG⋯GBGB⋯, etc. Below is an example of one of these formats:

Byte Order. Each cell is one byte, the 6 most significant bits in the high bytes are 0.

start + 0:	B_{00low}	B_{00high}	G_{01low}	G_{01high}	B_{02low}	B_{02high}	G_{03low}	G_{03high}
start + 8:	G_{10low}	G_{10high}	R_{11low}	R_{11high}	G_{12low}	G_{12high}	R_{13low}	R_{13high}
start + 16:	B_{20low}	B_{20high}	G_{21low}	G_{21high}	B_{22low}	B_{22high}	G_{23low}	G_{23high}
start + 24:	G_{30low}	G_{30high}	R_{31low}	R_{31high}	G_{32low}	G_{32high}	R_{33low}	R_{33high}

V4L2_PIX_FMT_SRGGB10P ('pRAA'), V4L2_PIX_FMT_SGRBG10P ('pgAA'), V4L2_PIX_FMT_SGBRG10P ('pGAA'), V4L2_PIX_FMT_SBGGR10P ('pBAA'),

V4L2_PIX_FMT_SGRBG10P V4L2_PIX_FMT_SGBRG10P V4L2_PIX_FMT_SBGGR10P 10-bit packed Bayer formats

Description

These four pixel formats are packed raw sRGB / Bayer formats with 10 bits per sample. Every four consecutive samples are packed into 5 bytes. Each of the first 4 bytes contain the 8 high order bits of the pixels, and the 5th byte contains the 2 least significants bits of each pixel, in the same order.

Each n-pixel row contains n/2 green samples and n/2 blue or red samples, with alternating green-red and green-blue rows. They are conventionally described as GRGR⋯BGBG⋯, RGRG⋯GBGB⋯, etc. Below is an example of a small V4L2_PIX_FMT_SBGGR10P image:

Byte Order. Each cell is one byte.

start + 0:	B_{00high}	G_{01high}	B_{02high}	G_{03high}	G_{03low}(bits 7–6) B_{02low}(bits 5–4) G_{01low}(bits 3–2) B_{00low}(bits 1–0)
start + 5:	G_{10high}	R_{11high}	G_{12high}	R_{13high}	R_{13low}(bits 7–6) G_{12low}(bits 5–4) R_{11low}(bits 3–2) G_{10low}(bits 1–0)
start + 10:	B_{20high}	G_{21high}	B_{22high}	G_{23high}	G_{23low}(bits 7–6) B_{22low}(bits 5–4) G_{21low}(bits 3–2) B_{20low}(bits 1–0)
start + 15:	G_{30high}	R_{31high}	G_{32high}	R_{33high}	R_{33low}(bits 7–6) G_{32low}(bits 5–4) R_{31low}(bits 3–2) G_{30low}(bits 1–0)

V4L2_PIX_FMT_SBGGR10ALAW8 ('aBA8'), V4L2_PIX_FMT_SGBRG10ALAW8 ('aGA8'), V4L2_PIX_FMT_SGRBG10ALAW8 ('agA8'), V4L2_PIX_FMT_SRGGB10ALAW8 ('aRA8'),

V4L2_PIX_FMT_SGBRG10ALAW8 V4L2_PIX_FMT_SGRBG10ALAW8 V4L2_PIX_FMT_SRGGB10ALAW8 10-bit Bayer formats compressed to 8 bits

Description

These four pixel formats are raw sRGB / Bayer formats with 10 bits per color compressed to 8 bits each, using the A-LAW algorithm. Each color component consumes 8 bits of memory. In other respects this format is similar to *V4L2_PIX_FMT_SRGGB8 ('RGGB'), V4L2_PIX_FMT_SGRBG8 ('GRBG'), V4L2_PIX_FMT_SGBRG8 ('GBRG'), V4L2_PIX_FMT_SBGGR8 ('BA81'),*.

V4L2_PIX_FMT_SBGGR10DPCM8 ('bBA8'), V4L2_PIX_FMT_SGBRG10DPCM8 ('bGA8'), V4L2_PIX_FMT_SGRBG10DPCM8 ('BD10'), V4L2_PIX_FMT_SRGGB10DPCM8 ('bRA8'),

man V4L2_PIX_FMT_SBGGR10DPCM8(2)

V4L2_PIX_FMT_SGBRG10DPCM8 V4L2_PIX_FMT_SGRBG10DPCM8 V4L2_PIX_FMT_SRGGB10DPCM8 10-bit Bayer formats compressed to 8 bits

Description

These four pixel formats are raw sRGB / Bayer formats with 10 bits per colour compressed to 8 bits each, using DPCM compression. DPCM, differential pulse-code modulation, is lossy. Each colour component consumes 8 bits of memory. In other respects this format is similar to *V4L2_PIX_FMT_SRGGB10 ('RG10'), V4L2_PIX_FMT_SGRBG10 ('BA10'), V4L2_PIX_FMT_SGBRG10 ('GB10'), V4L2_PIX_FMT_SBGGR10 ('BG10'),*.

V4L2_PIX_FMT_IPU3_SBGGR10 ('ip3b'), V4L2_PIX_FMT_IPU3_SGBRG10 ('ip3g'), V4L2_PIX_FMT_IPU3_SGRBG10 ('ip3G'), V4L2_PIX_FMT_IPU3_SRGGB10 ('ip3r')

10-bit Bayer formats

Description

These four pixel formats are used by Intel IPU3 driver, they are raw sRGB / Bayer formats with 10 bits per sample with every 25 pixels packed to 32 bytes leaving 6 most significant bits padding in the last byte. The format is little endian.

In other respects this format is similar to *V4L2_PIX_FMT_SRGGB10 ('RG10'), V4L2_PIX_FMT_SGRBG10 ('BA10'), V4L2_PIX_FMT_SGBRG10 ('GB10'), V4L2_PIX_FMT_SBGGR10 ('BG10'),*. Below is an example of a small image in V4L2_PIX_FMT_IPU3_SBGGR10 format.

Byte Order. Each cell is one byte.

start + 0:	$B_{0000low}$	$G_{0001low}$(bits 7–2) $B_{0000high}$(bits 1–0)	$B_{0002low}$(bits 7–4) $G_{0001high}$(bits 3–0)	$G_{0003low}$(bits 7–6) $B_{0002high}$(bits 5–0)
start + 4:	$G_{0003high}$	$B_{0004low}$	$G_{0005low}$(bits 7–2) $B_{0004high}$(bits 1–0)	$B_{0006low}$(bits 7–4) $G_{0005high}$(bits 3–0)
start + 8:	$G_{0007low}$(bits 7–6) $B_{0006high}$(bits 5–0)	$G_{0007high}$	$B_{0008low}$	$G_{0009low}$(bits 7–2) $B_{0008high}$(bits 1–0)

continues on next pag

Table 51 – continued from previous page

start + 12:	$B_{0010low}$(bits 7–4) $G_{0009high}$(bits 3–0)	$G_{0011low}$(bits 7–6) $B_{0010high}$(bits 5–0)	$G_{0011high}$	$B_{0012low}$
start + 16:	$G_{0013low}$(bits 7–2) $B_{0012high}$(bits 1–0)	$B_{0014low}$(bits 7–4) $G_{0013high}$(bits 3–0)	$G_{0015low}$(bits 7–6) $B_{0014high}$(bits 5–0)	$G_{0015high}$
start + 20	$B_{0016low}$	$G_{0017low}$(bits 7–2) $B_{0016high}$(bits 1–0)	$B_{0018low}$(bits 7–4) $G_{0017high}$(bits 3–0)	$G_{0019low}$(bits 7–6) $B_{0018high}$(bits 5–0)
start + 24:	$G_{0019high}$	$B_{0020low}$	$G_{0021low}$(bits 7–2) $B_{0020high}$(bits 1–0)	$B_{0022low}$(bits 7–4) $G_{0021high}$(bits 3–0)
start + 28:	$G_{0023low}$(bits 7–6) $B_{0022high}$(bits 5–0)	$G_{0023high}$	$B_{0024low}$	$B_{0024high}$(bits 1–0)
start + 32:	$G_{0100low}$	$R_{0101low}$(bits 7–2) $G_{0100high}$(bits 1–0)	$G_{0102low}$(bits 7–4) $R_{0101high}$(bits 3–0)	$R_{0103low}$(bits 7–6) $G_{0102high}$(bits 5–0)
start + 36:	$R_{0103high}$	$G_{0104low}$	$R_{0105low}$(bits 7–2) $G_{0104high}$(bits 1–0)	$G_{0106low}$(bits 7–4) $R_{0105high}$(bits 3–0)
start + 40:	$R_{0107low}$(bits 7–6) $G_{0106high}$(bits 5–0)	$R_{0107high}$	$G_{0108low}$	$R_{0109low}$(bits 7–2) $G_{0108high}$(bits 1–0)
start + 44:	$G_{0110low}$(bits 7–4) $R_{0109high}$(bits 3–0)	$R_{0111low}$(bits 7–6) $G_{0110high}$(bits 5–0)	$R_{0111high}$	$G_{0112low}$
start + 48:	$R_{0113low}$(bits 7–2) $G_{0112high}$(bits 1–0)	$G_{0114low}$(bits 7–4) $R_{0113high}$(bits 3–0)	$R_{0115low}$(bits 7–6) $G_{0114high}$(bits 5–0)	$R_{0115high}$
start + 52:	$G_{0116low}$	$R_{0117low}$(bits 7–2) $G_{0116high}$(bits 1–0)	$G_{0118low}$(bits 7–4) $R_{0117high}$(bits 3–0)	$R_{0119low}$(bits 7–6) $G_{0118high}$(bits 5–0)
start + 56:	$R_{0119high}$	$G_{0120low}$	$R_{0121low}$(bits 7–2) $G_{0120high}$(bits 1–0)	$G_{0122low}$(bits 7–4) $R_{0121high}$(bits 3–0)
start + 60:	$R_{0123low}$(bits 7–6) $G_{0122high}$(bits 5–0)	$R_{0123high}$	$G_{0124low}$	$G_{0124high}$(bits 1–0)
start + 64:	$B_{0200low}$	$G_{0201low}$(bits 7–2) $B_{0200high}$(bits 1–0)	$B_{0202low}$(bits 7–4) $G_{0201high}$(bits 3–0)	$G_{0203low}$(bits 7–6) $B_{0202high}$(bits 5–0)
start + 68:	$G_{0203high}$	$B_{0204low}$	$G_{0205low}$(bits 7–2) $B_{0204high}$(bits 1–0)	$B_{0206low}$(bits 7–4) $G_{0205high}$(bits 3–0)
start + 72:	$G_{0207low}$(bits 7–6) $B_{0206high}$(bits 5–0)	$G_{0207high}$	$B_{0208low}$	$G_{0209low}$(bits 7–2) $B_{0208high}$(bits 1–0)

continues on next pag

Table 51 – continued from previous page

start + 76:	$B_{0210low}$(bits 7–4) $G_{0209high}$(bits 3–0)	$G_{0211low}$(bits 7–6) $B_{0210high}$(bits 5–0)	$G_{0211high}$	$B_{0212low}$
start + 80:	$G_{0213low}$(bits 7–2) $B_{0212high}$(bits 1–0)	$B_{0214low}$(bits 7–4) $G_{0213high}$(bits 3–0)	$G_{0215low}$(bits 7–6) $B_{0214high}$(bits 5–0)	$G_{0215high}$
start + 84:	$B_{0216low}$	$G_{0217low}$(bits 7–2) $B_{0216high}$(bits 1–0)	$B_{0218low}$(bits 7–4) $G_{0217high}$(bits 3–0)	$G_{0219low}$(bits 7–6) $B_{0218high}$(bits 5–0)
start + 88:	$G_{0219high}$	$B_{0220low}$	$G_{0221low}$(bits 7–2) $B_{0220high}$(bits 1–0)	$B_{0222low}$(bits 7–4) $G_{0221high}$(bits 3–0)
start + 92:	$G_{0223low}$(bits 7–6) $B_{0222high}$(bits 5–0)	$G_{0223high}$	$B_{0224low}$	$B_{0224high}$(bits 1–0)
start + 96:	$G_{0300low}$	$R_{0301low}$(bits 7–2) $G_{0300high}$(bits 1–0)	$G_{0302low}$(bits 7–4) $R_{0301high}$(bits 3–0)	$R_{0303low}$(bits 7–6) $G_{0302high}$(bits 5–0)
start + 100:	$R_{0303high}$	$G_{0304low}$	$R_{0305low}$(bits 7–2) $G_{0304high}$(bits 1–0)	$G_{0306low}$(bits 7–4) $R_{0305high}$(bits 3–0)
start + 104:	$R_{0307low}$(bits 7–6) $G_{0306high}$(bits 5–0)	$R_{0307high}$	$G_{0308low}$	$R_{0309low}$(bits 7–2) $G_{0308high}$(bits 1–0)
start + 108:	$G_{0310low}$(bits 7–4) $R_{0309high}$(bits 3–0)	$R_{0311low}$(bits 7–6) $G_{0310high}$(bits 5–0)	$R_{0311high}$	$G_{0312low}$
start + 112:	$R_{0313low}$(bits 7–2) $G_{0312high}$(bits 1–0)	$G_{0314low}$(bits 7–4) $R_{0313high}$(bits 3–0)	$R_{0315low}$(bits 7–6) $G_{0314high}$(bits 5–0)	$R_{0315high}$
start + 116:	$G_{0316low}$	$R_{0317low}$(bits 7–2) $G_{0316high}$(bits 1–0)	$G_{0318low}$(bits 7–4) $R_{0317high}$(bits 3–0)	$R_{0319low}$(bits 7–6) $G_{0318high}$(bits 5–0)
start + 120:	$R_{0319high}$	$G_{0320low}$	$R_{0321low}$(bits 7–2) $G_{0320high}$(bits 1–0)	$G_{0322low}$(bits 7–4) $R_{0321high}$(bits 3–0)
start + 124:	$R_{0323low}$(bits 7–6) $G_{0322high}$(bits 5–0)	$R_{0323high}$	$G_{0324low}$	$G_{0324high}$(bits 1–0)

V4L2_PIX_FMT_SRGGB12 ('RG12'), V4L2_PIX_FMT_SGRBG12 ('BA12'), V4L2_PIX_FMT_SGBRG12 ('GB12'), V4L2_PIX_FMT_SBGGR12 ('BG12'),

V4L2_PIX_FMT_SGRBG12 V4L2_PIX_FMT_SGBRG12 V4L2_PIX_FMT_SBGGR12 12-bit Bayer formats expanded to 16 bits

Description

These four pixel formats are raw sRGB / Bayer formats with 12 bits per colour. Each colour component is stored in a 16-bit word, with 4 unused high bits filled with zeros. Each n-pixel row contains n/2 green samples and n/2 blue or red samples, with alternating red and blue rows. Bytes are stored in memory in little endian order. They are conventionally described as GRGR⋯BGBG⋯, RGRG⋯GBGB ⋯, etc. Below is an example of a small V4L2_PIX_FMT_SBGGR12 image:

Byte Order. Each cell is one byte, the 4 most significant bits in the high bytes are 0.

start + 0:	B_{00low}	B_{00high}	G_{01low}	G_{01high}	B_{02low}	B_{02high}	G_{03low}	G_{03high}
start + 8:	G_{10low}	G_{10high}	R_{11low}	R_{11high}	G_{12low}	G_{12high}	R_{13low}	R_{13high}
start + 16:	B_{20low}	B_{20high}	G_{21low}	G_{21high}	B_{22low}	B_{22high}	G_{23low}	G_{23high}
start + 24:	G_{30low}	G_{30high}	R_{31low}	R_{31high}	G_{32low}	G_{32high}	R_{33low}	R_{33high}

V4L2_PIX_FMT_SRGGB12P ('pRCC'), V4L2_PIX_FMT_SGRBG12P ('pgCC'), V4L2_PIX_FMT_SGBRG12P ('pGCC'), V4L2_PIX_FMT_SBGGR12P ('pBCC'),

12-bit packed Bayer formats

Description

These four pixel formats are packed raw sRGB / Bayer formats with 12 bits per colour. Every two consecutive samples are packed into three bytes. Each of the first two bytes contain the 8 high order bits of the pixels, and the third byte contains the four least significants bits of each pixel, in the same order.

Each n-pixel row contains n/2 green samples and n/2 blue or red samples, with alternating green-red and green-blue rows. They are conventionally described as GRGR⋯BGBG⋯, RGRG⋯GBGB⋯, etc. Below is an example of a small V4L2_PIX_FMT_SBGGR12P image:

Byte Order. Each cell is one byte.

start + 0:	B_{00high}	G_{01high}	G_{01low}(bits 7–4) B_{00low}(bits 3–0)	B_{02high}	G_{03high}	G_{03low}(bits 7–4) B_{02low}(bits 3–0)
start + 6:	G_{10high}	R_{11high}	R_{11low}(bits 7–4) G_{10low}(bits 3–0)	G_{12high}	R_{13high}	R_{13low}(bits 3–2) G_{12low}(bits 3–0)
start + 12:	B_{20high}	G_{21high}	G_{21low}(bits 7–4) B_{20low}(bits 3–0)	B_{22high}	G_{23high}	G_{23low}(bits 7–4) B_{22low}(bits 3–0)
start + 18:	G_{30high}	R_{31high}	R_{31low}(bits 7–4) G_{30low}(bits 3–0)	G_{32high}	R_{33high}	R_{33low}(bits 3–2) G_{32low}(bits 3–0)

V4L2_PIX_FMT_SRGGB14 ('RG14'), V4L2_PIX_FMT_SGRBG14 ('GR14'), V4L2_PIX_FMT_SGBRG14 ('GB14'), V4L2_PIX_FMT_SBGGR14 ('BG14'),

14-bit Bayer formats expanded to 16 bits

Description

These four pixel formats are raw sRGB / Bayer formats with 14 bits per colour. Each sample is stored in a 16-bit word, with two unused high bits filled with zeros. Each n-pixel row contains n/2 green samples and n/2 blue or red samples, with alternating red and blue rows. Bytes are stored in memory in little endian order. They are conventionally described as GRGR⋯BGBG⋯, RGRG⋯GBGB⋯, etc. Below is an example of a small V4L2_PIX_FMT_SBGGR14 image:

Byte Order. Each cell is one byte, the two most significant bits in the high bytes are zero.

start + 0:	B_{00low}	B_{00high}	G_{01low}	G_{01high}	B_{02low}	B_{02high}	G_{03low}	G_{03high}
start + 8:	G_{10low}	G_{10high}	R_{11low}	R_{11high}	G_{12low}	G_{12high}	R_{13low}	R_{13high}
start + 16:	B_{20low}	B_{20high}	G_{21low}	G_{21high}	B_{22low}	B_{22high}	G_{23low}	G_{23high}
start + 24:	G_{30low}	G_{30high}	R_{31low}	R_{31high}	G_{32low}	G_{32high}	R_{33low}	R_{33high}

V4L2_PIX_FMT_SRGGB14P ('pREE'), V4L2_PIX_FMT_SGRBG14P ('pgEE'), V4L2_PIX_FMT_SGBRG14P ('pGEE'), V4L2_PIX_FMT_SBGGR14P ('pBEE'),

man V4L2_PIX_FMT_SRGGB14P(2)

V4L2_PIX_FMT_SGRBG14P V4L2_PIX_FMT_SGBRG14P V4L2_PIX_FMT_SBGGR14P 14-bit packed Bayer formats

Description

These four pixel formats are packed raw sRGB / Bayer formats with 14 bits per colour. Every four consecutive samples are packed into seven bytes. Each of the first four bytes contain the eight high order bits of the pixels, and the three following bytes contains the six least significants bits of each pixel, in the same order.

Each n-pixel row contains n/2 green samples and n/2 blue or red samples, with alternating green-red and green-blue rows. They are conventionally described as GRGR⋯BGBG⋯, RGRG⋯GBGB⋯, etc. Below is an example of one of these formats:

Byte Order. Each cell is one byte.

start + 0	B_{00high}	G_{01high}	B_{02high}	G_{03high}	$G_{01low\ bits\ 1-0}$(bits 7–6) $B_{00low\ bits\ 5-0}$(bits 5–0)	$B_{02low\ bits\ 3-0}$(bits 7–4) $G_{01low\ bits\ 5-2}$(bits 3–0)	$G_{03low\ bits\ 5-0}$(bits $B_{02low\ bits\ 5-4}$(bits
start + 7	G_{10high}	R_{11high}	G_{12high}	R_{13high}	$R_{11low\ bits\ 1-0}$(bits 7–6) $G_{10low\ bits\ 5-0}$(bits 5–0)	$G_{12low\ bits\ 3-0}$(bits 7–4) $R_{11low\ bits\ 5-2}$(bits 3–0)	$R_{13low\ bits\ 5-0}$(bits $G_{12low\ bits\ 5-4}$(bits
start + 14	B_{20high}	G_{21high}	B_{22high}	G_{23high}	$G_{21low\ bits\ 1-0}$(bits 7–6) $B_{20low\ bits\ 5-0}$(bits 5–0)	$B_{22low\ bits\ 3-0}$(bits 7–4) $G_{21low\ bits\ 5-2}$(bits 3–0)	$G_{23low\ bits\ 5-0}$(bits $B_{22low\ bits\ 5-4}$(bits
start + 21	G_{30high}	R_{31high}	G_{32high}	R_{33high}	$R_{31low\ bits\ 1-0}$(bits 7–6) $G_{30low\ bits\ 5-0}$(bits 5–0)	$G_{32low\ bits\ 3-0}$(bits 7–4) $R_{31low\ bits\ 5-2}$(bits 3–0)	$R_{33low\ bits\ 5-0}$(bits $G_{32low\ bits\ 5-4}$(bits

V4L2_PIX_FMT_SRGGB16 ('RG16'), V4L2_PIX_FMT_SGRBG16 ('GR16'), V4L2_PIX_FMT_SGBRG16 ('GB16'), V4L2_PIX_FMT_SBGGR16 ('BYR2'),

16-bit Bayer formats

Description

These four pixel formats are raw sRGB / Bayer formats with 16 bits per sample. Each sample is stored in a 16-bit word. Each n-pixel row contains n/2 green samples and n/2 blue or red samples, with alternating red and blue rows. Bytes are stored in memory in little endian order. They are conventionally described as GRGR⋯BGBG⋯, RGRG⋯GBGB⋯, etc. Below is an example of a small V4L2_PIX_FMT_SBGGR16 image:

Byte Order. Each cell is one byte.

start + 0:	B_{00low}	B_{00high}	G_{01low}	G_{01high}	B_{02low}	B_{02high}	G_{03low}	G_{03high}
start + 8:	G_{10low}	G_{10high}	R_{11low}	R_{11high}	G_{12low}	G_{12high}	R_{13low}	R_{13high}
start + 16:	B_{20low}	B_{20high}	G_{21low}	G_{21high}	B_{22low}	B_{22high}	G_{23low}	G_{23high}
start + 24:	G_{30low}	G_{30high}	R_{31low}	R_{31high}	G_{32low}	G_{32high}	R_{33low}	R_{33high}

YUV Formats

YUV is the format native to TV broadcast and composite video signals. It separates the brightness information (Y) from the color information (U and V or Cb and Cr). The color information consists of red and blue *color difference* signals, this way the green component can be reconstructed by subtracting from the brightness component. See *Colorspaces* for conversion examples. YUV was chosen because early television would only transmit brightness information. To add color in a way compatible with existing receivers a new signal carrier was added to transmit the color difference signals. Secondary in the YUV format the U and V components usually have lower resolution than the Y component. This is an analog video compression technique taking advantage of a property of the human visual system, being more sensitive to brightness information.

Packed YUV formats

Description

Similar to the packed RGB formats these formats store the Y, Cb and Cr component of each pixel in one 16 or 32 bit word.

Table 52: Packed YUV Image Formats

Identifier	Code	Byte 0 in memory								Byte 1								Byte 2								Byte 3							
		7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
V4L2_PIX_FMT_YUV444	'Y444'	Cb_3	Cb_2	Cb_1	Cb_0	Cr_3	Cr_2	Cr_1	Cr_0	a_3	a_2	a_1	a_0	Y'_3	Y'_2	Y'_1	Y'_0																
V4L2_PIX_FMT_YUV555	'YUVO'	Cb_2	Cb_1	Cb_0	Cr_4	Cr_3	Cr_2	Cr_1	Cr_0	a	Y'_4	Y'_3	Y'_2	Y'_1	Y'_0	Cb_4	Cb_3																
V4L2_PIX_FMT_YUV565	'YUVP'	Cb_2	Cb_1	Cb_0	Cr_4	Cr_3	Cr_2	Cr_1	Cr_0	Y'_4	Y'_3	Y'_2	Y'_1	Y'_0	Cb_5	Cb_4	Cb_3																
V4L2_PIX_FMT_YUV32	'YUV4'	a_7	a_6	a_5	a_4	a_3	a_2	a_1	a_0	Y'_7	Y'_6	Y'_5	Y'_4	Y'_3	Y'_2	Y'_1	Y'_0	Cb_7	Cb_6	Cb_5	Cb_4	Cb_3	Cb_2	Cb_1	Cb_0	Cr_7	Cr_6	Cr_5	Cr_4	Cr_3	Cr_2	Cr_1	Cr_0
V4L2_PIX_FMT_AYUV32	'AYUV'	a_7	a_6	a_5	a_4	a_3	a_2	a_1	a_0	Y'_7	Y'_6	Y'_5	Y'_4	Y'_3	Y'_2	Y'_1	Y'_0	Cb_7	Cb_6	Cb_5	Cb_4	Cb_3	Cb_2	Cb_1	Cb_0	Cr_7	Cr_6	Cr_5	Cr_4	Cr_3	Cr_2	Cr_1	Cr_0
V4L2_PIX_FMT_XYUV32	'XYUV'									Y'_7	Y'_6	Y'_5	Y'_4	Y'_3	Y'_2	Y'_1	Y'_0	Cb_7	Cb_6	Cb_5	Cb_4	Cb_3	Cb_2	Cb_1	Cb_0	Cr_7	Cr_6	Cr_5	Cr_4	Cr_3	Cr_2	Cr_1	Cr_0
V4L2_PIX_FMT_VUYA32	'VUYA'	Cr_7	Cr_6	Cr_5	Cr_4	Cr_3	Cr_2	Cr_1	Cr_0	Cb_7	Cb_6	Cb_5	Cb_4	Cb_3	Cb_2	Cb_1	Cb_0	Y'_7	Y'_6	Y'_5	Y'_4	Y'_3	Y'_2	Y'_1	Y'_0	a_7	a_6	a_5	a_4	a_3	a_2	a_1	a_0
V4L2_PIX_FMT_VUYX32	'VUYX'	Cr_7	Cr_6	Cr_5	Cr_4	Cr_3	Cr_2	Cr_1	Cr_0	Cb_7	Cb_6	Cb_5	Cb_4	Cb_3	Cb_2	Cb_1	Cb_0	Y'_7	Y'_6	Y'_5	Y'_4	Y'_3	Y'_2	Y'_1	Y'_0								

Note:

1) Bit 7 is the most significant bit;

2) The value of a = alpha bits is undefined when reading from the driver, ignored when writing to the driver, except when alpha blending has been negotiated for a *Video Overlay* or *Video Output Overlay* for the formats Y444, YUV555 and YUV4. However, for formats AYUV32 and VUYA32, the alpha component is expected to contain a meaningful value that can be used by drivers and applications. And, the formats XYUV32 and VUYX32 contain undefined alpha values that must be ignored by all applications and drivers.

V4L2_PIX_FMT_GREY ('GREY')

Grey-scale image

Description

This is a grey-scale image. It is really a degenerate Y' CbCr format which simply contains no Cb or Cr data.

Byte Order. Each cell is one byte.

start + 0:	Y'_{00}	Y'_{01}	Y'_{02}	Y'_{03}
start + 4:	Y'_{10}	Y'_{11}	Y'_{12}	Y'_{13}
start + 8:	Y'_{20}	Y'_{21}	Y'_{22}	Y'_{23}
start + 12:	Y'_{30}	Y'_{31}	Y'_{32}	Y'_{33}

V4L2_PIX_FMT_Y10 ('Y10 ')

Grey-scale image

Description

This is a grey-scale image with a depth of 10 bits per pixel. Pixels are stored in 16-bit words with unused high bits padded with 0. The least significant byte is stored at lower memory addresses (little-endian).

Byte Order. Each cell is one byte.

start + 0:	Y'_{00low}	Y'_{00high}	Y'_{01low}	Y'_{01high}	Y'_{02low}	Y'_{02high}	Y'_{03low}	Y'_{03high}
start + 8:	Y'_{10low}	Y'_{10high}	Y'_{11low}	Y'_{11high}	Y'_{12low}	Y'_{12high}	Y'_{13low}	Y'_{13high}
start + 16:	Y'_{20low}	Y'_{20high}	Y'_{21low}	Y'_{21high}	Y'_{22low}	Y'_{22high}	Y'_{23low}	Y'_{23high}
start + 24:	Y'_{30low}	Y'_{30high}	Y'_{31low}	Y'_{31high}	Y'_{32low}	Y'_{32high}	Y'_{33low}	Y'_{33high}

V4L2_PIX_FMT_Y12 ('Y12 ')

Grey-scale image

Description

This is a grey-scale image with a depth of 12 bits per pixel. Pixels are stored in 16-bit words with unused high bits padded with 0. The least significant byte is stored at lower memory addresses (little-endian).

Byte Order. Each cell is one byte.

start + 0:	Y'$_{00low}$	Y' 00high	Y'$_{01low}$	Y' 01high	Y'$_{02low}$	Y' 02high	Y'$_{03low}$	Y' 03high
start + 8:	Y'$_{10low}$	Y' 10high	Y'$_{11low}$	Y' 11high	Y'$_{12low}$	Y' 12high	Y'$_{13low}$	Y' 13high
start + 16:	Y'$_{20low}$	Y' 20high	Y'$_{21low}$	Y' 21high	Y'$_{22low}$	Y' 22high	Y'$_{23low}$	Y' 23high
start + 24:	Y'$_{30low}$	Y' 30high	Y'$_{31low}$	Y' 31high	Y'$_{32low}$	Y' 32high	Y'$_{33low}$	Y' 33high

V4L2_PIX_FMT_Y14 ('Y14 ')

Grey-scale image

Description

This is a grey-scale image with a depth of 14 bits per pixel. Pixels are stored in 16-bit words with unused high bits padded with 0. The least significant byte is stored at lower memory addresses (little-endian).

Byte Order. Each cell is one byte.

start + 0:	Y'$_{00low}$	Y' 00high	Y'$_{01low}$	Y' 01high	Y'$_{02low}$	Y' 02high	Y'$_{03low}$	Y' 03high
start + 8:	Y'$_{10low}$	Y' 10high	Y'$_{11low}$	Y' 11high	Y'$_{12low}$	Y' 12high	Y'$_{13low}$	Y' 13high
start + 16:	Y'$_{20low}$	Y' 20high	Y'$_{21low}$	Y' 21high	Y'$_{22low}$	Y' 22high	Y'$_{23low}$	Y' 23high
start + 24:	Y'$_{30low}$	Y' 30high	Y'$_{31low}$	Y' 31high	Y'$_{32low}$	Y' 32high	Y'$_{33low}$	Y' 33high

V4L2_PIX_FMT_Y10BPACK ('Y10B')

Grey-scale image as a bit-packed array

Description

This is a packed grey-scale image format with a depth of 10 bits per pixel. Pixels are stored in a bit-packed array of 10bit bits per pixel, with no padding between them and with the most significant bits coming first from the left.

Bit-packed representation.

pixels cross the byte boundary and have a ratio of 5 bytes for each 4 pixels.

Y' 00[9:2]	Y' 00[1:0]Y' 01[9:4]	Y' 01[3:0]Y' 02[9:6]	Y' 02[5:0]Y' 03[9:8]	Y' 03[7:0]			

V4L2_PIX_FMT_Y10P ('Y10P')

Grey-scale image as a MIPI RAW10 packed array

Description

This is a packed grey-scale image format with a depth of 10 bits per pixel. Every four consecutive pixels are packed into 5 bytes. Each of the first 4 bytes contain the 8 high order bits of the pixels, and the 5th byte contains the 2 least significants bits of each pixel, in the same order.

Bit-packed representation.

Y' 00[9:2]	Y' 01[9:2]	Y' 02[9:2]	Y' 03[9:2]	Y' 03[1:0](bits 7–6) Y' 02[1:0](bits 5–4) Y' 01[1:0](bits 3–2) Y' 00[1:0](bits 1–0)

V4L2_PIX_FMT_Y16 ('Y16 ')

Grey-scale image

Description

This is a grey-scale image with a depth of 16 bits per pixel. The least significant byte is stored at lower memory addresses (little-endian).

Note: The actual sampling precision may be lower than 16 bits, for example 10 bits per pixel with values in range 0 to 1023.

Byte Order. Each cell is one byte.

start 0:	+	Y'_{00low}	Y'_{00high}	Y'_{01low}	Y'_{01high}	Y'_{02low}	Y'_{02high}	Y'_{03low}	Y'_{03high}
start 8:	+	Y'_{10low}	Y'_{10high}	Y'_{11low}	Y'_{11high}	Y'_{12low}	Y'_{12high}	Y'_{13low}	Y'_{13high}
start 16:	+	Y'_{20low}	Y'_{20high}	Y'_{21low}	Y'_{21high}	Y'_{22low}	Y'_{22high}	Y'_{23low}	Y'_{23high}
start 24:	+	Y'_{30low}	Y'_{30high}	Y'_{31low}	Y'_{31high}	Y'_{32low}	Y'_{32high}	Y'_{33low}	Y'_{33high}

V4L2_PIX_FMT_Y16_BE (‘Y16 ‘| (1 << 31))

Grey-scale image

Description

This is a grey-scale image with a depth of 16 bits per pixel. The most significant byte is stored at lower memory addresses (big-endian).

Note: The actual sampling precision may be lower than 16 bits, for example 10 bits per pixel with values in range 0 to 1023.

Byte Order. Each cell is one byte.

start 0:	+	Y'_{00high}	Y'_{00low}	Y'_{01high}	Y'_{01low}	Y'_{02high}	Y'_{02low}	Y'_{03high}	Y'_{03low}
start 8:	+	Y'_{10high}	Y'_{10low}	Y'_{11high}	Y'_{11low}	Y'_{12high}	Y'_{12low}	Y'_{13high}	Y'_{13low}
start 16:	+	Y'_{20high}	Y'_{20low}	Y'_{21high}	Y'_{21low}	Y'_{22high}	Y'_{22low}	Y'_{23high}	Y'_{23low}
start 24:	+	Y'_{30high}	Y'_{30low}	Y'_{31high}	Y'_{31low}	Y'_{32high}	Y'_{32low}	Y'_{33high}	Y'_{33low}

V4L2_PIX_FMT_Y8I ('Y8I ')

Interleaved grey-scale image, e.g. from a stereo-pair

Description

This is a grey-scale image with a depth of 8 bits per pixel, but with pixels from 2 sources interleaved. Each pixel is stored in a 16-bit word. E.g. the R200 RealSense camera stores pixel from the left sensor in lower and from the right sensor in the higher 8 bits.

Byte Order. Each cell is one byte.

start + 0:	Y'$_{00left}$	Y'$_{00right}$	Y'$_{01left}$	Y'$_{01right}$	Y'$_{02left}$	Y'$_{02right}$	Y'$_{03left}$	Y'$_{03right}$
start + 8:	Y'$_{10left}$	Y'$_{10right}$	Y'$_{11left}$	Y'$_{11right}$	Y'$_{12left}$	Y'$_{12right}$	Y'$_{13left}$	Y'$_{13right}$
start + 16:	Y'$_{20left}$	Y'$_{20right}$	Y'$_{21left}$	Y'$_{21right}$	Y'$_{22left}$	Y'$_{22right}$	Y'$_{23left}$	Y'$_{23right}$
start + 24:	Y'$_{30left}$	Y'$_{30right}$	Y'$_{31left}$	Y'$_{31right}$	Y'$_{32left}$	Y'$_{32right}$	Y'$_{33left}$	Y'$_{33right}$

V4L2_PIX_FMT_Y12I ('Y12I')

Interleaved grey-scale image, e.g. from a stereo-pair

Description

This is a grey-scale image with a depth of 12 bits per pixel, but with pixels from 2 sources interleaved and bit-packed. Each pixel is stored in a 24-bit word in the little-endian order. On a little-endian machine these pixels can be deinterlaced using

```
__u8 *buf;
left0 = 0xfff & *(__u16 *)buf;
right0 = *(__u16 *)(buf + 1) >> 4;
```

Bit-packed representation. pixels cross the byte boundary and have a ratio of 3 bytes for each interleaved pixel.

Y'$_{0left[7:0]}$	Y'$_{0right[3:0]}$Y'$_{0left[11:8]}$	Y'$_{0right[11:4]}$

V4L2_PIX_FMT_UV8 ('UV8')

UV plane interleaved

Description

In this format there is no Y plane, Only CbCr plane. ie (UV interleaved)

Byte Order. Each cell is one byte.

start + 0:	Cb_{00}	Cr_{00}	Cb_{01}	Cr_{01}
start + 4:	Cb_{10}	Cr_{10}	Cb_{11}	Cr_{11}
start + 8:	Cb_{20}	Cr_{20}	Cb_{21}	Cr_{21}
start + 12:	Cb_{30}	Cr_{30}	Cb_{31}	Cr_{31}

V4L2_PIX_FMT_YUYV ('YUYV')

Packed format with ½ horizontal chroma resolution, also known as YUV 4:2:2

Description

In this format each four bytes is two pixels. Each four bytes is two Y's, a Cb and a Cr. Each Y goes to one of the pixels, and the Cb and Cr belong to both pixels. As you can see, the Cr and Cb components have half the horizontal resolution of the Y component. `V4L2_PIX_FMT_YUYV` is known in the Windows environment as YUY2.

Byte Order. Each cell is one byte.

start + 0:	Y'_{00}	Cb_{00}	Y'_{01}	Cr_{00}	Y'_{02}	Cb_{01}	Y'_{03}	Cr_{01}
start + 8:	Y'_{10}	Cb_{10}	Y'_{11}	Cr_{10}	Y'_{12}	Cb_{11}	Y'_{13}	Cr_{11}
start + 16:	Y'_{20}	Cb_{20}	Y'_{21}	Cr_{20}	Y'_{22}	Cb_{21}	Y'_{23}	Cr_{21}
start + 24:	Y'_{30}	Cb_{30}	Y'_{31}	Cr_{30}	Y'_{32}	Cb_{31}	Y'_{33}	Cr_{31}

Color Sample Location:

	0		1		2		3
0	Y	C	Y		Y	C	Y
1	Y	C	Y		Y	C	Y
2	Y	C	Y		Y	C	Y
3	Y	C	Y		Y	C	Y

V4L2_PIX_FMT_UYVY ('UYVY')

Variation of V4L2_PIX_FMT_YUYV with different order of samples in memory

Description

In this format each four bytes is two pixels. Each four bytes is two Y's, a Cb and a Cr. Each Y goes to one of the pixels, and the Cb and Cr belong to both pixels. As you can see, the Cr and Cb components have half the horizontal resolution of the Y component.

Byte Order. Each cell is one byte.

start + 0:	Cb_{00}	Y'_{00}	Cr_{00}	Y'_{01}	Cb_{01}	Y'_{02}	Cr_{01}	Y'_{03}
start + 8:	Cb_{10}	Y'_{10}	Cr_{10}	Y'_{11}	Cb_{11}	Y'_{12}	Cr_{11}	Y'_{13}
start + 16:	Cb_{20}	Y'_{20}	Cr_{20}	Y'_{21}	Cb_{21}	Y'_{22}	Cr_{21}	Y'_{23}
start + 24:	Cb_{30}	Y'_{30}	Cr_{30}	Y'_{31}	Cb_{31}	Y'_{32}	Cr_{31}	Y'_{33}

Color Sample Location:

	0		1	2		3
0	Y	C	Y	Y	C	Y
1	Y	C	Y	Y	C	Y
2	Y	C	Y	Y	C	Y
3	Y	C	Y	Y	C	Y

V4L2_PIX_FMT_YVYU ('YVYU')

Variation of V4L2_PIX_FMT_YUYV with different order of samples in memory

Description

In this format each four bytes is two pixels. Each four bytes is two Y's, a Cb and a Cr. Each Y goes to one of the pixels, and the Cb and Cr belong to both pixels. As you can see, the Cr and Cb components have half the horizontal resolution of the Y component.

Byte Order. Each cell is one byte.

start + 0:	Y'_{00}	Cr_{00}	Y'_{01}	Cb_{00}	Y'_{02}	Cr_{01}	Y'_{03}	Cb_{01}
start + 8:	Y'_{10}	Cr_{10}	Y'_{11}	Cb_{10}	Y'_{12}	Cr_{11}	Y'_{13}	Cb_{11}
start + 16:	Y'_{20}	Cr_{20}	Y'_{21}	Cb_{20}	Y'_{22}	Cr_{21}	Y'_{23}	Cb_{21}
start + 24:	Y'_{30}	Cr_{30}	Y'_{31}	Cb_{30}	Y'_{32}	Cr_{31}	Y'_{33}	Cb_{31}

Color Sample Location:

	0		1	2		3
0	Y	C	Y	Y	C	Y
1	Y	C	Y	Y	C	Y
2	Y	C	Y	Y	C	Y
3	Y	C	Y	Y	C	Y

V4L2_PIX_FMT_VYUY ('VYUY')

Variation of `V4L2_PIX_FMT_YUYV` with different order of samples in memory

Description

In this format each four bytes is two pixels. Each four bytes is two Y's, a Cb and a Cr. Each Y goes to one of the pixels, and the Cb and Cr belong to both pixels. As you can see, the Cr and Cb components have half the horizontal resolution of the Y component.

Byte Order. Each cell is one byte.

start + 0:	Cr_{00}	Y'_{00}	Cb_{00}	Y'_{01}	Cr_{01}	Y'_{02}	Cb_{01}	Y'_{03}
start + 8:	Cr_{10}	Y'_{10}	Cb_{10}	Y'_{11}	Cr_{11}	Y'_{12}	Cb_{11}	Y'_{13}
start + 16:	Cr_{20}	Y'_{20}	Cb_{20}	Y'_{21}	Cr_{21}	Y'_{22}	Cb_{21}	Y'_{23}
start + 24:	Cr_{30}	Y'_{30}	Cb_{30}	Y'_{31}	Cr_{31}	Y'_{32}	Cb_{31}	Y'_{33}

Color Sample Location:

	0		1		2	3
0	Y	C	Y	Y	C	Y
1	Y	C	Y	Y	C	Y
2	Y	C	Y	Y	C	Y
3	Y	C	Y	Y	C	Y

V4L2_PIX_FMT_Y41P ('Y41P')

Format with ¼ horizontal chroma resolution, also known as YUV 4:1:1

Description

In this format each 12 bytes is eight pixels. In the twelve bytes are two CbCr pairs and eight Y' s. The first CbCr pair goes with the first four Y' s, and the second CbCr pair goes with the other four Y' s. The Cb and Cr components have one fourth the horizontal resolution of the Y component.

Do not confuse this format with *V4L2_PIX_FMT_YUV411P*. Y41P is derived from "YUV 4:1:1 *packed*", while YUV411P stands for "YUV 4:1:1 *planar*".

Byte Order. Each cell is one byte.

start + 0:	Cb_{00}	Y'_{00}	Cr_{00}	Y'_{01}	Cb_{01}	Y'_{02}	Cr_{01}	Y'_{03}	Y'_{04}	Y'_{05}	Y'_{06}	Y'_{07}
start + 12:	Cb_{10}	Y'_{10}	Cr_{10}	Y'_{11}	Cb_{11}	Y'_{12}	Cr_{11}	Y'_{13}	Y'_{14}	Y'_{15}	Y'_{16}	Y'_{17}
start + 24:	Cb_{20}	Y'_{20}	Cr_{20}	Y'_{21}	Cb_{21}	Y'_{22}	Cr_{21}	Y'_{23}	Y'_{24}	Y'_{25}	Y'_{26}	Y'_{27}
start + 36:	Cb_{30}	Y'_{30}	Cr_{30}	Y'_{31}	Cb_{31}	Y'_{32}	Cr_{31}	Y'_{33}	Y'_{34}	Y'_{35}	Y'_{36}	Y'_{37}

Color Sample Location:

	0	1		2	3	4	5		6	7
0	Y	Y	C	Y	Y	Y	Y	C	Y	Y
1	Y	Y	C	Y	Y	Y	Y	C	Y	Y
2	Y	Y	C	Y	Y	Y	Y	C	Y	Y
3	Y	Y	C	Y	Y	Y	Y	C	Y	Y

V4L2_PIX_FMT_YVU420 ('YV12'), V4L2_PIX_FMT_YUV420 ('YU12')

V4L2_PIX_FMT_YUV420 Planar formats with ½ horizontal and vertical chroma resolution, also known as YUV 4:2:0

Description

These are planar formats, as opposed to a packed format. The three components are separated into three sub- images or planes. The Y plane is first. The Y plane has one byte per pixel. For `V4L2_PIX_FMT_YVU420`, the Cr plane immediately follows the Y plane in memory. The Cr plane is half the width and half the height of the Y plane (and of the image). Each Cr belongs to four pixels, a two-by-two square of the image. For example, Cr_0 belongs to Y'_{00}, Y'_{01}, Y'_{10}, and Y'_{11}. Following the Cr plane is the Cb plane, just like the Cr plane. `V4L2_PIX_FMT_YUV420` is the same except the Cb plane comes first, then the Cr plane.

If the Y plane has pad bytes after each row, then the Cr and Cb planes have half as many pad bytes after their rows. In other words, two Cx rows (including padding) is exactly as long as one Y row (including padding).

Byte Order. Each cell is one byte.

start + 0:	Y'$_{00}$	Y'$_{01}$	Y'$_{02}$	Y'$_{03}$
start + 4:	Y'$_{10}$	Y'$_{11}$	Y'$_{12}$	Y'$_{13}$
start + 8:	Y'$_{20}$	Y'$_{21}$	Y'$_{22}$	Y'$_{23}$
start + 12:	Y'$_{30}$	Y'$_{31}$	Y'$_{32}$	Y'$_{33}$
start + 16:	Cr_{00}	Cr_{01}		
start + 18:	Cr_{10}	Cr_{11}		
start + 20:	Cb_{00}	Cb_{01}		
start + 22:	Cb_{10}	Cb_{11}		

Color Sample Location:

	0		1	2		3
0	Y		Y	Y		Y
		C			C	
1	Y		Y	Y		Y
2	Y		Y	Y		Y
		C			C	
3	Y		Y	Y		Y

V4L2_PIX_FMT_YUV420M ('YM12'), V4L2_PIX_FMT_YVU420M ('YM21')

V4L2_PIX_FMT_YVU420M Variation of `V4L2_PIX_FMT_YUV420` and `V4L2_PIX_FMT_YVU420` with planes non contiguous in memory.

Description

This is a multi-planar format, as opposed to a packed format. The three components are separated into three sub-images or planes.

The Y plane is first. The Y plane has one byte per pixel. For `V4L2_PIX_FMT_YUV420M` the Cb data constitutes the second plane which is half the width and half the height of the Y plane (and of the image). Each Cb belongs to four pixels, a two-by-two square of the image. For example, Cb_0 belongs to Y'$_{00}$, Y'$_{01}$, Y'$_{10}$, and Y'$_{11}$. The Cr data, just like the Cb plane, is in the third plane.

`V4L2_PIX_FMT_YVU420M` is the same except the Cr data is stored in the second plane and the Cb data in the third plane.

If the Y plane has pad bytes after each row, then the Cb and Cr planes have half as many pad bytes after their rows. In other words, two Cx rows (including padding) is exactly as long as one Y row (including padding).

`V4L2_PIX_FMT_YUV420M` and `V4L2_PIX_FMT_YVU420M` are intended to be used only in drivers and applications that support the multi-planar API, described in *Single- and multi-planar APIs*.

Byte Order. Each cell is one byte.

start0 + 0:	Y'_{00}	Y'_{01}	Y'_{02}	Y'_{03}
start0 + 4:	Y'_{10}	Y'_{11}	Y'_{12}	Y'_{13}
start0 + 8:	Y'_{20}	Y'_{21}	Y'_{22}	Y'_{23}
start0 + 12:	Y'_{30}	Y'_{31}	Y'_{32}	Y'_{33}
start1 + 0:	Cb_{00}	Cb_{01}		
start1 + 2:	Cb_{10}	Cb_{11}		
start2 + 0:	Cr_{00}	Cr_{01}		
start2 + 2:	Cr_{10}	Cr_{11}		

Color Sample Location:

	0		1	2		3
0	Y		Y	Y		Y
		C			C	
1	Y		Y	Y		Y
2	Y		Y	Y		Y
		C			C	
3	Y		Y	Y		Y

V4L2_PIX_FMT_YUV422M ('YM16'), V4L2_PIX_FMT_YVU422M ('YM61')

V4L2_PIX_FMT_YVU422M Planar formats with ½ horizontal resolution, also known as YUV and YVU 4:2:2

Description

This is a multi-planar format, as opposed to a packed format. The three components are separated into three sub-images or planes.

The Y plane is first. The Y plane has one byte per pixel. For `V4L2_PIX_FMT_YUV422M` the Cb data constitutes the second plane which is half the width of the Y plane (and of the image). Each Cb belongs to two pixels. For example, Cb_0 belongs to Y'_{00}, Y'_{01}. The Cr data, just like the Cb plane, is in the third plane.

`V4L2_PIX_FMT_YVU422M` is the same except the Cr data is stored in the second plane and the Cb data in the third plane.

If the Y plane has pad bytes after each row, then the Cb and Cr planes have half as many pad bytes after their rows. In other words, two Cx rows (including padding) is exactly as long as one Y row (including padding).

`V4L2_PIX_FMT_YUV422M` and `V4L2_PIX_FMT_YVU422M` are intended to be used only in drivers and applications that support the multi-planar API, described in *Single- and multi-planar APIs*.

Byte Order. Each cell is one byte.

start0 + 0:	Y'_{00}	Y'_{01}	Y'_{02}	Y'_{03}
start0 + 4:	Y'_{10}	Y'_{11}	Y'_{12}	Y'_{13}
start0 + 8:	Y'_{20}	Y'_{21}	Y'_{22}	Y'_{23}
start0 + 12:	Y'_{30}	Y'_{31}	Y'_{32}	Y'_{33}
start1 + 0:	Cb_{00}	Cb_{01}		
start1 + 2:	Cb_{10}	Cb_{11}		
start1 + 4:	Cb_{20}	Cb_{21}		
start1 + 6:	Cb_{30}	Cb_{31}		
start2 + 0:	Cr_{00}	Cr_{01}		
start2 + 2:	Cr_{10}	Cr_{11}		
start2 + 4:	Cr_{20}	Cr_{21}		
start2 + 6:	Cr_{30}	Cr_{31}		

Color Sample Location:

	0		1	2		3
0	Y	C	Y	Y	C	Y
1	Y	C	Y	Y	C	Y
2	Y	C	Y	Y	C	Y
3	Y	C	Y	Y	C	Y

V4L2_PIX_FMT_YUV444M ('YM24'), V4L2_PIX_FMT_YVU444M ('YM42')

V4L2_PIX_FMT_YVU444M Planar formats with full horizontal resolution, also known as YUV and YVU 4:4:4

Description

This is a multi-planar format, as opposed to a packed format. The three components are separated into three sub-images or planes.

The Y plane is first. The Y plane has one byte per pixel. For `V4L2_PIX_FMT_YUV444M` the Cb data constitutes the second plane which is the same width and height as the Y plane (and as the image). The Cr data, just like the Cb plane, is in the third plane.

`V4L2_PIX_FMT_YVU444M` is the same except the Cr data is stored in the second plane and the Cb data in the third plane.

If the Y plane has pad bytes after each row, then the Cb and Cr planes have the same number of pad bytes after their rows.

`V4L2_PIX_FMT_YUV444M` and `V4L2_PIX_FMT_YUV444M` are intended to be used only in drivers and applications that support the multi-planar API, described in *Single- and multi-planar APIs*.

Byte Order. Each cell is one byte.

start0 + 0:	Y'_{00}	Y'_{01}	Y'_{02}	Y'_{03}
start0 + 4:	Y'_{10}	Y'_{11}	Y'_{12}	Y'_{13}
start0 + 8:	Y'_{20}	Y'_{21}	Y'_{22}	Y'_{23}
start0 + 12:	Y'_{30}	Y'_{31}	Y'_{32}	Y'_{33}
start1 + 0:	Cb_{00}	Cb_{01}	Cb_{02}	Cb_{03}
start1 + 4:	Cb_{10}	Cb_{11}	Cb_{12}	Cb_{13}
start1 + 8:	Cb_{20}	Cb_{21}	Cb_{22}	Cb_{23}
start1 + 12:	Cb_{20}	Cb_{21}	Cb_{32}	Cb_{33}
start2 + 0:	Cr_{00}	Cr_{01}	Cr_{02}	Cr_{03}
start2 + 4:	Cr_{10}	Cr_{11}	Cr_{12}	Cr_{13}
start2 + 8:	Cr_{20}	Cr_{21}	Cr_{22}	Cr_{23}
start2 + 12:	Cr_{30}	Cr_{31}	Cr_{32}	Cr_{33}

Color Sample Location:

	0	1	2	3
0	YC	YC	YC	YC
1	YC	YC	YC	YC
2	YC	YC	YC	YC
3	YC	YC	YC	YC

V4L2_PIX_FMT_YVU410 ('YVU9'), V4L2_PIX_FMT_YUV410 ('YUV9')

V4L2_PIX_FMT_YUV410 Planar formats with ¼ horizontal and vertical chroma resolution, also known as YUV 4:1:0

Description

These are planar formats, as opposed to a packed format. The three components are separated into three sub-images or planes. The Y plane is first. The Y plane has one byte per pixel. For `V4L2_PIX_FMT_YVU410`, the Cr plane immediately follows the Y plane in memory. The Cr plane is ¼ the width and ¼ the height of the Y plane (and of the image). Each Cr belongs to 16 pixels, a four-by-four square of the image. Following the Cr plane is the Cb plane, just like the Cr plane. `V4L2_PIX_FMT_YUV410` is the same, except the Cb plane comes first, then the Cr plane.

If the Y plane has pad bytes after each row, then the Cr and Cb planes have ¼ as many pad bytes after their rows. In other words, four Cx rows (including padding) are exactly as long as one Y row (including padding).

Byte Order. Each cell is one byte.

start + 0:	Y’ $_{00}$	Y’ $_{01}$	Y’ $_{02}$	Y’ $_{03}$
start + 4:	Y’ $_{10}$	Y’ $_{11}$	Y’ $_{12}$	Y’ $_{13}$
start + 8:	Y’ $_{20}$	Y’ $_{21}$	Y’ $_{22}$	Y’ $_{23}$
start + 12:	Y’ $_{30}$	Y’ $_{31}$	Y’ $_{32}$	Y’ $_{33}$
start + 16:	Cr_{00}			
start + 17:	Cb_{00}			

Color Sample Location:

	0	1		2	3
0	Y	Y		Y	Y
1	Y	Y		Y	Y
			C		
2	Y	Y		Y	Y
3	Y	Y		Y	Y

V4L2_PIX_FMT_YUV422P (‘422P’)

Format with ½ horizontal chroma resolution, also known as YUV 4:2:2. Planar layout as opposed to `V4L2_PIX_FMT_YUYV`

Description

This format is not commonly used. This is a planar version of the YUYV format. The three components are separated into three sub-images or planes. The Y plane is first. The Y plane has one byte per pixel. The Cb plane immediately follows the Y plane in memory. The Cb plane is half the width of the Y plane (and of the image). Each Cb belongs to two pixels. For example, Cb_0 belongs to Y’ $_{00}$, Y’ $_{01}$. Following the Cb plane is the Cr plane, just like the Cb plane.

If the Y plane has pad bytes after each row, then the Cr and Cb planes have half as many pad bytes after their rows. In other words, two Cx rows (including padding) is exactly as long as one Y row (including padding).

Byte Order. Each cell is one byte.

start + 0:	Y'_{00}	Y'_{01}	Y'_{02}	Y'_{03}
start + 4:	Y'_{10}	Y'_{11}	Y'_{12}	Y'_{13}
start + 8:	Y'_{20}	Y'_{21}	Y'_{22}	Y'_{23}
start + 12:	Y'_{30}	Y'_{31}	Y'_{32}	Y'_{33}
start + 16:	Cb_{00}	Cb_{01}		
start + 18:	Cb_{10}	Cb_{11}		
start + 20:	Cb_{20}	Cb_{21}		
start + 22:	Cb_{30}	Cb_{31}		
start + 24:	Cr_{00}	Cr_{01}		
start + 26:	Cr_{10}	Cr_{11}		
start + 28:	Cr_{20}	Cr_{21}		
start + 30:	Cr_{30}	Cr_{31}		

Color Sample Location:

	0		1	2		3
0	Y	C	Y	Y	C	Y
1	Y	C	Y	Y	C	Y
2	Y	C	Y	Y	C	Y
3	Y	C	Y	Y	C	Y

V4L2_PIX_FMT_YUV411P ('411P')

Format with ¼ horizontal chroma resolution, also known as YUV 4:1:1. Planar layout as opposed to `V4L2_PIX_FMT_Y41P`

Description

This format is not commonly used. This is a planar format similar to the 4:2:2 planar format except with half as many chroma. The three components are separated into three sub-images or planes. The Y plane is first. The Y plane has one byte per pixel. The Cb plane immediately follows the Y plane in memory. The Cb plane is ¼ the width of the Y plane (and of the image). Each Cb belongs to 4 pixels all on the same row. For example, Cb_0 belongs to Y'_{00}, Y'_{01}, Y'_{02} and Y'_{03}. Following the Cb plane is the Cr plane, just like the Cb plane.

If the Y plane has pad bytes after each row, then the Cr and Cb planes have ¼ as many pad bytes after their rows. In other words, four C x rows (including padding) is exactly as long as one Y row (including padding).

Byte Order. Each cell is one byte.

start + 0:	Y' $_{00}$	Y' $_{01}$	Y' $_{02}$	Y' $_{03}$
start + 4:	Y' $_{10}$	Y' $_{11}$	Y' $_{12}$	Y' $_{13}$
start + 8:	Y' $_{20}$	Y' $_{21}$	Y' $_{22}$	Y' $_{23}$
start + 12:	Y' $_{30}$	Y' $_{31}$	Y' $_{32}$	Y' $_{33}$
start + 16:	Cb_{00}			
start + 17:	Cb_{10}			
start + 18:	Cb_{20}			
start + 19:	Cb_{30}			
start + 20:	Cr_{00}			
start + 21:	Cr_{10}			
start + 22:	Cr_{20}			
start + 23:	Cr_{30}			

Color Sample Location:

	0	1		2	3
0	Y	Y	C	Y	Y
1	Y	Y	C	Y	Y
2	Y	Y	C	Y	Y
3	Y	Y	C	Y	Y

V4L2_PIX_FMT_NV12 ('NV12'), V4L2_PIX_FMT_NV21 ('NV21')

V4L2_PIX_FMT_NV21 Formats with ½ horizontal and vertical chroma resolution, also known as YUV 4:2:0. One luminance and one chrominance plane with alternating chroma samples as opposed to `V4L2_PIX_FMT_YVU420`

Description

These are two-plane versions of the YUV 4:2:0 format. The three components are separated into two sub-images or planes. The Y plane is first. The Y plane has one byte per pixel. For `V4L2_PIX_FMT_NV12`, a combined CbCr plane immediately follows the Y plane in memory. The CbCr plane is the same width, in bytes, as the Y plane (and of the image), but is half as tall in pixels. Each CbCr pair belongs to four pixels. For example, Cb_0/Cr_0 belongs to Y' $_{00}$, Y' $_{01}$, Y' $_{10}$, Y' $_{11}$. `V4L2_PIX_FMT_NV21` is the same except the Cb and Cr bytes are swapped, the CrCb plane starts with a Cr byte.

If the Y plane has pad bytes after each row, then the CbCr plane has as many pad bytes after its rows.

Byte Order. Each cell is one byte.

start + 0:	Y’ $_{00}$	Y’ $_{01}$	Y’ $_{02}$	Y’ $_{03}$
start + 4:	Y’ $_{10}$	Y’ $_{11}$	Y’ $_{12}$	Y’ $_{13}$
start + 8:	Y’ $_{20}$	Y’ $_{21}$	Y’ $_{22}$	Y’ $_{23}$
start + 12:	Y’ $_{30}$	Y’ $_{31}$	Y’ $_{32}$	Y’ $_{33}$
start + 16:	Cb_{00}	Cr_{00}	Cb_{01}	Cr_{01}
start + 20:	Cb_{10}	Cr_{10}	Cb_{11}	Cr_{11}

Color Sample Location:

	0		1	2		3
0	Y		Y	Y		Y
		C			C	
1	Y		Y	Y		Y
2	Y		Y	Y		Y
		C			C	
3	Y		Y	Y		Y

V4L2_PIX_FMT_NV12M (‘NM12’), V4L2_PIX_FMT_NV21M (‘NM21’), V4L2_PIX_FMT_NV12MT_16X16

V4L2_PIX_FMT_NV21M V4L2_PIX_FMT_NV12MT_16X16 Variation of `V4L2_PIX_FMT_NV12` and `V4L2_PIX_FMT_NV21` with planes non contiguous in memory.

Description

This is a multi-planar, two-plane version of the YUV 4:2:0 format. The three components are separated into two sub-images or planes. `V4L2_PIX_FMT_NV12M` differs from `V4L2_PIX_FMT_NV12` in that the two planes are non-contiguous in memory, i.e. the chroma plane do not necessarily immediately follows the luma plane. The luminance data occupies the first plane. The Y plane has one byte per pixel. In the second plane there is a chrominance data with alternating chroma samples. The CbCr plane is the same width, in bytes, as the Y plane (and of the image), but is half as tall in pixels. Each CbCr pair belongs to four pixels. For example, Cb_0/Cr_0 belongs to Y’ $_{00}$, Y’ $_{01}$, Y’ $_{10}$, Y’ $_{11}$. `V4L2_PIX_FMT_NV12MT_16X16` is the tiled version of `V4L2_PIX_FMT_NV12M` with 16x16 macroblock tiles. Here pixels are arranged in 16x16 2D tiles and tiles are arranged in linear order in memory. `V4L2_PIX_FMT_NV21M` is the same as `V4L2_PIX_FMT_NV12M` except the Cb and Cr bytes are swapped, the CrCb plane starts with a Cr byte.

`V4L2_PIX_FMT_NV12M` is intended to be used only in drivers and applications that support the multi-planar API, described in *Single- and multi-planar APIs*.

If the Y plane has pad bytes after each row, then the CbCr plane has as many pad bytes after its rows.

Byte Order. Each cell is one byte.

start0 + 0:	Y'$_{00}$	Y'$_{01}$	Y'$_{02}$	Y'$_{03}$
start0 + 4:	Y'$_{10}$	Y'$_{11}$	Y'$_{12}$	Y'$_{13}$
start0 + 8:	Y'$_{20}$	Y'$_{21}$	Y'$_{22}$	Y'$_{23}$
start0 + 12:	Y'$_{30}$	Y'$_{31}$	Y'$_{32}$	Y'$_{33}$
start1 + 0:	Cb_{00}	Cr_{00}	Cb_{01}	Cr_{01}
start1 + 4:	Cb_{10}	Cr_{10}	Cb_{11}	Cr_{11}

Color Sample Location:

	0		1	2		3
0	Y		Y	Y		Y
		C			C	
1	Y		Y	Y		Y
2	Y		Y	Y		Y
		C				C
3	Y		Y	Y		Y

V4L2_PIX_FMT_NV12MT ('TM12')

Formats with ½ horizontal and vertical chroma resolution. This format has two planes - one for luminance and one for chrominance. Chroma samples are interleaved. The difference to `V4L2_PIX_FMT_NV12` is the memory layout. Pixels are grouped in macroblocks of 64x32 size. The order of macroblocks in memory is also not standard.

Description

This is the two-plane versions of the YUV 4:2:0 format where data is grouped into 64x32 macroblocks. The three components are separated into two sub-images or planes. The Y plane has one byte per pixel and pixels are grouped into 64x32 macroblocks. The CbCr plane has the same width, in bytes, as the Y plane (and the image), but is half as tall in pixels. The chroma plane is also grouped into 64x32 macroblocks.

Width of the buffer has to be aligned to the multiple of 128, and height alignment is 32. Every four adjacent buffers - two horizontally and two vertically are grouped together and are located in memory in Z or flipped Z order.

Layout of macroblocks in memory is presented in the following figure.

The requirement that width is multiple of 128 is implemented because, the Z shape cannot be cut in half horizontally. In case the vertical resolution of macroblocks is odd then the last row of macroblocks is arranged in a linear order.

In case of chroma the layout is identical. Cb and Cr samples are interleaved. Height of the buffer is aligned to 32.

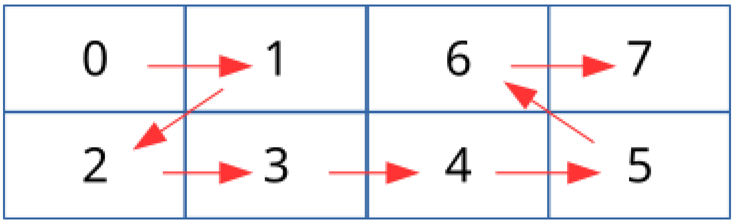

Fig. 4: V4L2_PIX_FMT_NV12MT macroblock Z shape memory layout

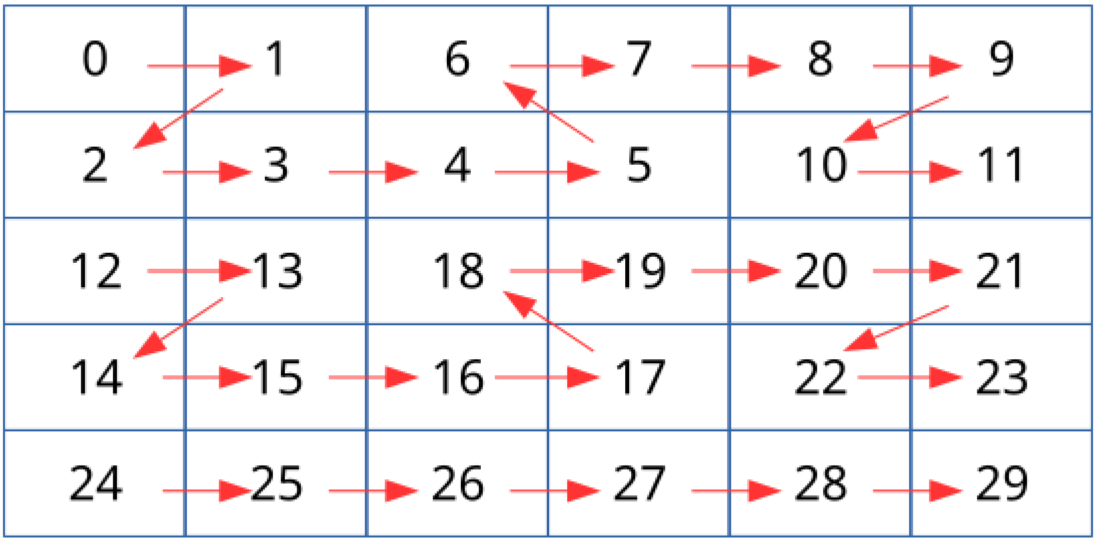

Fig. 5: Example V4L2_PIX_FMT_NV12MT memory layout of macroblocks

Memory layout of macroblocks of `V4L2_PIX_FMT_NV12MT` format in most extreme case.

V4L2_PIX_FMT_NV16 ('NV16'), V4L2_PIX_FMT_NV61 ('NV61')

V4L2_PIX_FMT_NV61 Formats with ½ horizontal chroma resolution, also known as YUV 4:2:2. One luminance and one chrominance plane with alternating chroma samples as opposed to `V4L2_PIX_FMT_YVU420`

Description

These are two-plane versions of the YUV 4:2:2 format. The three components are separated into two sub-images or planes. The Y plane is first. The Y plane has one byte per pixel. For `V4L2_PIX_FMT_NV16`, a combined CbCr plane immediately follows the Y plane in memory. The CbCr plane is the same width and height, in bytes, as the Y plane (and of the image). Each CbCr pair belongs to two pixels. For example, Cb_0/Cr_0 belongs to Y'_{00}, Y'_{01}. `V4L2_PIX_FMT_NV61` is the same except the Cb and Cr bytes are swapped, the CrCb plane starts with a Cr byte.

If the Y plane has pad bytes after each row, then the CbCr plane has as many pad bytes after its rows.

Byte Order. Each cell is one byte.

start + 0:	Y'_{00}	Y'_{01}	Y'_{02}	Y'_{03}
start + 4:	Y'_{10}	Y'_{11}	Y'_{12}	Y'_{13}
start + 8:	Y'_{20}	Y'_{21}	Y'_{22}	Y'_{23}
start + 12:	Y'_{30}	Y'_{31}	Y'_{32}	Y'_{33}
start + 16:	Cb_{00}	Cr_{00}	Cb_{01}	Cr_{01}
start + 20:	Cb_{10}	Cr_{10}	Cb_{11}	Cr_{11}
start + 24:	Cb_{20}	Cr_{20}	Cb_{21}	Cr_{21}
start + 28:	Cb_{30}	Cr_{30}	Cb_{31}	Cr_{31}

Color Sample Location:

	0		1	2		3
0	Y		Y	Y		Y
		C			C	
1	Y		Y	Y		Y
		C			C	
2	Y		Y	Y		Y
		C			C	
3	Y		Y	Y		Y
		C			C	

V4L2_PIX_FMT_NV16M ('NM16'), V4L2_PIX_FMT_NV61M ('NM61')

V4L2_PIX_FMT_NV61M Variation of `V4L2_PIX_FMT_NV16` and `V4L2_PIX_FMT_NV61` with planes non contiguous in memory.

Description

This is a multi-planar, two-plane version of the YUV 4:2:2 format. The three components are separated into two sub-images or planes. `V4L2_PIX_FMT_NV16M` differs from `V4L2_PIX_FMT_NV16` in that the two planes are non-contiguous in memory, i.e. the chroma plane does not necessarily immediately follow the luma plane. The luminance data occupies the first plane. The Y plane has one byte per pixel. In the second plane there is chrominance data with alternating chroma samples. The CbCr plane is the same width and height, in bytes, as the Y plane. Each CbCr pair belongs to two pixels. For example, Cb_0/Cr_0 belongs to Y'_{00}, Y'_{01}. `V4L2_PIX_FMT_NV61M` is the same as `V4L2_PIX_FMT_NV16M` except the Cb and Cr bytes are swapped, the CrCb plane starts with a Cr byte.

`V4L2_PIX_FMT_NV16M` and `V4L2_PIX_FMT_NV61M` are intended to be used only in drivers and applications that support the multi-planar API, described in *Single- and multi-planar APIs*.

Byte Order. Each cell is one byte.

start0 + 0:	Y'_{00}	Y'_{01}	Y'_{02}	Y'_{03}
start0 + 4:	Y'_{10}	Y'_{11}	Y'_{12}	Y'_{13}
start0 + 8:	Y'_{20}	Y'_{21}	Y'_{22}	Y'_{23}
start0 + 12:	Y'_{30}	Y'_{31}	Y'_{32}	Y'_{33}
start1 + 0:	Cb_{00}	Cr_{00}	Cb_{02}	Cr_{02}
start1 + 4:	Cb_{10}	Cr_{10}	Cb_{12}	Cr_{12}
start1 + 8:	Cb_{20}	Cr_{20}	Cb_{22}	Cr_{22}
start1 + 12:	Cb_{30}	Cr_{30}	Cb_{32}	Cr_{32}

Color Sample Location:

	0		1	2		3
0	Y		Y	Y		Y
		C			C	
1	Y		Y	Y		Y
		C			C	
2	Y		Y	Y		Y
		C			C	
3	Y		Y	Y		Y
		C			C	

V4L2_PIX_FMT_NV24 (‘NV24’), V4L2_PIX_FMT_NV42 (‘NV42’)

V4L2_PIX_FMT_NV42 Formats with full horizontal and vertical chroma resolutions, also known as YUV 4:4:4. One luminance and one chrominance plane with alternating chroma samples as opposed to `V4L2_PIX_FMT_YVU420`

Description

These are two-plane versions of the YUV 4:4:4 format. The three components are separated into two sub-images or planes. The Y plane is first, with each Y sample stored in one byte per pixel. For `V4L2_PIX_FMT_NV24`, a combined CbCr plane immediately follows the Y plane in memory. The CbCr plane has the same width and height, in pixels, as the Y plane (and the image). Each line contains one CbCr pair per pixel, with each Cb and Cr sample stored in one byte. `V4L2_PIX_FMT_NV42` is the same except that the Cb and Cr samples are swapped, the CrCb plane starts with a Cr sample.

If the Y plane has pad bytes after each row, then the CbCr plane has twice as many pad bytes after its rows.

Byte Order. Each cell is one byte.

start + 0:	Y'_{00}	Y'_{01}	Y'_{02}	Y'_{03}				
start + 4:	Y'_{10}	Y'_{11}	Y'_{12}	Y'_{13}				
start + 8:	Y'_{20}	Y'_{21}	Y'_{22}	Y'_{23}				
start + 12:	Y'_{30}	Y'_{31}	Y'_{32}	Y'_{33}				
start + 16:	Cb_{00}	Cr_{00}	Cb_{01}	Cr_{01}	Cb_{02}	Cr_{02}	Cb_{03}	Cr_{03}
start + 24:	Cb_{10}	Cr_{10}	Cb_{11}	Cr_{11}	Cb_{12}	Cr_{12}	Cb_{13}	Cr_{13}
start + 32:	Cb_{20}	Cr_{20}	Cb_{21}	Cr_{21}	Cb_{22}	Cr_{22}	Cb_{23}	Cr_{23}
start + 40:	Cb_{30}	Cr_{30}	Cb_{31}	Cr_{31}	Cb_{32}	Cr_{32}	Cb_{33}	Cr_{33}

V4L2_PIX_FMT_M420 ('M420')

Format with ½ horizontal and vertical chroma resolution, also known as YUV 4:2:0. Hybrid plane line-interleaved layout.

Description

M420 is a YUV format with ½ horizontal and vertical chroma subsampling (YUV 4:2:0). Pixels are organized as interleaved luma and chroma planes. Two lines of luma data are followed by one line of chroma data.

The luma plane has one byte per pixel. The chroma plane contains interleaved CbCr pixels subsampled by ½ in the horizontal and vertical directions. Each CbCr pair belongs to four pixels. For example, Cb_0/Cr_0 belongs to Y'_{00}, Y'_{01}, Y'_{10}, Y'_{11}.

All line lengths are identical: if the Y lines include pad bytes so do the CbCr lines.

Byte Order. Each cell is one byte.

start + 0:	Y'_{00}	Y'_{01}	Y'_{02}	Y'_{03}
start + 4:	Y'_{10}	Y'_{11}	Y'_{12}	Y'_{13}
start + 8:	Cb_{00}	Cr_{00}	Cb_{01}	Cr_{01}
start + 16:	Y'_{20}	Y'_{21}	Y'_{22}	Y'_{23}
start + 20:	Y'_{30}	Y'_{31}	Y'_{32}	Y'_{33}
start + 24:	Cb_{10}	Cr_{10}	Cb_{11}	Cr_{11}

Color Sample Location:

	0		1	2		3
0	Y		Y	Y		Y
		C			C	
1	Y		Y	Y		Y
2	Y		Y	Y		Y
		C			C	
3	Y		Y	Y		Y

HSV Formats

These formats store the color information of the image in a geometrical representation. The colors are mapped into a cylinder, where the angle is the HUE, the height is the VALUE and the distance to the center is the SATURATION. This is a very useful format for image segmentation algorithms.

Packed HSV formats

Description

The *hue* (h) is measured in degrees, the equivalence between degrees and LSBs depends on the hsv-encoding used, see *Colorspaces*. The *saturation* (s) and the *value* (v) are measured in percentage of the cylinder: 0 being the smallest value and 255 the maximum.

The values are packed in 24 or 32 bit formats.

Table 53: Packed HSV Image Formats

Identifier	Code		Byte 0 in memory								Byte 1								Byte 2								Byte 3							
		Bit	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
V4L2_PIX_FMT_HSV32	'HSV4'										h_7	h_6	h_5	h_4	h_3	h_2	h_1	h_0	s_7	s_6	s_5	s_4	s_3	s_2	s_1	s_0	v_7	v_6	v_5	v_4	v_3	v_2	v_1	v_0
V4L2_PIX_FMT_HSV24	'HSV3'		h_7	h_6	h_5	h_4	h_3	h_2	h_1	h_0	s_7	s_6	s_5	s_4	s_3	s_2	s_1	s_0	v_7	v_6	v_5	v_4	v_3	v_2	v_1	v_0								

Bit 7 is the most significant bit.

Depth Formats

Depth data provides distance to points, mapped onto the image plane

V4L2_PIX_FMT_INZI ('INZI')

Infrared 10-bit linked with Depth 16-bit images

Description

Proprietary multi-planar format used by Intel SR300 Depth cameras, comprise of Infrared image followed by Depth data. The pixel definition is 32-bpp, with the Depth and Infrared Data split into separate continuous planes of identical dimensions.

The first plane - Infrared data - is stored according to *V4L2_PIX_FMT_Y10* greyscale format. Each pixel is 16-bit cell, with actual data stored in the 10 LSBs with values in range 0 to 1023. The six remaining MSBs are padded with zeros.

The second plane provides 16-bit per-pixel Depth data arranged in *V4L2-PIX-FMT-Z16* format.

Frame Structure. Each cell is a 16-bit word with more significant data stored at higher memory address (byte order is little-endian).

$Ir_{0,0}$	$Ir_{0,1}$	$Ir_{0,2}$	…	…	…
…					
Infrared Data					
…					
…	…	…	$Ir_{n-1,n-3}$	$Ir_{n-1,n-2}$	$Ir_{n-1,n-1}$
$Depth_{0,0}$	$Depth_{0,1}$	$Depth_{0,2}$	…	…	…
…					
Depth Data					
…					
…	…	…	$Depth_{n-1,n-3}$	$Depth_{n-1,n-2}$	$Depth_{n-1,n-1}$

V4L2_PIX_FMT_Z16 ('Z16 ')

16-bit depth data with distance values at each pixel

Description

This is a 16-bit format, representing depth data. Each pixel is a distance to the respective point in the image coordinates. Distance unit can vary and has to be negotiated with the device separately. Each pixel is stored in a 16-bit word in the little endian byte order.

Byte Order. Each cell is one byte.

start + 0:	Z_{00low}	Z_{00high}	Z_{01low}	Z_{01high}	Z_{02low}	Z_{02high}	Z_{03low}	Z_{03high}
start + 8:	Z_{10low}	Z_{10high}	Z_{11low}	Z_{11high}	Z_{12low}	Z_{12high}	Z_{13low}	Z_{13high}
start + 16:	Z_{20low}	Z_{20high}	Z_{21low}	Z_{21high}	Z_{22low}	Z_{22high}	Z_{23low}	Z_{23high}
start + 24:	Z_{30low}	Z_{30high}	Z_{31low}	Z_{31high}	Z_{32low}	Z_{32high}	Z_{33low}	Z_{33high}

V4L2_PIX_FMT_CNF4 ('CNF4')

Depth sensor confidence information as a 4 bits per pixel packed array

Description

Proprietary format used by Intel RealSense Depth cameras containing depth confidence information in range 0-15 with 0 indicating that the sensor was unable to resolve any signal and 15 indicating maximum level of confidence for the specific sensor (actual error margins might change from sensor to sensor).

Every two consecutive pixels are packed into a single byte. Bits 0-3 of byte n refer to confidence value of depth pixel 2*n, bits 4-7 to confidence value of depth pixel 2*n+1.

Bit-packed representation.

$Y'_{01[3:0]}$(bits 7–4) $Y'_{00[3:0]}$(bits 3–0)	$Y'_{03[3:0]}$(bits 7–4) $Y'_{02[3:0]}$(bits 3–0)

Compressed Formats

Table 54: Compressed Image Formats

Identifier	Code	Details
V4L2_PIX_FMT_JPEG	'JPEG'	TBD. See also *VIDIOC_G_JPEGCOMP*, *VIDIOC_S_JPEGCOMP*.
V4L2_PIX_FMT_MPEG	'MPEG'	MPEG multiplexed stream. The actual format is determined by extended control V4L2_CID_MPEG_STREAM_TYPE, see *Codec Control IDs*.
V4L2_PIX_FMT_H264	'H264'	H264 Access Unit. The decoder expects one Access Unit per buffer. The encoder generates one Access Unit per buffer. If *ioctl VIDIOC_ENUM_FMT* reports V4L2_FMT_FLAG_CONTINUOUS_BYTESTREAM then the decoder has no requirements since it can parse all the information from the raw bytestream.
V4L2_PIX_FMT_H264_NO_SC	'AVC1'	H264 video elementary stream without start codes.
V4L2_PIX_FMT_H264_MVC	'M264'	H264 MVC video elementary stream.
V4L2_PIX_FMT_H264_SLICE	'S264'	H264 parsed slice data, including slice headers, either with or without the start code, as extracted from the H264 bitstream. This format is adapted for stateless video decoders that implement an H264 pipeline (using the *Video Memory-To-Memory Interface* and *Request API*). This pixelformat has two modifiers that must be set at least once through the V4L2_CID_MPEG_VIDEO_H264_DECODE_MODE and V4L2_CID_MPEG_VIDEO_H264_START_CODE controls. In addition, metadata associated with the frame to decode are required to be passed through the V4L2_CID_MPEG_VIDEO_H264_SPS, V4L2_CID_MPEG_VIDEO_H264_PPS, V4L2_CID_MPEG_VIDEO_H264_SCALING_MATRIX, V4L2_CID_MPEG_VIDEO_H264_SLICE_PARAMS and V4L2_CID_MPEG_VIDEO_H264_DECODE_PARAMS controls. See the *associated Codec Control IDs*. Exactly one output and one capture buffer must be provided for use with this pixel format. The output buffer must contain the appropriate number of macroblocks to decode a full corresponding frame to the matching capture buffer. The syntax for this format is documented in *ITU-T Rec. H.264 Specification (04/2017 Edition)*, section 7.3.2.8 "Slice layer without partitioning RBSP syntax" and the following

SDR Formats

These formats are used for *SDR* interface only.

V4L2_SDR_FMT_CU8 ('CU08')

Complex unsigned 8-bit IQ sample

Description

This format contains sequence of complex number samples. Each complex number consist two parts, called In-phase and Quadrature (IQ). Both I and Q are represented as a 8 bit unsigned number. I value comes first and Q value after that.

Byte Order. Each cell is one byte.

start + 0:	I'_0
start + 1:	Q'_0

V4L2_SDR_FMT_CU16LE ('CU16')

Complex unsigned 16-bit little endian IQ sample

Description

This format contains sequence of complex number samples. Each complex number consist two parts, called In-phase and Quadrature (IQ). Both I and Q are represented as a 16 bit unsigned little endian number. I value comes first and Q value after that.

Byte Order. Each cell is one byte.

start + 0:	$I'_{0[7:0]}$	$I'_{0[15:8]}$
start + 2:	$Q'_{0[7:0]}$	$Q'_{0[15:8]}$

V4L2_SDR_FMT_CS8 ('CS08')

Complex signed 8-bit IQ sample

Description

This format contains sequence of complex number samples. Each complex number consist two parts, called In-phase and Quadrature (IQ). Both I and Q are represented as a 8 bit signed number. I value comes first and Q value after that.

Byte Order. Each cell is one byte.

start + 0:	I'_0
start + 1:	Q'_0

V4L2_SDR_FMT_CS14LE ('CS14')

Complex signed 14-bit little endian IQ sample

Description

This format contains sequence of complex number samples. Each complex number consist two parts, called In-phase and Quadrature (IQ). Both I and Q are represented as a 14 bit signed little endian number. I value comes first and Q value after that. 14 bit value is stored in 16 bit space with unused high bits padded with 0.

Byte Order. Each cell is one byte.

start + 0:	$I'_{0[7:0]}$	$I'_{0[13:8]}$
start + 2:	$Q'_{0[7:0]}$	$Q'_{0[13:8]}$

V4L2_SDR_FMT_RU12LE ('RU12')

Real unsigned 12-bit little endian sample

Description

This format contains sequence of real number samples. Each sample is represented as a 12 bit unsigned little endian number. Sample is stored in 16 bit space with unused high bits padded with 0.

Byte Order. Each cell is one byte.

start + 0:	$I'_{0[7:0]}$	$I'_{0[11:8]}$

V4L2_SDR_FMT_PCU16BE ('PC16')

Planar complex unsigned 16-bit big endian IQ sample

Description

This format contains a sequence of complex number samples. Each complex number consist of two parts called In-phase and Quadrature (IQ). Both I and Q are represented as a 16 bit unsigned big endian number stored in 32 bit space. The remaining unused bits within the 32 bit space will be padded with 0. I value starts first and Q value starts at an offset equalling half of the buffer size (i.e.) offset = buffersize/2. Out of the 16 bits, bit 15:2 (14 bit) is data and bit 1:0 (2 bit) can be any value.

Byte Order. Each cell is one byte.

Offset:	Byte B0	Byte B1	Byte B2	Byte B3
start + 0:	I' 0[13:6]	I' 0[5:0]; B1[1:0]=pad	pad	pad
start + 4:	I' 1[13:6]	I' 1[5:0]; B1[1:0]=pad	pad	pad
…				
start + offset:	Q' 0[13:6]	Q' 0[5:0]; B1[1:0]=pad	pad	pad
start + offset + 4:	Q' 1[13:6]	Q' 1[5:0]; B1[1:0]=pad	pad	pad

V4L2_SDR_FMT_PCU18BE ('PC18')

Planar complex unsigned 18-bit big endian IQ sample

Description

This format contains a sequence of complex number samples. Each complex number consist of two parts called In-phase and Quadrature (IQ). Both I and Q are represented as a 18 bit unsigned big endian number stored in 32 bit space. The remaining unused bits within the 32 bit space will be padded with 0. I value starts first and Q value starts at an offset equalling half of the buffer size (i.e.) offset = buffersize/2. Out of the 18 bits, bit 17:2 (16 bit) is data and bit 1:0 (2 bit) can be any value.

Byte Order. Each cell is one byte.

Offset:	Byte B0	Byte B1	Byte B2	Byte B3
start + 0:	I' 0[17:10]	I' 0[9:2]	I' 0[1:0]; B2[5:0]=pad	pad
start + 4:	I' 1[17:10]	I' 1[9:2]	I' 1[1:0]; B2[5:0]=pad	pad
…				
start + offset:	Q' 0[17:10]	Q' 0[9:2]	Q' 0[1:0]; B2[5:0]=pad	pad
start + offset + 4:	Q' 1[17:10]	Q' 1[9:2]	Q' 1[1:0]; B2[5:0]=pad	pad

V4L2_SDR_FMT_PCU20BE ('PC20')

Planar complex unsigned 20-bit big endian IQ sample

Description

This format contains a sequence of complex number samples. Each complex number consist of two parts called In-phase and Quadrature (IQ). Both I and Q are represented as a 20 bit unsigned big endian number stored in 32 bit space. The remaining unused bits within the 32 bit space will be padded with 0. I value starts first and Q value starts at an offset equalling half of the buffer size (i.e.) offset = buffersize/2. Out of the 20 bits, bit 19:2 (18 bit) is data and bit 1:0 (2 bit) can be any value.

Byte Order. Each cell is one byte.

Offset:	Byte B0	Byte B1	Byte B2	Byte B3
start + 0:	I'$_{0[19:12]}$	I'$_{0[11:4]}$	I'$_{0[3:0]; B2[3:0]=pad}$	pad
start + 4:	I'$_{1[19:12]}$	I'$_{1[11:4]}$	I'$_{1[3:0]; B2[3:0]=pad}$	pad
…				
start + offset:	Q'$_{0[19:12]}$	Q'$_{0[11:4]}$	Q'$_{0[3:0]; B2[3:0]=pad}$	pad
start + offset + 4:	Q'$_{1[19:12]}$	Q'$_{1[11:4]}$	Q'$_{1[3:0]; B2[3:0]=pad}$	pad

Touch Formats

These formats are used for *Touch Devices* interface only.

V4L2_TCH_FMT_DELTA_TD16 ('TD16')

man V4L2_TCH_FMT_DELTA_TD16(2)

16-bit signed little endian Touch Delta

Description

This format represents delta data from a touch controller.

Delta values may range from -32768 to 32767. Typically the values will vary through a small range depending on whether the sensor is touched or not. The full value may be seen if one of the touchscreen nodes has a fault or the line is not connected.

Byte Order. Each cell is one byte.

start + 0:	D'00low	D'00high	D'01low	D'01high	D'02low	D'02high	D'03low	D'03high
start + 8:	D'10low	D'10high	D'11low	D'11high	D'12low	D'12high	D'13low	D'13high
start + 16:	D'20low	D'20high	D'21low	D'21high	D'22low	D'22high	D'23low	D'23high
start + 24:	D'30low	D'30high	D'31low	D'31high	D'32low	D'32high	D'33low	D'33high

V4L2_TCH_FMT_DELTA_TD08 ('TD08')

man V4L2_TCH_FMT_DELTA_TD08(2)

8-bit signed Touch Delta

Description

This format represents delta data from a touch controller.

Delta values may range from -128 to 127. Typically the values will vary through a small range depending on whether the sensor is touched or not. The full value may be seen if one of the touchscreen nodes has a fault or the line is not connected.

Byte Order. Each cell is one byte.

start + 0:	D'00	D'01	D'02	D'03
start + 4:	D'10	D'11	D'12	D'13
start + 8:	D'20	D'21	D'22	D'23
start + 12:	D'30	D'31	D'32	D'33

V4L2_TCH_FMT_TU16 ('TU16')

man V4L2_TCH_FMT_TU16(2)

16-bit unsigned little endian raw touch data

Description

This format represents unsigned 16-bit data from a touch controller.

This may be used for output for raw and reference data. Values may range from 0 to 65535.

Byte Order. Each cell is one byte.

start + 0:	R'00low	R'00high	R'01low	R'01high	R'02low	R'02high	R'03low	R'03high
start + 8:	R'10low	R'10high	R'11low	R'11high	R'12low	R'12high	R'13low	R'13high
start + 16:	R'20low	R'20high	R'21low	R'21high	R'22low	R'22high	R'23low	R'23high
start + 24:	R'30low	R'30high	R'31low	R'31high	R'32low	R'32high	R'33low	R'33high

V4L2_TCH_FMT_TU08 ('TU08')

man V4L2_TCH_FMT_TU08(2)

8-bit unsigned raw touch data

Description

This format represents unsigned 8-bit data from a touch controller.

This may be used for output for raw and reference data. Values may range from 0 to 255.

Byte Order. Each cell is one byte.

start + 0:	R'00	R'01	R'02	R'03
start + 4:	R'10	R'11	R'12	R'13
start + 8:	R'20	R'21	R'22	R'23
start + 12:	R'30	R'31	R'32	R'33

Metadata Formats

These formats are used for the *Metadata Interface* interface only.

V4L2_META_FMT_D4XX ('D4XX')

Intel D4xx UVC Cameras Metadata

Description

Intel D4xx (D435 and other) cameras include per-frame metadata in their UVC payload headers, following the Microsoft(R) UVC extension proposal [*1*]. That means, that the private D4XX metadata, following the standard UVC header, is organised in blocks. D4XX cameras implement several standard block types, proposed by Microsoft, and several proprietary ones. Supported standard metadata types are MetadataId_CaptureStats (ID 3), MetadataId_CameraExtrinsics (ID 4), and MetadataId_CameraIntrinsics (ID 5). For their description see [1]. This document describes proprietary metadata types, used by D4xx cameras.

V4L2_META_FMT_D4XX buffers follow the metadata buffer layout of V4L2_META_FMT_UVC with the only difference, that it also includes proprietary payload header data. D4xx cameras use bulk transfers and only send one payload per frame, therefore their headers cannot be larger than 255 bytes.

Below are proprietary Microsoft style metadata types, used by D4xx cameras, where all fields are in little endian order:

Table 55: D4xx metadata

Field	Description
Depth Control	
__u32 ID	0x80000000
__u32 Size	Size in bytes (currently 56)
__u32 Version	Version of this structure. The documentation herein corresponds to version xxx. The version number will be incremented when new fields are added.
__u32 Flags	A bitmask of flags: see [2] below
__u32 Gain	Gain value in internal units, same as the V4L2_CID_GAIN control, used to capture the frame
__u32 Exposure	Exposure time (in microseconds) used to capture the frame
__u32 Laser power	Power of the laser LED 0-360, used for depth measurement
__u32 AE mode	0: manual; 1: automatic exposure
__u32 Exposure priority	Exposure priority value: 0 - constant frame rate
__u32 AE ROI left	Left border of the AE Region of Interest (all ROI values are in pixels and lie between 0 and maximum width or height respectively)
__u32 AE ROI right	Right border of the AE Region of Interest
__u32 AE ROI top	Top border of the AE Region of Interest
__u32 AE ROI bottom	Bottom border of the AE Region of Interest
__u32 Preset	Preset selector value, default: 0, unless changed by the user
__u32 Laser mode	0: off, 1: on
Capture Timing	
__u32 ID	0x80000001
__u32 Size	Size in bytes (currently 40)
__u32 Version	Version of this structure. The documentation herein corresponds to version xxx. The version number will be incremented when new fields are added.
__u32 Flags	A bitmask of flags: see [3] below
__u32 Frame counter	Monotonically increasing counter
__u32 Optical time	Time in microseconds from the beginning of a frame till its middle
__u32 Readout time	Time, used to read out a frame in microseconds
__u32 Exposure time	Frame exposure time in microseconds
__u32 Frame interval	In microseconds = 1000000 / framerate
__u32 Pipe latency	Time in microseconds from start of frame to data in USB buffer
Configuration	
__u32 ID	0x80000002

continues on next page

Table 55 – continued from previous page

Field	Description
__u32 Size	Size in bytes (currently 40)
__u32 Version	Version of this structure. The documentation herein corresponds to version xxx. The version number will be incremented when new fields are added.
__u32 Flags	A bitmask of flags: see [4] below
__u8 Hardware type	Camera hardware version [5]
__u8 SKU ID	Camera hardware configuration [6]
__u32 Cookie	Internal synchronisation
__u16 Format	Image format code [7]
__u16 Width	Width in pixels
__u16 Height	Height in pixels
__u16 Framerate	Requested frame rate per second
__u16 Trigger	Byte 0: bit 0: depth and RGB are synchronised, bit 1: external trigger

[1] https://docs.microsoft.com/en-us/windows-hardware/drivers/stream/uvc-extensions-1-5

[2] Depth Control flags specify which fields are valid:

```
0x00000001 Gain
0x00000002 Exposure
0x00000004 Laser power
0x00000008 AE mode
0x00000010 Exposure priority
0x00000020 AE ROI
0x00000040 Preset
```

[3] Capture Timing flags specify which fields are valid:

```
0x00000001 Frame counter
0x00000002 Optical time
0x00000004 Readout time
0x00000008 Exposure time
0x00000010 Frame interval
0x00000020 Pipe latency
```

[4] Configuration flags specify which fields are valid:

```
0x00000001 Hardware type
0x00000002 SKU ID
0x00000004 Cookie
0x00000008 Format
0x00000010 Width
0x00000020 Height
0x00000040 Framerate
0x00000080 Trigger
0x00000100 Cal count
```

[5] Camera model:

```
0 DS5
1 IVCAM2
```

[6] 8-bit camera hardware configuration bitfield:

```
[1:0] depthCamera
      00: no depth
      01: standard depth
      10: wide depth
      11: reserved
[2]   depthIsActive - has a laser projector
[3]   RGB presence
[4]   Inertial Measurement Unit (IMU) presence
[5]   projectorType
      0: HPTG
      1: Princeton
[6]   0: a projector, 1: an LED
[7]   reserved
```

[7] Image format codes per video streaming interface:

Depth:

```
1 Z16
2 Z
```

Left sensor:

```
1 Y8
2 UYVY
3 R8L8
4 Calibration
5 W10
```

Fish Eye sensor:

```
1 RAW8
```

V4L2_META_FMT_IPU3_PARAMS ('ip3p'), V4L2_META_FMT_IPU3_3A ('ip3s')

3A statistics

The IPU3 ImgU 3A statistics accelerators collect different statistics over an input Bayer frame. Those statistics are obtained from the "ipu3-imgu [01] 3a stat" metadata capture video nodes, using the *v4l2_meta_format* interface. They are formatted as described by the *ipu3_uapi_stats_3a* structure.

The statistics collected are AWB (Auto-white balance) RGBS (Red, Green, Blue and Saturation measure) cells, AWB filter response, AF (Auto-focus) filter response, and AE (Auto-exposure) histogram.

The struct *ipu3_uapi_4a_config* saves all configurable parameters.

```
struct ipu3_uapi_stats_3a {
        struct ipu3_uapi_awb_raw_buffer awb_raw_buffer;
        struct ipu3_uapi_ae_raw_buffer_aligned ae_raw_buffer[IPU3_
→UAPI_MAX_STRIPES];
        struct ipu3_uapi_af_raw_buffer af_raw_buffer;
        struct ipu3_uapi_awb_fr_raw_buffer awb_fr_raw_buffer;
        struct ipu3_uapi_4a_config stats_4a_config;
        __u32 ae_join_buffers;
        __u8 padding[28];
        struct ipu3_uapi_stats_3a_bubble_info_per_stripe stats_3a_
→bubble_per_stripe;
        struct ipu3_uapi_ff_status stats_3a_status;
};
```

Pipeline parameters

The pipeline parameters are passed to the "ipu3-imgu [01] parameters" metadata output video nodes, using the *v4l2_meta_format* interface. They are formatted as described by the *ipu3_uapi_params* structure.

Both 3A statistics and pipeline parameters described here are closely tied to the underlying camera sub-system (CSS) APIs. They are usually consumed and produced by dedicated user space libraries that comprise the important tuning tools, thus freeing the developers from being bothered with the low level hardware and algorithm details.

```
struct ipu3_uapi_params {
        /* Flags which of the settings below are to be applied */
        struct ipu3_uapi_flags use;

        /* Accelerator cluster parameters */
        struct ipu3_uapi_acc_param acc_param;

        /* ISP vector address space parameters */
        struct ipu3_uapi_isp_lin_vmem_params lin_vmem_params;
        struct ipu3_uapi_isp_tnr3_vmem_params tnr3_vmem_params;
        struct ipu3_uapi_isp_xnr3_vmem_params xnr3_vmem_params;

        /* ISP data memory (DMEM) parameters */
        struct ipu3_uapi_isp_tnr3_params tnr3_dmem_params;
        struct ipu3_uapi_isp_xnr3_params xnr3_dmem_params;

        /* Optical black level compensation */
        struct ipu3_uapi_obgrid_param obgrid_param;
};
```

Intel IPU3 ImgU uAPI data types

struct **ipu3_uapi_grid_config**

Grid plane config

Definition

```
struct ipu3_uapi_grid_config {
  __u8 width;
  __u8 height;
  __u16 block_width_log2:3;
  __u16 block_height_log2:3;
  __u16 height_per_slice:8;
  __u16 x_start;
  __u16 y_start;
  __u16 x_end;
  __u16 y_end;
};
```

Members

width
Grid horizontal dimensions, in number of grid blocks(cells).

height
Grid vertical dimensions, in number of grid cells.

block_width_log2
Log2 of the width of each cell in pixels. for (2^3, 2^4, 2^5, 2^6, 2^7), values [3, 7].

block_height_log2
Log2 of the height of each cell in pixels. for (2^3, 2^4, 2^5, 2^6, 2^7), values [3, 7].

height_per_slice
The number of blocks in vertical axis per slice. Default 2.

x_start
X value of top left corner of Region of Interest(ROI).

y_start
Y value of top left corner of ROI

x_end
X value of bottom right corner of ROI

y_end
Y value of bottom right corner of ROI

Description

Due to the size of total amount of collected data, most statistics create a grid-based output, and the data is then divided into "slices".

struct **ipu3_uapi_awb_raw_buffer**

AWB raw buffer

Definition

```
struct ipu3_uapi_awb_raw_buffer {
  __u8 meta_data[IPU3_UAPI_AWB_MAX_BUFFER_SIZE] ;
};
```

Members

meta_data
buffer to hold auto white balance meta data which is the average values for each color channel.

struct **ipu3_uapi_awb_config_s**
AWB config

Definition

```
struct ipu3_uapi_awb_config_s {
  __u16 rgbs_thr_gr;
  __u16 rgbs_thr_r;
  __u16 rgbs_thr_gb;
  __u16 rgbs_thr_b;
  struct ipu3_uapi_grid_config grid;
};
```

Members

rgbs_thr_gr
gr threshold value.

rgbs_thr_r
Red threshold value.

rgbs_thr_gb
gb threshold value.

rgbs_thr_b
Blue threshold value.

grid
ipu3_uapi_grid_config, the default grid resolution is 16x16 cells.

Description

The threshold is a saturation measure range [0, 8191], 8191 is default. Values over threshold may be optionally rejected for averaging.

struct **ipu3_uapi_awb_config**
AWB config wrapper

Definition

```
struct ipu3_uapi_awb_config {
  struct ipu3_uapi_awb_config_s config ;
};
```

Members

config
config for auto white balance as defined by *ipu3_uapi_awb_config_s*

struct **ipu3_uapi_ae_raw_buffer**
AE global weighted histogram

Definition

```
struct ipu3_uapi_ae_raw_buffer {
  __u32 vals[IPU3_UAPI_AE_BINS * IPU3_UAPI_AE_COLORS];
};
```

Members

vals
Sum of IPU3_UAPI_AE_COLORS in cell

Description

Each histogram contains IPU3_UAPI_AE_BINS bins. Each bin has 24 bit unsigned for counting the number of the pixel.

struct **ipu3_uapi_ae_raw_buffer_aligned**
AE raw buffer

Definition

```
struct ipu3_uapi_ae_raw_buffer_aligned {
  struct ipu3_uapi_ae_raw_buffer buff ;
};
```

Members

buff
ipu3_uapi_ae_raw_buffer to hold full frame meta data.

struct **ipu3_uapi_ae_grid_config**
AE weight grid

Definition

```
struct ipu3_uapi_ae_grid_config {
  __u8 width;
  __u8 height;
  __u8 block_width_log2:4;
  __u8 block_height_log2:4;
  __u8 reserved0:5;
  __u8 ae_en:1;
  __u8 rst_hist_array:1;
  __u8 done_rst_hist_array:1;
  __u16 x_start;
  __u16 y_start;
  __u16 x_end;
  __u16 y_end;
};
```

Members

width
Grid horizontal dimensions. Value: [16, 32], default 16.

height
Grid vertical dimensions. Value: [16, 24], default 16.

block_width_log2
Log2 of the width of the grid cell, value: [3, 7].

block_height_log2
Log2 of the height of the grid cell, value: [3, 7]. default is 3 (cell size 8x8), 4 cell per grid.

reserved0
reserved

ae_en
0: does not write to *ipu3_uapi_ae_raw_buffer_aligned* array, 1: write normally.

rst_hist_array
write 1 to trigger histogram array reset.

done_rst_hist_array
flag for histogram array reset done.

x_start
X value of top left corner of ROI, default 0.

y_start
Y value of top left corner of ROI, default 0.

x_end
X value of bottom right corner of ROI

y_end
Y value of bottom right corner of ROI

Description

The AE block accumulates 4 global weighted histograms(R, G, B, Y) over a defined ROI within the frame. The contribution of each pixel into the histogram, defined by *ipu3_uapi_ae_weight_elem* LUT, is indexed by a grid.

struct **ipu3_uapi_ae_weight_elem**
AE weights LUT

Definition

```
struct ipu3_uapi_ae_weight_elem {
  __u32 cell0:4;
  __u32 cell1:4;
  __u32 cell2:4;
  __u32 cell3:4;
  __u32 cell4:4;
  __u32 cell5:4;
```

(continues on next page)

(continued from previous page)

```
  __u32 cell6:4;
  __u32 cell7:4;
};
```

Members

cell0
weighted histogram grid value.

cell1
weighted histogram grid value.

cell2
weighted histogram grid value.

cell3
weighted histogram grid value.

cell4
weighted histogram grid value.

cell5
weighted histogram grid value.

cell6
weighted histogram grid value.

cell7
weighted histogram grid value.

Description

Use weighted grid value to give a different contribution factor to each cell. Precision u4, range [0, 15].

struct **ipu3_uapi_ae_ccm**

AE coefficients for WB and CCM

Definition

```
struct ipu3_uapi_ae_ccm {
  __u16 gain_gr;
  __u16 gain_r;
  __u16 gain_b;
  __u16 gain_gb;
  __s16 mat[16];
};
```

Members

gain_gr
WB gain factor for the gr channels. Default 256.

gain_r
WB gain factor for the r channel. Default 256.

gain_b
WB gain factor for the b channel. Default 256.

gain_gb
WB gain factor for the gb channels. Default 256.

mat
4x4 matrix that transforms Bayer quad output from WB to RGB+Y.

Description

Default:
128, 0, 0, 0, 0, 128, 0, 0, 0, 0, 128, 0, 0, 0, 0, 128,

As part of the raw frame pre-process stage, the WB and color conversion need to be applied to expose the impact of these gain operations.

struct **ipu3_uapi_ae_config**
AE config

Definition

```
struct ipu3_uapi_ae_config {
  struct ipu3_uapi_ae_grid_config grid_cfg ;
  struct ipu3_uapi_ae_weight_elem weights[ IPU3_UAPI_AE_WEIGHTS] ;
  struct ipu3_uapi_ae_ccm ae_ccm ;
};
```

Members

grid_cfg
config for auto exposure statistics grid. See struct *ipu3_uapi_ae_grid_config*

weights
IPU3_UAPI_AE_WEIGHTS is based on 32x24 blocks in the grid. Each grid cell has a corresponding value in weights LUT called grid value, global histogram is updated based on grid value and pixel value.

ae_ccm
Color convert matrix pre-processing block.

Description

Calculate AE grid from image resolution, resample ae weights.

struct **ipu3_uapi_af_filter_config**
AF 2D filter for contrast measurements

Definition

```
struct ipu3_uapi_af_filter_config {
  struct {
    __u8 a1;
    __u8 a2;
    __u8 a3;
    __u8 a4;
  } y1_coeff_0;
  struct {
    __u8 a5;
```

(continues on next page)

(continued from previous page)

```
    __u8 a6;
    __u8 a7;
    __u8 a8;
  } y1_coeff_1;
  struct {
    __u8 a9;
    __u8 a10;
    __u8 a11;
    __u8 a12;
  } y1_coeff_2;
  __u32 y1_sign_vec;
  struct {
    __u8 a1;
    __u8 a2;
    __u8 a3;
    __u8 a4;
  } y2_coeff_0;
  struct {
    __u8 a5;
    __u8 a6;
    __u8 a7;
    __u8 a8;
  } y2_coeff_1;
  struct {
    __u8 a9;
    __u8 a10;
    __u8 a11;
    __u8 a12;
  } y2_coeff_2;
  __u32 y2_sign_vec;
  struct {
    __u8 y_gen_rate_gr;
    __u8 y_gen_rate_r;
    __u8 y_gen_rate_b;
    __u8 y_gen_rate_gb;
  } y_calc;
  struct {
    __u32 reserved0:8;
    __u32 y1_nf:4;
    __u32 reserved1:4;
    __u32 y2_nf:4;
    __u32 reserved2:12;
  } nf;
};
```

Members

y1_coeff_0
filter Y1, structure: 3x11, support both symmetry and anti-symmetry type. A12 is center, A1-A11 are neighbours. for analyzing low frequency content,

used to calculate sum of gradients in x direction.

y1_coeff_0.a1
filter1 coefficients A1, u8, default 0.

y1_coeff_0.a2
filter1 coefficients A2, u8, default 0.

y1_coeff_0.a3
filter1 coefficients A3, u8, default 0.

y1_coeff_0.a4
filter1 coefficients A4, u8, default 0.

y1_coeff_1
Struct

y1_coeff_1.a5
filter1 coefficients A5, u8, default 0.

y1_coeff_1.a6
filter1 coefficients A6, u8, default 0.

y1_coeff_1.a7
filter1 coefficients A7, u8, default 0.

y1_coeff_1.a8
filter1 coefficients A8, u8, default 0.

y1_coeff_2
Struct

y1_coeff_2.a9
filter1 coefficients A9, u8, default 0.

y1_coeff_2.a10
filter1 coefficients A10, u8, default 0.

y1_coeff_2.a11
filter1 coefficients A11, u8, default 0.

y1_coeff_2.a12
filter1 coefficients A12, u8, default 128.

y1_sign_vec
Each bit corresponds to one coefficient sign bit, 0: positive, 1: negative, default 0.

y2_coeff_0
Y2, same structure as Y1. For analyzing high frequency content.

y2_coeff_0.a1
filter2 coefficients A1, u8, default 0.

y2_coeff_0.a2
filter2 coefficients A2, u8, default 0.

y2_coeff_0.a3
filter2 coefficients A3, u8, default 0.

y2_coeff_0.a4
filter2 coefficients A4, u8, default 0.

y2_coeff_1
Struct

y2_coeff_1.a5
filter2 coefficients A5, u8, default 0.

y2_coeff_1.a6
filter2 coefficients A6, u8, default 0.

y2_coeff_1.a7
filter2 coefficients A7, u8, default 0.

y2_coeff_1.a8
filter2 coefficients A8, u8, default 0.

y2_coeff_2
Struct

y2_coeff_2.a9
filter1 coefficients A9, u8, default 0.

y2_coeff_2.a10
filter1 coefficients A10, u8, default 0.

y2_coeff_2.a11
filter1 coefficients A11, u8, default 0.

y2_coeff_2.a12
filter1 coefficients A12, u8, default 128.

y2_sign_vec
Each bit corresponds to one coefficient sign bit, 0: positive, 1: negative, default 0.

y_calc
Pre-processing that converts Bayer quad to RGB+Y values to be used for building histogram. Range [0, 32], default 8. Rule: y_gen_rate_gr + y_gen_rate_r + y_gen_rate_b + y_gen_rate_gb = 32 A single Y is calculated based on sum of Gr/R/B/Gb based on their contribution ratio.

y_calc.y_gen_rate_gr
Contribution ratio Gr for Y

y_calc.y_gen_rate_r
Contribution ratio R for Y

y_calc.y_gen_rate_b
Contribution ratio B for Y

y_calc.y_gen_rate_gb
Contribution ratio Gb for Y

nf
The shift right value that should be applied during the Y1/Y2 filter to make sure the total memory needed is 2 bytes per grid cell.

nf.reserved0
reserved

nf.y1_nf
Normalization factor for the convolution coeffs of y1, should be log2 of the sum of the abs values of the filter coeffs, default 7 (2^7 = 128).

nf.reserved1
reserved

nf.y2_nf
Normalization factor for y2, should be log2 of the sum of the abs values of the filter coeffs.

nf.reserved2
reserved

struct **ipu3_uapi_af_raw_buffer**
AF meta data

Definition

```
struct ipu3_uapi_af_raw_buffer {
  __u8 y_table[IPU3_UAPI_AF_Y_TABLE_MAX_SIZE] ;
};
```

Members

y_table
Each color component will be convolved separately with filter1 and filter2 and the result will be summed out and averaged for each cell.

struct **ipu3_uapi_af_config_s**
AF config

Definition

```
struct ipu3_uapi_af_config_s {
  struct ipu3_uapi_af_filter_config filter_config ;
  __u8 padding[4];
  struct ipu3_uapi_grid_config grid_cfg ;
};
```

Members

filter_config
AF uses Y1 and Y2 filters as configured in *ipu3_uapi_af_filter_config*

padding
paddings

grid_cfg
See *ipu3_uapi_grid_config*, default resolution 16x16. Use large grid size for large image and vice versa.

struct **ipu3_uapi_awb_fr_raw_buffer**
AWB filter response meta data

Definition

```
struct ipu3_uapi_awb_fr_raw_buffer {
  __u8 meta_data[IPU3_UAPI_AWB_FR_BAYER_TABLE_MAX_SIZE] ;
};
```

Members

meta_data
Statistics output on the grid after convolving with 1D filter.

struct **ipu3_uapi_awb_fr_config_s**
AWB filter response config

Definition

```
struct ipu3_uapi_awb_fr_config_s {
  struct ipu3_uapi_grid_config grid_cfg;
  __u8 bayer_coeff[6];
  __u16 reserved1;
  __u32 bayer_sign;
  __u8 bayer_nf;
  __u8 reserved2[7];
};
```

Members

grid_cfg
grid config, default 16x16.

bayer_coeff
1D Filter 1x11 center symmetry/anti-symmetry. coefficients defaults { 0, 0, 0, 0, 0, 128 }. Applied on whole image for each Bayer channel separately by a weighted sum of its 11x1 neighbors.

reserved1
reserved

bayer_sign
sign of filter coefficients, default 0.

bayer_nf
normalization factor for the convolution coeffs, to make sure total memory needed is within pre-determined range. NF should be the log2 of the sum of the abs values of the filter coeffs, range [7, 14], default 7.

reserved2
reserved

struct **ipu3_uapi_4a_config**
4A config

Definition

```
struct ipu3_uapi_4a_config {
  struct ipu3_uapi_awb_config_s awb_config ;
```

(continues on next page)

(continued from previous page)

```
  struct ipu3_uapi_ae_grid_config ae_grd_config;
  __u8 padding[20];
  struct ipu3_uapi_af_config_s af_config;
  struct ipu3_uapi_awb_fr_config_s awb_fr_config ;
};
```

Members

awb_config
ipu3_uapi_awb_config_s, default resolution 16x16

ae_grd_config
auto exposure statistics *ipu3_uapi_ae_grid_config*

padding
paddings

af_config
auto focus config *ipu3_uapi_af_config_s*

awb_fr_config
ipu3_uapi_awb_fr_config_s, default resolution 16x16

struct **ipu3_uapi_bubble_info**
Bubble info for host side debugging

Definition

```
struct ipu3_uapi_bubble_info {
  __u32 num_of_stripes ;
  __u8 padding[28];
  __u32 num_sets;
  __u8 padding1[28];
  __u32 size_of_set;
  __u8 padding2[28];
  __u32 bubble_size;
  __u8 padding3[28];
};
```

Members

num_of_stripes
A single frame is divided into several parts called stripes due to limitation on line buffer memory. The separation between the stripes is vertical. Each such stripe is processed as a single frame by the ISP pipe.

padding
padding bytes.

num_sets
number of sets.

padding1
padding bytes.

size_of_set
set size.

padding2
padding bytes.

bubble_size
is the amount of padding in the bubble expressed in "sets".

padding3
padding bytes.

struct **ipu3_uapi_ff_status**
Enable bits for each 3A fixed function

Definition

```
struct ipu3_uapi_ff_status {
  __u32 awb_en ;
  __u8 padding[28];
  __u32 ae_en;
  __u8 padding1[28];
  __u32 af_en;
  __u8 padding2[28];
  __u32 awb_fr_en;
  __u8 padding3[28];
};
```

Members

awb_en
auto white balance enable

padding
padding config

ae_en
auto exposure enable

padding1
padding config

af_en
auto focus enable

padding2
padding config

awb_fr_en
awb filter response enable bit

padding3
padding config

struct **ipu3_uapi_stats_3a**
3A statistics

Definition

```
struct ipu3_uapi_stats_3a {
  struct ipu3_uapi_awb_raw_buffer awb_raw_buffer;
  struct ipu3_uapi_ae_raw_buffer_aligned ae_raw_buffer[IPU3_UAPI_
→MAX_STRIPES];
  struct ipu3_uapi_af_raw_buffer af_raw_buffer;
  struct ipu3_uapi_awb_fr_raw_buffer awb_fr_raw_buffer;
  struct ipu3_uapi_4a_config stats_4a_config;
  __u32 ae_join_buffers;
  __u8 padding[28];
  struct ipu3_uapi_stats_3a_bubble_info_per_stripe stats_3a_bubble_
→per_stripe;
  struct ipu3_uapi_ff_status stats_3a_status;
};
```

Members

awb_raw_buffer
auto white balance meta data *ipu3_uapi_awb_raw_buffer*

ae_raw_buffer
auto exposure raw data *ipu3_uapi_ae_raw_buffer_aligned*

af_raw_buffer
ipu3_uapi_af_raw_buffer for auto focus meta data

awb_fr_raw_buffer
value as specified by *ipu3_uapi_awb_fr_raw_buffer*

stats_4a_config
4a statistics config as defined by *ipu3_uapi_4a_config*.

ae_join_buffers
1 to use ae_raw_buffer.

padding
padding config

stats_3a_bubble_per_stripe
a ipu3_uapi_stats_3a_bubble_info_per_stripe

stats_3a_status
3a statistics status set in *ipu3_uapi_ff_status*

struct **ipu3_uapi_bnr_static_config_wb_gains_config**
White balance gains

Definition

```
struct ipu3_uapi_bnr_static_config_wb_gains_config {
  __u16 gr;
  __u16 r;
  __u16 b;
  __u16 gb;
};
```

Members

gr
: white balance gain for Gr channel.

r
: white balance gain for R channel.

b
: white balance gain for B channel.

gb
: white balance gain for Gb channel.

Description

Precision u3.13, range [0, 8). White balance correction is done by applying a multiplicative gain to each color channels prior to BNR.

struct **ipu3_uapi_bnr_static_config_wb_gains_thr_config**
Threshold config

Definition

```
struct ipu3_uapi_bnr_static_config_wb_gains_thr_config {
  __u8 gr;
  __u8 r;
  __u8 b;
  __u8 gb;
};
```

Members

gr
: white balance threshold gain for Gr channel.

r
: white balance threshold gain for R channel.

b
: white balance threshold gain for B channel.

gb
: white balance threshold gain for Gb channel.

Description

Defines the threshold that specifies how different a defect pixel can be from its neighbors.(used by dynamic defect pixel correction sub block) Precision u4.4 range [0, 8].

struct **ipu3_uapi_bnr_static_config_thr_coeffs_config**
Noise model coefficients that controls noise threshold

Definition

```
struct ipu3_uapi_bnr_static_config_thr_coeffs_config {
  __u32 cf:13;
  __u32 reserved0:3;
  __u32 cg:5;
```

(continues on next page)

(continued from previous page)

```
  __u32 ci:5;
  __u32 reserved1:1;
  __u32 r_nf:5;
};
```

Members

cf
Free coefficient for threshold calculation, range [0, 8191], default 0.

reserved0
reserved

cg
Gain coefficient for threshold calculation, [0, 31], default 8.

ci
Intensity coefficient for threshold calculation. range [0, 0x1f] default 6. format: u3.2 (3 most significant bits represent whole number, 2 least significant bits represent the fractional part with each count representing 0.25) e.g. 6 in binary format is 00110, that translates to 1.5

reserved1
reserved

r_nf
Normalization shift value for r^2 calculation, range [12, 20] where r is a radius of pixel [row, col] from centor of sensor. default 14.

Description

Threshold used to distinguish between noise and details.

struct **ipu3_uapi_bnr_static_config_thr_ctrl_shd_config**
Shading config

Definition

```
struct ipu3_uapi_bnr_static_config_thr_ctrl_shd_config {
  __u8 gr;
  __u8 r;
  __u8 b;
  __u8 gb;
};
```

Members

gr
Coefficient defines lens shading gain approximation for gr channel

r
Coefficient defines lens shading gain approximation for r channel

b
Coefficient defines lens shading gain approximation for b channel

gb
Coefficient defines lens shading gain approximation for gb channel

Description

Parameters for noise model (NM) adaptation of BNR due to shading correction. All above have precision of u3.3, default to 0.

struct **ipu3_uapi_bnr_static_config_opt_center_config**
Optical center config

Definition

```
struct ipu3_uapi_bnr_static_config_opt_center_config {
  __s32 x_reset:13;
  __u32 reserved0:3;
  __s32 y_reset:13;
  __u32 reserved2:3;
};
```

Members

x_reset
Reset value of X (col start - X center). Precision s12.0.

reserved0
reserved

y_reset
Reset value of Y (row start - Y center). Precision s12.0.

reserved2
reserved

Description

Distance from corner to optical center for NM adaptation due to shading correction (should be calculated based on shading tables)

struct **ipu3_uapi_bnr_static_config_lut_config**
BNR square root lookup table

Definition

```
struct ipu3_uapi_bnr_static_config_lut_config {
  __u8 values[IPU3_UAPI_BNR_LUT_SIZE];
};
```

Members

values
pre-calculated values of square root function.

Description

LUT implementation of square root operation.

struct **ipu3_uapi_bnr_static_config_bp_ctrl_config**
Detect bad pixels (bp)

Definition

```
struct ipu3_uapi_bnr_static_config_bp_ctrl_config {
  __u32 bp_thr_gain:5;
  __u32 reserved0:2;
  __u32 defect_mode:1;
  __u32 bp_gain:6;
  __u32 reserved1:18;
  __u32 w0_coeff:4;
  __u32 reserved2:4;
  __u32 w1_coeff:4;
  __u32 reserved3:20;
};
```

Members

bp_thr_gain
Defines the threshold that specifies how different a defect pixel can be from its neighbors. Threshold is dependent on de-noise threshold calculated by algorithm. Range [4, 31], default 4.

reserved0
reserved

defect_mode
Mode of addressed defect pixels, 0 - single defect pixel is expected, 1 - 2 adjacent defect pixels are expected, default 1.

bp_gain
Defines how 2nd derivation that passes through a defect pixel is different from 2nd derivations that pass through neighbor pixels. u4.2, range [0, 256], default 8.

reserved1
reserved

w0_coeff
Blending coefficient of defect pixel correction. Precision u4, range [0, 8], default 8.

reserved2
reserved

w1_coeff
Enable influence of incorrect defect pixel correction to be avoided. Precision u4, range [1, 8], default 8.

reserved3
reserved

struct **ipu3_uapi_bnr_static_config_dn_detect_ctrl_config**
Denoising config

Definition

```
struct ipu3_uapi_bnr_static_config_dn_detect_ctrl_config {
  __u32 alpha:4;
  __u32 beta:4;
  __u32 gamma:4;
  __u32 reserved0:4;
  __u32 max_inf:4;
  __u32 reserved1:7;
  __u32 gd_enable:1;
  __u32 bpc_enable:1;
  __u32 bnr_enable:1;
  __u32 ff_enable:1;
  __u32 reserved2:1;
};
```

Members

alpha
: Weight of central element of smoothing filter.

beta
: Weight of peripheral elements of smoothing filter, default 4.

gamma
: Weight of diagonal elements of smoothing filter, default 4.

reserved0
: reserved

max_inf
: Maximum increase of peripheral or diagonal element influence relative to the pre-defined value range: [0x5, 0xa]

reserved1
: reserved

gd_enable
: Green disparity enable control, 0 - disable, 1 - enable.

bpc_enable
: Bad pixel correction enable control, 0 - disable, 1 - enable.

bnr_enable
: Bayer noise removal enable control, 0 - disable, 1 - enable.

ff_enable
: Fixed function enable, 0 - disable, 1 - enable.

reserved2
: reserved

Description

beta and gamma parameter define the strength of the noise removal filter.
: All above has precision u0.4, range [0, 0xf] format: u0.4 (no / zero bits represent whole number, 4 bits represent the fractional part with each count representing 0.0625) e.g. 0xf translates to 0.0625x15 = 0.9375

struct **ipu3_uapi_bnr_static_config_opt_center_sqr_config**
BNR optical square

Definition

```
struct ipu3_uapi_bnr_static_config_opt_center_sqr_config {
  __u32 x_sqr_reset;
  __u32 y_sqr_reset;
};
```

Members

x_sqr_reset
Reset value of X^2.

y_sqr_reset
Reset value of Y^2.

Description

Please note:

1. X and Y ref to *ipu3_uapi_bnr_static_config_opt_center_config*
2. Both structs are used in threshold formula to calculate r^2, where r is a radius of pixel [row, col] from centor of sensor.

struct **ipu3_uapi_bnr_static_config**
BNR static config

Definition

```
struct ipu3_uapi_bnr_static_config {
  struct ipu3_uapi_bnr_static_config_wb_gains_config wb_gains;
  struct ipu3_uapi_bnr_static_config_wb_gains_thr_config wb_gains_
→thr;
  struct ipu3_uapi_bnr_static_config_thr_coeffs_config thr_coeffs;
  struct ipu3_uapi_bnr_static_config_thr_ctrl_shd_config thr_ctrl_
→shd;
  struct ipu3_uapi_bnr_static_config_opt_center_config opt_center;
  struct ipu3_uapi_bnr_static_config_lut_config lut;
  struct ipu3_uapi_bnr_static_config_bp_ctrl_config bp_ctrl;
  struct ipu3_uapi_bnr_static_config_dn_detect_ctrl_config dn_
→detect_ctrl;
  __u32 column_size;
  struct ipu3_uapi_bnr_static_config_opt_center_sqr_config opt_
→center_sqr;
};
```

Members

wb_gains
white balance gains *ipu3_uapi_bnr_static_config_wb_gains_config*

wb_gains_thr
white balance gains threshold as defined by *ipu3_uapi_bnr_static_config_wb_gains_thr_config*

thr_coeffs
coefficients of threshold *ipu3_uapi_bnr_static_config_thr_coeffs_config*

thr_ctrl_shd
control of shading threshold *ipu3_uapi_bnr_static_config_thr_ctrl_shd_config*

opt_center
optical center *ipu3_uapi_bnr_static_config_opt_center_config*

lut
lookup table *ipu3_uapi_bnr_static_config_lut_config*

bp_ctrl
detect and remove bad pixels as defined in struct *ipu3_uapi_bnr_static_config_bp_ctrl_config*

dn_detect_ctrl
detect and remove noise. *ipu3_uapi_bnr_static_config_dn_detect_ctrl_config*

column_size
The number of pixels in column.

opt_center_sqr
Reset value of r^2 to optical center, see *ipu3_uapi_bnr_static_config_opt_center_sqr_*

Description

Above parameters and opt_center_sqr are used for white balance and shading.

struct **ipu3_uapi_bnr_static_config_green_disparity**
Correct green disparity

Definition

```
struct ipu3_uapi_bnr_static_config_green_disparity {
  __u32 gd_red:6;
  __u32 reserved0:2;
  __u32 gd_green:6;
  __u32 reserved1:2;
  __u32 gd_blue:6;
  __u32 reserved2:10;
  __u32 gd_black:14;
  __u32 reserved3:2;
  __u32 gd_shading:7;
  __u32 reserved4:1;
  __u32 gd_support:2;
  __u32 reserved5:1;
  __u32 gd_clip:1;
  __u32 gd_central_weight:4;
};
```

Members

gd_red
Shading gain coeff for gr disparity level in bright red region. Precision u0.6, default 4(0.0625).

reserved0
reserved

gd_green
Shading gain coeff for gr disparity level in bright green region. Precision u0.6, default 4(0.0625).

reserved1
reserved

gd_blue
Shading gain coeff for gr disparity level in bright blue region. Precision u0.6, default 4(0.0625).

reserved2
reserved

gd_black
Maximal green disparity level in dark region (stronger disparity assumed to be image detail). Precision u14, default 80.

reserved3
reserved

gd_shading
Change maximal green disparity level according to square distance from image center.

reserved4
reserved

gd_support
Lower bound for the number of second green color pixels in current pixel neighborhood with less than threshold difference from it.

reserved5
reserved

gd_clip
Turn green disparity clip on/off, [0, 1], default 1.

gd_central_weight
Central pixel weight in 9 pixels weighted sum.

Description

The shading gain coeff of red, green, blue and black are used to calculate threshold given a pixel's color value and its coordinates in the image.

struct **ipu3_uapi_dm_config**

De-mosaic parameters

Definition

```
struct ipu3_uapi_dm_config {
  __u32 dm_en:1;
  __u32 ch_ar_en:1;
  __u32 fcc_en:1;
```

(continues on next page)

(continued from previous page)

```
  __u32 reserved0:13;
  __u32 frame_width:16;
  __u32 gamma_sc:5;
  __u32 reserved1:3;
  __u32 lc_ctrl:5;
  __u32 reserved2:3;
  __u32 cr_param1:5;
  __u32 reserved3:3;
  __u32 cr_param2:5;
  __u32 reserved4:3;
  __u32 coring_param:5;
  __u32 reserved5:27;
};
```

Members

dm_en
: de-mosaic enable.

ch_ar_en
: Checker artifacts removal enable flag. Default 0.

fcc_en
: False color correction (FCC) enable flag. Default 0.

reserved0
: reserved

frame_width
: do not care

gamma_sc
: Sharpening coefficient (coefficient of 2-d derivation of complementary color in Hamilton-Adams interpolation). u5, range [0, 31], default 8.

reserved1
: reserved

lc_ctrl
: Parameter that controls weights of Chroma Homogeneity metric in calculation of final homogeneity metric. u5, range [0, 31], default 7.

reserved2
: reserved

cr_param1
: First parameter that defines Checker artifact removal feature gain. Precision u5, range [0, 31], default 8.

reserved3
: reserved

cr_param2
: Second parameter that defines Checker artifact removal feature gain. Precision u5, range [0, 31], default 8.

reserved4
reserved

coring_param
Defines power of false color correction operation. low for preserving edge colors, high for preserving gray edge artifacts. Precision u1.4, range [0, 1.9375], default 4 (0.25).

reserved5
reserved

Description

The demosaic fixed function block is responsible to covert Bayer(mosaiced) images into color images based on demosaicing algorithm.

struct **ipu3_uapi_ccm_mat_config**
Color correction matrix

Definition

```
struct ipu3_uapi_ccm_mat_config {
  __s16 coeff_m11;
  __s16 coeff_m12;
  __s16 coeff_m13;
  __s16 coeff_o_r;
  __s16 coeff_m21;
  __s16 coeff_m22;
  __s16 coeff_m23;
  __s16 coeff_o_g;
  __s16 coeff_m31;
  __s16 coeff_m32;
  __s16 coeff_m33;
  __s16 coeff_o_b;
};
```

Members

coeff_m11
CCM 3x3 coefficient, range [-65536, 65535]

coeff_m12
CCM 3x3 coefficient, range [-8192, 8191]

coeff_m13
CCM 3x3 coefficient, range [-32768, 32767]

coeff_o_r
Bias 3x1 coefficient, range [-8191, 8181]

coeff_m21
CCM 3x3 coefficient, range [-32767, 32767]

coeff_m22
CCM 3x3 coefficient, range [-8192, 8191]

coeff_m23
CCM 3x3 coefficient, range [-32768, 32767]

coeff_o_g
Bias 3x1 coefficient, range [-8191, 8181]

coeff_m31
CCM 3x3 coefficient, range [-32768, 32767]

coeff_m32
CCM 3x3 coefficient, range [-8192, 8191]

coeff_m33
CCM 3x3 coefficient, range [-32768, 32767]

coeff_o_b
Bias 3x1 coefficient, range [-8191, 8181]

Description

Transform sensor specific color space to standard sRGB by applying 3x3 matrix and adding a bias vector O. The transformation is basically a rotation and translation in the 3-dimensional color spaces. Here are the defaults:

9775, -2671, 1087, 0 -1071, 8303, 815, 0 -23, -7887, 16103, 0

struct **ipu3_uapi_gamma_corr_ctrl**
Gamma correction

Definition

```
struct ipu3_uapi_gamma_corr_ctrl {
  __u32 enable:1;
  __u32 reserved:31;
};
```

Members

enable
gamma correction enable.

reserved
reserved

struct **ipu3_uapi_gamma_corr_lut**
Per-pixel tone mapping implemented as LUT.

Definition

```
struct ipu3_uapi_gamma_corr_lut {
  __u16 lut[IPU3_UAPI_GAMMA_CORR_LUT_ENTRIES];
};
```

Members

lut
256 tabulated values of the gamma function. LUT[1].. LUT[256] format u13.0, range [0, 8191].

Description

The tone mapping operation is done by a Piece wise linear graph that is implemented as a lookup table(LUT). The pixel component input intensity is the X-axis of the graph which is the table entry.

struct **ipu3_uapi_gamma_config**

Gamma config

Definition

```
struct ipu3_uapi_gamma_config {
  struct ipu3_uapi_gamma_corr_ctrl gc_ctrl ;
  struct ipu3_uapi_gamma_corr_lut gc_lut ;
};
```

Members

gc_ctrl
control of gamma correction *ipu3_uapi_gamma_corr_ctrl*

gc_lut
lookup table of gamma correction *ipu3_uapi_gamma_corr_lut*

struct **ipu3_uapi_csc_mat_config**

Color space conversion matrix config

Definition

```
struct ipu3_uapi_csc_mat_config {
  __s16 coeff_c11;
  __s16 coeff_c12;
  __s16 coeff_c13;
  __s16 coeff_b1;
  __s16 coeff_c21;
  __s16 coeff_c22;
  __s16 coeff_c23;
  __s16 coeff_b2;
  __s16 coeff_c31;
  __s16 coeff_c32;
  __s16 coeff_c33;
  __s16 coeff_b3;
};
```

Members

coeff_c11
Conversion matrix value, format s0.14, range [-16384, 16383].

coeff_c12
Conversion matrix value, format s0.14, range [-8192, 8191].

coeff_c13
Conversion matrix value, format s0.14, range [-16384, 16383].

coeff_b1
Bias 3x1 coefficient, s13.0 range [-8192, 8191].

coeff_c21
Conversion matrix value, format s0.14, range [-16384, 16383].

coeff_c22
Conversion matrix value, format s0.14, range [-8192, 8191].

coeff_c23
Conversion matrix value, format s0.14, range [-16384, 16383].

coeff_b2
Bias 3x1 coefficient, s13.0 range [-8192, 8191].

coeff_c31
Conversion matrix value, format s0.14, range [-16384, 16383].

coeff_c32
Conversion matrix value, format s0.14, range [-8192, 8191].

coeff_c33
Conversion matrix value, format s0.14, range [-16384, 16383].

coeff_b3
Bias 3x1 coefficient, s13.0 range [-8192, 8191].

Description

To transform each pixel from RGB to YUV (Y - brightness/luminance, UV -chroma) by applying the pixel's values by a 3x3 matrix and adding an optional bias 3x1 vector. Here are the default values for the matrix:

4898, 9617, 1867, 0, -2410, -4732, 7143, 0, 10076, -8437, -1638, 0,

(i.e. for real number 0.299, 0.299 * 2^14 becomes 4898.)

struct **ipu3_uapi_cds_params**
Chroma down-scaling

Definition

```
struct ipu3_uapi_cds_params {
  __u32 ds_c00:2;
  __u32 ds_c01:2;
  __u32 ds_c02:2;
  __u32 ds_c03:2;
  __u32 ds_c10:2;
  __u32 ds_c11:2;
  __u32 ds_c12:2;
  __u32 ds_c13:2;
  __u32 ds_nf:5;
  __u32 reserved0:3;
  __u32 csc_en:1;
  __u32 uv_bin_output:1;
  __u32 reserved1:6;
};
```

Members

ds_c00
range [0, 3]

ds_c01
range [0, 3]

ds_c02
range [0, 3]

ds_c03
range [0, 3]

ds_c10
range [0, 3]

ds_c11
range [0, 3]

ds_c12
range [0, 3]

ds_c13
range [0, 3]

ds_nf
Normalization factor for Chroma output downscaling filter, range 0,4, default 2.

reserved0
reserved

csc_en
Color space conversion enable

uv_bin_output
0: output YUV 4.2.0, 1: output YUV 4.2.2(default).

reserved1
reserved

Description

In case user does not provide, above 4x2 filter will use following defaults:
1, 3, 3, 1, 1, 3, 3, 1,

struct **ipu3_uapi_shd_grid_config**

Bayer shading(darkening) correction

Definition

```
struct ipu3_uapi_shd_grid_config {
  __u8 width;
  __u8 height;
  __u8 block_width_log2:3;
  __u8 reserved0:1;
  __u8 block_height_log2:3;
  __u8 reserved1:1;
  __u8 grid_height_per_slice;
```

(continues on next page)

(continued from previous page)

```
  __s16 x_start;
  __s16 y_start;
};
```

Members

width
Grid horizontal dimensions, u8, [8, 128], default 73

height
Grid vertical dimensions, u8, [8, 128], default 56

block_width_log2
Log2 of the width of the grid cell in pixel count u4, [0, 15], default value 5.

reserved0
reserved

block_height_log2
Log2 of the height of the grid cell in pixel count u4, [0, 15], default value 6.

reserved1
reserved

grid_height_per_slice
SHD_MAX_CELLS_PER_SET/width. (with SHD_MAX_CELLS_PER_SET = 146).

x_start
X value of top left corner of sensor relative to ROI s13, [-4096, 0], default 0, only negative values.

y_start
Y value of top left corner of sensor relative to ROI s13, [-4096, 0], default 0, only negative values.

struct **ipu3_uapi_shd_general_config**
Shading general config

Definition

```
struct ipu3_uapi_shd_general_config {
  __u32 init_set_vrt_offst_ul:8;
  __u32 shd_enable:1;
  __u32 gain_factor:2;
  __u32 reserved:21;
};
```

Members

init_set_vrt_offst_ul
set vertical offset, y_start >> block_height_log2 % grid_height_per_slice.

shd_enable
shading enable.

gain_factor
Gain factor. Shift calculated anti shading value. Precision u2. 0x0 - gain factor [1, 5], means no shift interpolated value. 0x1 - gain factor [1, 9], means shift interpolated by 1. 0x2 - gain factor [1, 17], means shift interpolated by 2.

reserved
reserved

Description

Correction is performed by multiplying a gain factor for each of the 4 Bayer channels as a function of the pixel location in the sensor.

struct **ipu3_uapi_shd_black_level_config**
Black level correction

Definition

```
struct ipu3_uapi_shd_black_level_config {
  __s16 bl_r;
  __s16 bl_gr;
  __s16 bl_gb;
  __s16 bl_b;
};
```

Members

bl_r
Bios values for green red. s11 range [-2048, 2047].

bl_gr
Bios values for green blue. s11 range [-2048, 2047].

bl_gb
Bios values for red. s11 range [-2048, 2047].

bl_b
Bios values for blue. s11 range [-2048, 2047].

struct **ipu3_uapi_shd_config_static**
Shading config static

Definition

```
struct ipu3_uapi_shd_config_static {
  struct ipu3_uapi_shd_grid_config grid;
  struct ipu3_uapi_shd_general_config general;
  struct ipu3_uapi_shd_black_level_config black_level;
};
```

Members

grid
shading grid config *ipu3_uapi_shd_grid_config*

general
shading general config *ipu3_uapi_shd_general_config*

black_level
black level config for shading correction as defined by *ipu3_uapi_shd_black_level_config*

struct **ipu3_uapi_shd_lut**
Shading gain factor lookup table.

Definition

```
struct ipu3_uapi_shd_lut {
  struct {
    struct {
      __u16 r;
      __u16 gr;
    } r_and_gr[IPU3_UAPI_SHD_MAX_CELLS_PER_SET];
    __u8 reserved1[24];
    struct {
      __u16 gb;
      __u16 b;
    } gb_and_b[IPU3_UAPI_SHD_MAX_CELLS_PER_SET];
    __u8 reserved2[24];
  } sets[IPU3_UAPI_SHD_MAX_CFG_SETS];
};
```

Members

sets
array

sets.r_and_gr
Red and GreenR Lookup table.

sets.r_and_gr.r
Red shading factor.

sets.r_and_gr.gr
GreenR shading factor.

sets.reserved1
reserved

sets.gb_and_b
GreenB and Blue Lookup table.

sets.gb_and_b.gb
GreenB shading factor.

sets.gb_and_b.b
Blue shading factor.

sets.reserved2
reserved

Description

Map to shading correction LUT register set.

struct **ipu3_uapi_shd_config**
Shading config

Definition

```
struct ipu3_uapi_shd_config {
  struct ipu3_uapi_shd_config_static shd ;
  struct ipu3_uapi_shd_lut shd_lut ;
};
```

Members

shd
shading static config, see *ipu3_uapi_shd_config_static*

shd_lut
shading lookup table *ipu3_uapi_shd_lut*

struct **ipu3_uapi_iefd_cux2**
IEFd Config Unit 2 parameters

Definition

```
struct ipu3_uapi_iefd_cux2 {
  __u32 x0:9;
  __u32 x1:9;
  __u32 a01:9;
  __u32 b01:5;
};
```

Members

x0
X0 point of Config Unit, u9.0, default 0.

x1
X1 point of Config Unit, u9.0, default 0.

a01
Slope A of Config Unit, s4.4, default 0.

b01
Slope B, always 0.

Description

Calculate weight for blending directed and non-directed denoise elements

All CU inputs are unsigned, they will be converted to signed when written to register, i.e. a01 will be written to 9 bit register in s4.4 format. The data precision s4.4 means 4 bits for integer parts and 4 bits for the fractional part, the first bit indicates positive or negative value. For userspace software (commonly the imaging library), the computation for the CU slope values should be based on the slope resolution 1/16 (binary 0.0001 - the minimal interval value), the slope value range is [-256, +255]. This applies to *ipu3_uapi_iefd_cux6_ed*, *ipu3_uapi_iefd_cux2_1*, *ipu3_uapi_iefd_cux2_1*, *ipu3_uapi_iefd_cux4* and *ipu3_uapi_iefd_cux6_rad*.

Note

Each instance of Config Unit needs X coordinate of n points and slope A factor between points calculated by driver based on calibration parameters.

struct **ipu3_uapi_iefd_cux6_ed**

Calculate power of non-directed sharpening element, Config Unit 6 for edge detail (ED).

Definition

```
struct ipu3_uapi_iefd_cux6_ed {
  __u32 x0:9;
  __u32 x1:9;
  __u32 x2:9;
  __u32 reserved0:5;
  __u32 x3:9;
  __u32 x4:9;
  __u32 x5:9;
  __u32 reserved1:5;
  __u32 a01:9;
  __u32 a12:9;
  __u32 a23:9;
  __u32 reserved2:5;
  __u32 a34:9;
  __u32 a45:9;
  __u32 reserved3:14;
  __u32 b01:9;
  __u32 b12:9;
  __u32 b23:9;
  __u32 reserved4:5;
  __u32 b34:9;
  __u32 b45:9;
  __u32 reserved5:14;
};
```

Members

x0
X coordinate of point 0, u9.0, default 0.

x1
X coordinate of point 1, u9.0, default 0.

x2
X coordinate of point 2, u9.0, default 0.

reserved0
reserved

x3
X coordinate of point 3, u9.0, default 0.

x4
X coordinate of point 4, u9.0, default 0.

x5
X coordinate of point 5, u9.0, default 0.

reserved1
reserved

a01
slope A points 01, s4.4, default 0.

a12
slope A points 12, s4.4, default 0.

a23
slope A points 23, s4.4, default 0.

reserved2
reserved

a34
slope A points 34, s4.4, default 0.

a45
slope A points 45, s4.4, default 0.

reserved3
reserved

b01
slope B points 01, s4.4, default 0.

b12
slope B points 12, s4.4, default 0.

b23
slope B points 23, s4.4, default 0.

reserved4
reserved

b34
slope B points 34, s4.4, default 0.

b45
slope B points 45, s4.4, default 0.

reserved5
reserved.

struct **ipu3_uapi_iefd_cux2_1**

Calculate power of non-directed denoise element apply.

Definition

```
struct ipu3_uapi_iefd_cux2_1 {
  __u32 x0:9;
  __u32 x1:9;
  __u32 a01:9;
  __u32 reserved1:5;
  __u32 b01:8;
  __u32 reserved2:24;
};
```

Members

x0
X0 point of Config Unit, u9.0, default 0.

x1
X1 point of Config Unit, u9.0, default 0.

a01
Slope A of Config Unit, s4.4, default 0.

reserved1
reserved

b01
offset B0 of Config Unit, u7.0, default 0.

reserved2
reserved

struct **ipu3_uapi_iefd_cux4**
Calculate power of non-directed sharpening element.

Definition

```
struct ipu3_uapi_iefd_cux4 {
  __u32 x0:9;
  __u32 x1:9;
  __u32 x2:9;
  __u32 reserved0:5;
  __u32 x3:9;
  __u32 a01:9;
  __u32 a12:9;
  __u32 reserved1:5;
  __u32 a23:9;
  __u32 b01:8;
  __u32 b12:8;
  __u32 reserved2:7;
  __u32 b23:8;
  __u32 reserved3:24;
};
```

Members

x0
X0 point of Config Unit, u9.0, default 0.

x1
X1 point of Config Unit, u9.0, default 0.

x2
X2 point of Config Unit, u9.0, default 0.

reserved0
reserved

x3
X3 point of Config Unit, u9.0, default 0.

a01
: Slope A0 of Config Unit, s4.4, default 0.

a12
: Slope A1 of Config Unit, s4.4, default 0.

reserved1
: reserved

a23
: Slope A2 of Config Unit, s4.4, default 0.

b01
: Offset B0 of Config Unit, s7.0, default 0.

b12
: Offset B1 of Config Unit, s7.0, default 0.

reserved2
: reserved

b23
: Offset B2 of Config Unit, s7.0, default 0.

reserved3
: reserved

struct **ipu3_uapi_iefd_cux6_rad**
: Radial Config Unit (CU)

Definition

```
struct ipu3_uapi_iefd_cux6_rad {
  __u32 x0:8;
  __u32 x1:8;
  __u32 x2:8;
  __u32 x3:8;
  __u32 x4:8;
  __u32 x5:8;
  __u32 reserved1:16;
  __u32 a01:16;
  __u32 a12:16;
  __u32 a23:16;
  __u32 a34:16;
  __u32 a45:16;
  __u32 reserved2:16;
  __u32 b01:10;
  __u32 b12:10;
  __u32 b23:10;
  __u32 reserved4:2;
  __u32 b34:10;
  __u32 b45:10;
  __u32 reserved5:12;
};
```

Members

x0
: x0 points of Config Unit radial, u8.0

x1
: x1 points of Config Unit radial, u8.0

x2
: x2 points of Config Unit radial, u8.0

x3
: x3 points of Config Unit radial, u8.0

x4
: x4 points of Config Unit radial, u8.0

x5
: x5 points of Config Unit radial, u8.0

reserved1
: reserved

a01
: Slope A of Config Unit radial, s7.8

a12
: Slope A of Config Unit radial, s7.8

a23
: Slope A of Config Unit radial, s7.8

a34
: Slope A of Config Unit radial, s7.8

a45
: Slope A of Config Unit radial, s7.8

reserved2
: reserved

b01
: Slope B of Config Unit radial, s9.0

b12
: Slope B of Config Unit radial, s9.0

b23
: Slope B of Config Unit radial, s9.0

reserved4
: reserved

b34
: Slope B of Config Unit radial, s9.0

b45
: Slope B of Config Unit radial, s9.0

reserved5
: reserved

struct **ipu3_uapi_yuvp1_iefd_cfg_units**
IEFd Config Units parameters

Definition

```
struct ipu3_uapi_yuvp1_iefd_cfg_units {
  struct ipu3_uapi_iefd_cux2 cu_1;
  struct ipu3_uapi_iefd_cux6_ed cu_ed;
  struct ipu3_uapi_iefd_cux2 cu_3;
  struct ipu3_uapi_iefd_cux2_1 cu_5;
  struct ipu3_uapi_iefd_cux4 cu_6;
  struct ipu3_uapi_iefd_cux2 cu_7;
  struct ipu3_uapi_iefd_cux4 cu_unsharp;
  struct ipu3_uapi_iefd_cux6_rad cu_radial;
  struct ipu3_uapi_iefd_cux2 cu_vssnlm;
};
```

Members

cu_1
calculate weight for blending directed and non-directed denoise elements. See *ipu3_uapi_iefd_cux2*

cu_ed
calculate power of non-directed sharpening element, see *ipu3_uapi_iefd_cux6_ed*

cu_3
calculate weight for blending directed and non-directed denoise elements. A *ipu3_uapi_iefd_cux2*

cu_5
calculate power of non-directed denoise element apply, use *ipu3_uapi_iefd_cux2_1*

cu_6
calculate power of non-directed sharpening element. See *ipu3_uapi_iefd_cux4*

cu_7
calculate weight for blending directed and non-directed denoise elements. Use *ipu3_uapi_iefd_cux2*

cu_unsharp
Config Unit of unsharp *ipu3_uapi_iefd_cux4*

cu_radial
Config Unit of radial *ipu3_uapi_iefd_cux6_rad*

cu_vssnlm
Config Unit of vssnlm *ipu3_uapi_iefd_cux2*

struct **ipu3_uapi_yuvp1_iefd_config_s**
IEFd config

Definition

```
struct ipu3_uapi_yuvp1_iefd_config_s {
  __u32 horver_diag_coeff:7;
  __u32 reserved0:1;
  __u32 clamp_stitch:6;
  __u32 reserved1:2;
  __u32 direct_metric_update:5;
  __u32 reserved2:3;
  __u32 ed_horver_diag_coeff:7;
  __u32 reserved3:1;
};
```

Members

horver_diag_coeff
Gradient compensation. Compared with vertical / horizontal (0 / 90 degree), coefficient of diagonal (45 / 135 degree) direction should be corrected by approx. 1/sqrt(2).

reserved0
reserved

clamp_stitch
Slope to stitch between clamped and unclamped edge values

reserved1
reserved

direct_metric_update
Update coeff for direction metric

reserved2
reserved

ed_horver_diag_coeff
Radial Coefficient that compensates for different distance for vertical/horizontal and diagonal gradient calculation (approx. 1/sqrt(2))

reserved3
reserved

struct **ipu3_uapi_yuvp1_iefd_control**
IEFd control

Definition

```
struct ipu3_uapi_yuvp1_iefd_control {
  __u32 iefd_en:1;
  __u32 denoise_en:1;
  __u32 direct_smooth_en:1;
  __u32 rad_en:1;
  __u32 vssnlm_en:1;
  __u32 reserved:27;
};
```

Members

iefd_en
Enable IEFd

denoise_en
Enable denoise

direct_smooth_en
Enable directional smooth

rad_en
Enable radial update

vssnlm_en
Enable VSSNLM output filter

reserved
reserved

struct **ipu3_uapi_sharp_cfg**
Sharpening config

Definition

```
struct ipu3_uapi_sharp_cfg {
  __u32 nega_lmt_txt:13;
  __u32 reserved0:19;
  __u32 posi_lmt_txt:13;
  __u32 reserved1:19;
  __u32 nega_lmt_dir:13;
  __u32 reserved2:19;
  __u32 posi_lmt_dir:13;
  __u32 reserved3:19;
};
```

Members

nega_lmt_txt
Sharpening limit for negative overshoots for texture.

reserved0
reserved

posi_lmt_txt
Sharpening limit for positive overshoots for texture.

reserved1
reserved

nega_lmt_dir
Sharpening limit for negative overshoots for direction (edge).

reserved2
reserved

posi_lmt_dir
Sharpening limit for positive overshoots for direction (edge).

reserved3
reserved

Description

Fixed point type u13.0, range [0, 8191].

struct **ipu3_uapi_far_w**

Sharpening config for far sub-group

Definition

```
struct ipu3_uapi_far_w {
  __u32 dir_shrp:7;
  __u32 reserved0:1;
  __u32 dir_dns:7;
  __u32 reserved1:1;
  __u32 ndir_dns_powr:7;
  __u32 reserved2:9;
};
```

Members

dir_shrp
Weight of wide direct sharpening, u1.6, range [0, 64], default 64.

reserved0
reserved

dir_dns
Weight of wide direct denoising, u1.6, range [0, 64], default 0.

reserved1
reserved

ndir_dns_powr
Power of non-direct denoising, Precision u1.6, range [0, 64], default 64.

reserved2
reserved

struct **ipu3_uapi_unsharp_cfg**

Unsharp config

Definition

```
struct ipu3_uapi_unsharp_cfg {
  __u32 unsharp_weight:7;
  __u32 reserved0:1;
  __u32 unsharp_amount:9;
  __u32 reserved1:15;
};
```

Members

unsharp_weight
Unsharp mask blending weight. u1.6, range [0, 64], default 16. 0 - disabled, 64 - use only unsharp.

reserved0
reserved

unsharp_amount
Unsharp mask amount, u4.5, range [0, 511], default 0.

reserved1
reserved

struct **ipu3_uapi_yuvp1_iefd_shrp_cfg**
IEFd sharpness config

Definition

```
struct ipu3_uapi_yuvp1_iefd_shrp_cfg {
  struct ipu3_uapi_sharp_cfg cfg;
  struct ipu3_uapi_far_w far_w;
  struct ipu3_uapi_unsharp_cfg unshrp_cfg;
};
```

Members

cfg
sharpness config *ipu3_uapi_sharp_cfg*

far_w
wide range config, value as specified by *ipu3_uapi_far_w*: The 5x5 environment is separated into 2 sub-groups, the 3x3 nearest neighbors (8 pixels called Near), and the second order neighborhood around them (16 pixels called Far).

unshrp_cfg
unsharpness config. *ipu3_uapi_unsharp_cfg*

struct **ipu3_uapi_unsharp_coef0**
Unsharp mask coefficients

Definition

```
struct ipu3_uapi_unsharp_coef0 {
  __u32 c00:9;
  __u32 c01:9;
  __u32 c02:9;
  __u32 reserved:5;
};
```

Members

c00
Coeff11, s0.8, range [-255, 255], default 1.

c01
Coeff12, s0.8, range [-255, 255], default 5.

c02
Coeff13, s0.8, range [-255, 255], default 9.

reserved
reserved

Description

Configurable registers for common sharpening support.

struct **ipu3_uapi_unsharp_coef1**

Unsharp mask coefficients

Definition

```
struct ipu3_uapi_unsharp_coef1 {
  __u32 c11:9;
  __u32 c12:9;
  __u32 c22:9;
  __u32 reserved:5;
};
```

Members

c11
Coeff22, s0.8, range [-255, 255], default 29.

c12
Coeff23, s0.8, range [-255, 255], default 55.

c22
Coeff33, s0.8, range [-255, 255], default 96.

reserved
reserved

struct **ipu3_uapi_yuvp1_iefd_unshrp_cfg**

Unsharp mask config

Definition

```
struct ipu3_uapi_yuvp1_iefd_unshrp_cfg {
  struct ipu3_uapi_unsharp_coef0 unsharp_coef0;
  struct ipu3_uapi_unsharp_coef1 unsharp_coef1;
};
```

Members

unsharp_coef0
unsharp coefficient 0 config. See *ipu3_uapi_unsharp_coef0*

unsharp_coef1
unsharp coefficient 1 config. See *ipu3_uapi_unsharp_coef1*

struct **ipu3_uapi_radial_reset_xy**

Radial coordinate reset

Definition

```
struct ipu3_uapi_radial_reset_xy {
  __s32 x:13;
  __u32 reserved0:3;
  __s32 y:13;
```

(continues on next page)

(continued from previous page)

```
  __u32 reserved1:3;
};
```

Members

x
: Radial reset of x coordinate. Precision s12, [-4095, 4095], default 0.

reserved0
: reserved

y
: Radial center y coordinate. Precision s12, [-4095, 4095], default 0.

reserved1
: reserved

struct **ipu3_uapi_radial_reset_x2**
: Radial X^2 reset

Definition

```
struct ipu3_uapi_radial_reset_x2 {
  __u32 x2:24;
  __u32 reserved:8;
};
```

Members

x2
: Radial reset of x^2 coordinate. Precision u24, default 0.

reserved
: reserved

struct **ipu3_uapi_radial_reset_y2**
: Radial Y^2 reset

Definition

```
struct ipu3_uapi_radial_reset_y2 {
  __u32 y2:24;
  __u32 reserved:8;
};
```

Members

y2
: Radial reset of y^2 coordinate. Precision u24, default 0.

reserved
: reserved

struct **ipu3_uapi_radial_cfg**
: Radial config

Definition

```
struct ipu3_uapi_radial_cfg {
  __u32 rad_nf:4;
  __u32 reserved0:4;
  __u32 rad_inv_r2:7;
  __u32 reserved1:17;
};
```

Members

rad_nf
    Radial. R^2 normalization factor is scale down by 2^ - (15 + scale)

reserved0
    reserved

rad_inv_r2
    Radial R^-2 normelized to (0.5..1). Precision u7, range [0, 127].

reserved1
    reserved

struct **ipu3_uapi_rad_far_w**

    Radial FAR sub-group

Definition

```
struct ipu3_uapi_rad_far_w {
  __u32 rad_dir_far_sharp_w:8;
  __u32 rad_dir_far_dns_w:8;
  __u32 rad_ndir_far_dns_power:8;
  __u32 reserved:8;
};
```

Members

rad_dir_far_sharp_w
    Weight of wide direct sharpening, u1.6, range [0, 64], default 64.

rad_dir_far_dns_w
    Weight of wide direct denoising, u1.6, range [0, 64], default 0.

rad_ndir_far_dns_power
    power of non-direct sharpening, u1.6, range [0, 64], default 0.

reserved
    reserved

struct **ipu3_uapi_cu_cfg0**

    Radius Config Unit cfg0 register

Definition

```
struct ipu3_uapi_cu_cfg0 {
  __u32 cu6_pow:7;
  __u32 reserved0:1;
  __u32 cu_unsharp_pow:7;
```

(continues on next page)

(continued from previous page)

```
  __u32 reserved1:1;
  __u32 rad_cu6_pow:7;
  __u32 reserved2:1;
  __u32 rad_cu_unsharp_pow:6;
  __u32 reserved3:2;
};
```

Members

cu6_pow
: Power of CU6. Power of non-direct sharpening, u3.4.

reserved0
: reserved

cu_unsharp_pow
: Power of unsharp mask, u2.4.

reserved1
: reserved

rad_cu6_pow
: Radial/corner CU6. Directed sharpening power, u3.4.

reserved2
: reserved

rad_cu_unsharp_pow
: Radial power of unsharp mask, u2.4.

reserved3
: reserved

struct **ipu3_uapi_cu_cfg1**

Radius Config Unit cfg1 register

Definition

```
struct ipu3_uapi_cu_cfg1 {
  __u32 rad_cu6_x1:9;
  __u32 reserved0:1;
  __u32 rad_cu_unsharp_x1:9;
  __u32 reserved1:13;
};
```

Members

rad_cu6_x1
: X1 point of Config Unit 6, precision u9.0.

reserved0
: reserved

rad_cu_unsharp_x1
: X1 point for Config Unit unsharp for radial/corner point precision u9.0.

reserved1
reserved

struct **ipu3_uapi_yuvp1_iefd_rad_cfg**
IEFd parameters changed radially over the picture plane.

Definition

```
struct ipu3_uapi_yuvp1_iefd_rad_cfg {
  struct ipu3_uapi_radial_reset_xy reset_xy;
  struct ipu3_uapi_radial_reset_x2 reset_x2;
  struct ipu3_uapi_radial_reset_y2 reset_y2;
  struct ipu3_uapi_radial_cfg cfg;
  struct ipu3_uapi_rad_far_w rad_far_w;
  struct ipu3_uapi_cu_cfg0 cu_cfg0;
  struct ipu3_uapi_cu_cfg1 cu_cfg1;
};
```

Members

reset_xy
reset xy value in radial calculation. *ipu3_uapi_radial_reset_xy*

reset_x2
reset x square value in radial calculation. See struct *ipu3_uapi_radial_reset_x2*

reset_y2
reset y square value in radial calculation. See struct *ipu3_uapi_radial_reset_y2*

cfg
radial config defined in *ipu3_uapi_radial_cfg*

rad_far_w
weight for wide range radial. *ipu3_uapi_rad_far_w*

cu_cfg0
configuration unit 0. See *ipu3_uapi_cu_cfg0*

cu_cfg1
configuration unit 1. See *ipu3_uapi_cu_cfg1*

struct **ipu3_uapi_vss_lut_x**
Vssnlm LUT x0/x1/x2

Definition

```
struct ipu3_uapi_vss_lut_x {
  __u32 vs_x0:8;
  __u32 vs_x1:8;
  __u32 vs_x2:8;
  __u32 reserved2:8;
};
```

Members

vs_x0
Vssnlm LUT x0, precision u8, range [0, 255], default 16.

vs_x1
Vssnlm LUT x1, precision u8, range [0, 255], default 32.

vs_x2
Vssnlm LUT x2, precision u8, range [0, 255], default 64.

reserved2
reserved

struct **ipu3_uapi_vss_lut_y**
Vssnlm LUT y0/y1/y2

Definition

```
struct ipu3_uapi_vss_lut_y {
  __u32 vs_y1:4;
  __u32 reserved0:4;
  __u32 vs_y2:4;
  __u32 reserved1:4;
  __u32 vs_y3:4;
  __u32 reserved2:12;
};
```

Members

vs_y1
Vssnlm LUT y1, precision u4, range [0, 8], default 1.

reserved0
reserved

vs_y2
Vssnlm LUT y2, precision u4, range [0, 8], default 3.

reserved1
reserved

vs_y3
Vssnlm LUT y3, precision u4, range [0, 8], default 8.

reserved2
reserved

struct **ipu3_uapi_yuvp1_iefd_vssnlm_cfg**
IEFd Vssnlm Lookup table

Definition

```
struct ipu3_uapi_yuvp1_iefd_vssnlm_cfg {
  struct ipu3_uapi_vss_lut_x vss_lut_x;
  struct ipu3_uapi_vss_lut_y vss_lut_y;
};
```

Members

vss_lut_x
vss lookup table. See *ipu3_uapi_vss_lut_x* description

vss_lut_y
vss lookup table. See *ipu3_uapi_vss_lut_y* description

struct **ipu3_uapi_yuvp1_iefd_config**
IEFd config

Definition

```
struct ipu3_uapi_yuvp1_iefd_config {
  struct ipu3_uapi_yuvp1_iefd_cfg_units units;
  struct ipu3_uapi_yuvp1_iefd_config_s config;
  struct ipu3_uapi_yuvp1_iefd_control control;
  struct ipu3_uapi_yuvp1_iefd_shrp_cfg sharp;
  struct ipu3_uapi_yuvp1_iefd_unshrp_cfg unsharp;
  struct ipu3_uapi_yuvp1_iefd_rad_cfg rad;
  struct ipu3_uapi_yuvp1_iefd_vssnlm_cfg vsslnm;
};
```

Members

units
configuration unit setting, *ipu3_uapi_yuvp1_iefd_cfg_units*

config
configuration, as defined by *ipu3_uapi_yuvp1_iefd_config_s*

control
control setting, as defined by *ipu3_uapi_yuvp1_iefd_control*

sharp
sharpness setting, as defined by *ipu3_uapi_yuvp1_iefd_shrp_cfg*

unsharp
unsharpness setting, as defined by *ipu3_uapi_yuvp1_iefd_unshrp_cfg*

rad
radial setting, as defined by *ipu3_uapi_yuvp1_iefd_rad_cfg*

vsslnm
vsslnm setting, as defined by *ipu3_uapi_yuvp1_iefd_vssnlm_cfg*

struct **ipu3_uapi_yuvp1_yds_config**
Y Down-Sampling config

Definition

```
struct ipu3_uapi_yuvp1_yds_config {
  __u32 c00:2;
  __u32 c01:2;
  __u32 c02:2;
  __u32 c03:2;
  __u32 c10:2;
  __u32 c11:2;
```

(continues on next page)

(continued from previous page)

```
    __u32 c12:2;
    __u32 c13:2;
    __u32 norm_factor:5;
    __u32 reserved0:4;
    __u32 bin_output:1;
    __u32 reserved1:6;
};
```

Members

c00
range [0, 3], default 0x0

c01
range [0, 3], default 0x1

c02
range [0, 3], default 0x1

c03
range [0, 3], default 0x0

c10
range [0, 3], default 0x0

c11
range [0, 3], default 0x1

c12
range [0, 3], default 0x1

c13
range [0, 3], default 0x0

norm_factor
Normalization factor, range [0, 4], default 2 0 - divide by 1 1 - divide by 2 2 - divide by 4 3 - divide by 8 4 - divide by 16

reserved0
reserved

bin_output
Down sampling on Luma channel in two optional modes 0 - Bin output 4.2.0 (default), 1 output 4.2.2.

reserved1
reserved

Description

Above are 4x2 filter coefficients for chroma output downscaling.

struct **ipu3_uapi_yuvp1_chnr_enable_config**
Chroma noise reduction enable

Definition

```
struct ipu3_uapi_yuvp1_chnr_enable_config {
  __u32 enable:1;
  __u32 yuv_mode:1;
  __u32 reserved0:14;
  __u32 col_size:12;
  __u32 reserved1:4;
};
```

Members

enable
enable/disable chroma noise reduction

yuv_mode
0 - YUV420, 1 - YUV422

reserved0
reserved

col_size
number of columns in the frame, max width is 2560

reserved1
reserved

struct **ipu3_uapi_yuvp1_chnr_coring_config**
Coring thresholds for UV

Definition

```
struct ipu3_uapi_yuvp1_chnr_coring_config {
  __u32 u:13;
  __u32 reserved0:3;
  __u32 v:13;
  __u32 reserved1:3;
};
```

Members

u
U coring level, u0.13, range [0.0, 1.0], default 0.0

reserved0
reserved

v
V coring level, u0.13, range [0.0, 1.0], default 0.0

reserved1
reserved

struct **ipu3_uapi_yuvp1_chnr_sense_gain_config**
Chroma noise reduction gains

Definition

```
struct ipu3_uapi_yuvp1_chnr_sense_gain_config {
  __u32 vy:8;
  __u32 vu:8;
  __u32 vv:8;
  __u32 reserved0:8;
  __u32 hy:8;
  __u32 hu:8;
  __u32 hv:8;
  __u32 reserved1:8;
};
```

Members

vy
: Sensitivity of horizontal edge of Y, default 100

vu
: Sensitivity of horizontal edge of U, default 100

vv
: Sensitivity of horizontal edge of V, default 100

reserved0
: reserved

hy
: Sensitivity of vertical edge of Y, default 50

hu
: Sensitivity of vertical edge of U, default 50

hv
: Sensitivity of vertical edge of V, default 50

reserved1
: reserved

Description

All sensitivity gain parameters have precision u13.0, range [0, 8191].

struct **ipu3_uapi_yuvp1_chnr_iir_fir_config**
: Chroma IIR/FIR filter config

Definition

```
struct ipu3_uapi_yuvp1_chnr_iir_fir_config {
  __u32 fir_0h:6;
  __u32 reserved0:2;
  __u32 fir_1h:6;
  __u32 reserved1:2;
  __u32 fir_2h:6;
  __u32 dalpha_clip_val:9;
  __u32 reserved2:1;
};
```

Members

fir_0h
Value of center tap in horizontal FIR, range [0, 32], default 8.

reserved0
reserved

fir_1h
Value of distance 1 in horizontal FIR, range [0, 32], default 12.

reserved1
reserved

fir_2h
Value of distance 2 tap in horizontal FIR, range [0, 32], default 0.

dalpha_clip_val
weight for previous row in IIR, range [1, 256], default 0.

reserved2
reserved

struct **ipu3_uapi_yuvp1_chnr_config**
Chroma noise reduction config

Definition

```
struct ipu3_uapi_yuvp1_chnr_config {
  struct ipu3_uapi_yuvp1_chnr_enable_config enable;
  struct ipu3_uapi_yuvp1_chnr_coring_config coring;
  struct ipu3_uapi_yuvp1_chnr_sense_gain_config sense_gain;
  struct ipu3_uapi_yuvp1_chnr_iir_fir_config iir_fir;
};
```

Members

enable
chroma noise reduction enable, see *ipu3_uapi_yuvp1_chnr_enable_config*

coring
coring config for chroma noise reduction, see *ipu3_uapi_yuvp1_chnr_coring_config*

sense_gain
sensitivity config for chroma noise reduction, see ipu3_uapi_yuvp1_chnr_sense_gain_config

iir_fir
iir and fir config for chroma noise reduction, see ipu3_uapi_yuvp1_chnr_iir_fir_config

struct **ipu3_uapi_yuvp1_y_ee_nr_lpf_config**
Luma(Y) edge enhancement low-pass filter coefficients

Definition

```
struct ipu3_uapi_yuvp1_y_ee_nr_lpf_config {
  __u32 a_diag:5;
```

(continues on next page)

(continued from previous page)

```
    __u32 reserved0:3;
    __u32 a_periph:5;
    __u32 reserved1:3;
    __u32 a_cent:5;
    __u32 reserved2:9;
    __u32 enable:1;
};
```

Members

a_diag
Smoothing diagonal coefficient, u5.0.

reserved0
reserved

a_periph
Image smoothing perpherial, u5.0.

reserved1
reserved

a_cent
Image Smoothing center coefficient, u5.0.

reserved2
reserved

enable
0: Y_EE_NR disabled, output = input; 1: Y_EE_NR enabled.

struct **ipu3_uapi_yuvp1_y_ee_nr_sense_config**

Luma(Y) edge enhancement noise reduction sensitivity gains

Definition

```
struct ipu3_uapi_yuvp1_y_ee_nr_sense_config {
    __u32 edge_sense_0:13;
    __u32 reserved0:3;
    __u32 delta_edge_sense:13;
    __u32 reserved1:3;
    __u32 corner_sense_0:13;
    __u32 reserved2:3;
    __u32 delta_corner_sense:13;
    __u32 reserved3:3;
};
```

Members

edge_sense_0
Sensitivity of edge in dark area. u13.0, default 8191.

reserved0
reserved

delta_edge_sense
Difference in the sensitivity of edges between the bright and dark areas. u13.0, default 0.

reserved1
reserved

corner_sense_0
Sensitivity of corner in dark area. u13.0, default 0.

reserved2
reserved

delta_corner_sense
Difference in the sensitivity of corners between the bright and dark areas. u13.0, default 8191.

reserved3
reserved

struct **ipu3_uapi_yuvp1_y_ee_nr_gain_config**
Luma(Y) edge enhancement noise reduction gain config

Definition

```
struct ipu3_uapi_yuvp1_y_ee_nr_gain_config {
  __u32 gain_pos_0:5;
  __u32 reserved0:3;
  __u32 delta_gain_posi:5;
  __u32 reserved1:3;
  __u32 gain_neg_0:5;
  __u32 reserved2:3;
  __u32 delta_gain_neg:5;
  __u32 reserved3:3;
};
```

Members

gain_pos_0
Gain for positive edge in dark area. u5.0, [0, 16], default 2.

reserved0
reserved

delta_gain_posi
Difference in the gain of edges between the bright and dark areas for positive edges. u5.0, [0, 16], default 0.

reserved1
reserved

gain_neg_0
Gain for negative edge in dark area. u5.0, [0, 16], default 8.

reserved2
reserved

delta_gain_neg
Difference in the gain of edges between the bright and dark areas for negative edges. u5.0, [0, 16], default 0.

reserved3
reserved

struct **ipu3_uapi_yuvp1_y_ee_nr_clip_config**
Luma(Y) edge enhancement noise reduction clipping config

Definition

```
struct ipu3_uapi_yuvp1_y_ee_nr_clip_config {
  __u32 clip_pos_0:5;
  __u32 reserved0:3;
  __u32 delta_clip_posi:5;
  __u32 reserved1:3;
  __u32 clip_neg_0:5;
  __u32 reserved2:3;
  __u32 delta_clip_neg:5;
  __u32 reserved3:3;
};
```

Members

clip_pos_0
Limit of positive edge in dark area u5, value [0, 16], default 8.

reserved0
reserved

delta_clip_posi
Difference in the limit of edges between the bright and dark areas for positive edges. u5, value [0, 16], default 8.

reserved1
reserved

clip_neg_0
Limit of negative edge in dark area u5, value [0, 16], default 8.

reserved2
reserved

delta_clip_neg
Difference in the limit of edges between the bright and dark areas for negative edges. u5, value [0, 16], default 8.

reserved3
reserved

struct **ipu3_uapi_yuvp1_y_ee_nr_frng_config**
Luma(Y) edge enhancement noise reduction fringe config

Definition

```
struct ipu3_uapi_yuvp1_y_ee_nr_frng_config {
  __u32 gain_exp:4;
  __u32 reserved0:28;
  __u32 min_edge:13;
  __u32 reserved1:3;
  __u32 lin_seg_param:4;
  __u32 reserved2:4;
  __u32 t1:1;
  __u32 t2:1;
  __u32 reserved3:6;
};
```

Members

gain_exp
Common exponent of gains, u4, [0, 8], default 2.

reserved0
reserved

min_edge
Threshold for edge and smooth stitching, u13.

reserved1
reserved

lin_seg_param
Power of LinSeg, u4.

reserved2
reserved

t1
Parameter for enabling/disabling the edge enhancement, u1.0, [0, 1], default 1.

t2
Parameter for enabling/disabling the smoothing, u1.0, [0, 1], default 1.

reserved3
reserved

struct **ipu3_uapi_yuvp1_y_ee_nr_diag_config**

Luma(Y) edge enhancement noise reduction diagonal config

Definition

```
struct ipu3_uapi_yuvp1_y_ee_nr_diag_config {
  __u32 diag_disc_g:4;
  __u32 reserved0:4;
  __u32 hvw_hor:4;
  __u32 dw_hor:4;
  __u32 hvw_diag:4;
  __u32 dw_diag:4;
  __u32 reserved1:8;
};
```

Members

diag_disc_g
: Coefficient that prioritize diagonal edge direction on horizontal or vertical for final enhancement. u4.0, [1, 15], default 1.

reserved0
: reserved

hvw_hor
: Weight of horizontal/vertical edge enhancement for hv edge. u2.2, [1, 15], default 4.

dw_hor
: Weight of diagonal edge enhancement for hv edge. u2.2, [1, 15], default 1.

hvw_diag
: Weight of horizontal/vertical edge enhancement for diagonal edge. u2.2, [1, 15], default 1.

dw_diag
: Weight of diagonal edge enhancement for diagonal edge. u2.2, [1, 15], default 4.

reserved1
: reserved

struct **ipu3_uapi_yuvp1_y_ee_nr_fc_coring_config**

Luma(Y) edge enhancement noise reduction false color correction (FCC) coring config

Definition

```
struct ipu3_uapi_yuvp1_y_ee_nr_fc_coring_config {
  __u32 pos_0:13;
  __u32 reserved0:3;
  __u32 pos_delta:13;
  __u32 reserved1:3;
  __u32 neg_0:13;
  __u32 reserved2:3;
  __u32 neg_delta:13;
  __u32 reserved3:3;
};
```

Members

pos_0
: Gain for positive edge in dark, u13.0, [0, 16], default 0.

reserved0
: reserved

pos_delta
: Gain for positive edge in bright, value: pos_0 + pos_delta <=16 u13.0, default 0.

reserved1
: reserved

neg_0
Gain for negative edge in dark area, u13.0, range [0, 16], default 0.

reserved2
reserved

neg_delta
Gain for negative edge in bright area. neg_0 + neg_delta <=16 u13.0, default 0.

reserved3
reserved

Description

Coring is a simple soft thresholding technique.

struct **ipu3_uapi_yuvp1_y_ee_nr_config**
Edge enhancement and noise reduction

Definition

```
struct ipu3_uapi_yuvp1_y_ee_nr_config {
  struct ipu3_uapi_yuvp1_y_ee_nr_lpf_config lpf;
  struct ipu3_uapi_yuvp1_y_ee_nr_sense_config sense;
  struct ipu3_uapi_yuvp1_y_ee_nr_gain_config gain;
  struct ipu3_uapi_yuvp1_y_ee_nr_clip_config clip;
  struct ipu3_uapi_yuvp1_y_ee_nr_frng_config frng;
  struct ipu3_uapi_yuvp1_y_ee_nr_diag_config diag;
  struct ipu3_uapi_yuvp1_y_ee_nr_fc_coring_config fc_coring;
};
```

Members

lpf
low-pass filter config. See *ipu3_uapi_yuvp1_y_ee_nr_lpf_config*

sense
sensitivity config. See *ipu3_uapi_yuvp1_y_ee_nr_sense_config*

gain
gain config as defined in *ipu3_uapi_yuvp1_y_ee_nr_gain_config*

clip
clip config as defined in *ipu3_uapi_yuvp1_y_ee_nr_clip_config*

frng
fringe config as defined in *ipu3_uapi_yuvp1_y_ee_nr_frng_config*

diag
diagonal edge config. See *ipu3_uapi_yuvp1_y_ee_nr_diag_config*

fc_coring
coring config for fringe control. See *ipu3_uapi_yuvp1_y_ee_nr_fc_coring_config*

struct **ipu3_uapi_yuvp2_tcc_gen_control_static_config**
Total color correction general control config

Definition

```
struct ipu3_uapi_yuvp2_tcc_gen_control_static_config {
  __u32 en:1;
  __u32 blend_shift:3;
  __u32 gain_according_to_y_only:1;
  __u32 reserved0:11;
  __s32 gamma:5;
  __u32 reserved1:3;
  __s32 delta:5;
  __u32 reserved2:3;
};
```

Members

en
: 0 - TCC disabled. Output = input 1 - TCC enabled.

blend_shift
: blend shift, Range[3, 4], default NA.

gain_according_to_y_only
: 0: Gain is calculated according to YUV, 1: Gain is calculated according to Y only

reserved0
: reserved

gamma
: Final blending coefficients. Values[-16, 16], default NA.

reserved1
: reserved

delta
: Final blending coefficients. Values[-16, 16], default NA.

reserved2
: reserved

struct **ipu3_uapi_yuvp2_tcc_macc_elem_static_config**

Total color correction multi-axis color control (MACC) config

Definition

```
struct ipu3_uapi_yuvp2_tcc_macc_elem_static_config {
  __s32 a:12;
  __u32 reserved0:4;
  __s32 b:12;
  __u32 reserved1:4;
  __s32 c:12;
  __u32 reserved2:4;
  __s32 d:12;
  __u32 reserved3:4;
};
```

Members

a
a coefficient for 2x2 MACC conversion matrix.

reserved0
reserved

b
b coefficient 2x2 MACC conversion matrix.

reserved1
reserved

c
c coefficient for 2x2 MACC conversion matrix.

reserved2
reserved

d
d coefficient for 2x2 MACC conversion matrix.

reserved3
reserved

struct **ipu3_uapi_yuvp2_tcc_macc_table_static_config**
Total color correction multi-axis color control (MACC) table array

Definition

```
struct ipu3_uapi_yuvp2_tcc_macc_table_static_config {
  struct ipu3_uapi_yuvp2_tcc_macc_elem_static_config entries[IPU3_
→UAPI_YUVP2_TCC_MACC_TABLE_ELEMENTS];
};
```

Members

entries
config for multi axis color correction, as specified by *ipu3_uapi_yuvp2_tcc_macc_elem_static_config*

struct **ipu3_uapi_yuvp2_tcc_inv_y_lut_static_config**
Total color correction inverse y lookup table

Definition

```
struct ipu3_uapi_yuvp2_tcc_inv_y_lut_static_config {
  __u16 entries[IPU3_UAPI_YUVP2_TCC_INV_Y_LUT_ELEMENTS];
};
```

Members

entries
lookup table for inverse y estimation, and use it to estimate the ratio between luma and chroma. Chroma by approximate the absolute value of the radius on the chroma plane (R = sqrt(u^2+v^2)) and luma by approximate by 1/Y.

struct **ipu3_uapi_yuvp2_tcc_gain_pcwl_lut_static_config**
Total color correction lookup table for PCWL

Definition

```
struct ipu3_uapi_yuvp2_tcc_gain_pcwl_lut_static_config {
  __u16 entries[IPU3_UAPI_YUVP2_TCC_GAIN_PCWL_LUT_ELEMENTS];
};
```

Members

entries
lookup table for gain piece wise linear transformation (PCWL)

struct **ipu3_uapi_yuvp2_tcc_r_sqr_lut_static_config**
Total color correction lookup table for r square root

Definition

```
struct ipu3_uapi_yuvp2_tcc_r_sqr_lut_static_config {
  __s16 entries[IPU3_UAPI_YUVP2_TCC_R_SQR_LUT_ELEMENTS];
};
```

Members

entries
lookup table for r square root estimation

struct **ipu3_uapi_yuvp2_tcc_static_config**
Total color correction static

Definition

```
struct ipu3_uapi_yuvp2_tcc_static_config {
  struct ipu3_uapi_yuvp2_tcc_gen_control_static_config gen_control;
  struct ipu3_uapi_yuvp2_tcc_macc_table_static_config macc_table;
  struct ipu3_uapi_yuvp2_tcc_inv_y_lut_static_config inv_y_lut;
  struct ipu3_uapi_yuvp2_tcc_gain_pcwl_lut_static_config gain_pcwl;
  struct ipu3_uapi_yuvp2_tcc_r_sqr_lut_static_config r_sqr_lut;
};
```

Members

gen_control
general config for Total Color Correction

macc_table
config for multi axis color correction

inv_y_lut
lookup table for inverse y estimation

gain_pcwl
lookup table for gain PCWL

r_sqr_lut
lookup table for r square root estimation.

struct **ipu3_uapi_anr_transform_config**
Advanced noise reduction transform

Definition

```
struct ipu3_uapi_anr_transform_config {
  __u32 enable:1;
  __u32 adaptive_treshhold_en:1;
  __u32 reserved1:30;
  __u8 reserved2[44];
  struct ipu3_uapi_anr_alpha alpha[3];
  struct ipu3_uapi_anr_beta beta[3];
  struct ipu3_uapi_anr_plane_color color[3];
  __u16 sqrt_lut[IPU3_UAPI_ANR_LUT_SIZE];
  __s16 xreset:13;
  __u16 reserved3:3;
  __s16 yreset:13;
  __u16 reserved4:3;
  __u32 x_sqr_reset:24;
  __u32 r_normfactor:5;
  __u32 reserved5:3;
  __u32 y_sqr_reset:24;
  __u32 gain_scale:8;
};
```

Members

enable
advanced noise reduction enabled.

adaptive_treshhold_en
On IPU3, adaptive threshold is always enabled.

reserved1
reserved

reserved2
reserved

alpha
using following defaults: 13, 13, 13, 13, 0, 0, 0, 0 11, 11, 11, 11, 0, 0, 0, 0 14, 14, 14, 14, 0, 0, 0, 0

beta
use following defaults: 24, 24, 24, 24 21, 20, 20, 21 25, 25, 25, 25

color
use defaults defined in driver/media/pci/intel/ipu3-tables.c

sqrt_lut
11 bits per element, values = [724 768 810 849 887 923 958 991 1024 1056 1116 1145 1173 1201 1086 1228 1254 1280 1305 1330 1355 1379 1402 1425 1448]

xreset
Reset value of X for r^2 calculation Value: col_start-X_center Constraint: Xreset + FrameWdith=4095 Xreset= -4095, default -1632.

reserved3

reserved

yreset
Reset value of Y for r^2 calculation Value: row_start-Y_center Constraint: Yreset + FrameHeight=4095 Yreset= -4095, default -1224.

reserved4
reserved

x_sqr_reset
Reset value of X^2 for r^2 calculation Value = (Xreset)^2

r_normfactor
Normalization factor for R. Default 14.

reserved5
reserved

y_sqr_reset
Reset value of Y^2 for r^2 calculation Value = (Yreset)^2

gain_scale
Parameter describing shading gain as a function of distance from the image center. A single value per frame, loaded by the driver. Default 115.

struct **ipu3_uapi_anr_stitch_pyramid**
ANR stitch pyramid

Definition

```
struct ipu3_uapi_anr_stitch_pyramid {
  __u32 entry0:6;
  __u32 entry1:6;
  __u32 entry2:6;
  __u32 reserved:14;
};
```

Members

entry0
pyramid LUT entry0, range [0x0, 0x3f]

entry1
pyramid LUT entry1, range [0x0, 0x3f]

entry2
pyramid LUT entry2, range [0x0, 0x3f]

reserved
reserved

struct **ipu3_uapi_anr_stitch_config**
ANR stitch config

Definition

```
struct ipu3_uapi_anr_stitch_config {
  __u32 anr_stitch_en;
```

(continues on next page)

(continued from previous page)

```
  __u8 reserved[44];
  struct ipu3_uapi_anr_stitch_pyramid pyramid[IPU3_UAPI_ANR_PYRAMID_
↪SIZE];
};
```

Members

anr_stitch_en
enable stitch. Enabled with 1.

reserved
reserved

pyramid
pyramid table as defined by *ipu3_uapi_anr_stitch_pyramid* default values: { 1, 3, 5 }, { 7, 7, 5 }, { 3, 1, 3 }, { 9, 15, 21 }, { 21, 15, 9 }, { 3, 5, 15 }, { 25, 35, 35 }, { 25, 15, 5 }, { 7, 21, 35 }, { 49, 49, 35 }, { 21, 7, 7 }, { 21, 35, 49 }, { 49, 35, 21 }, { 7, 5, 15 }, { 25, 35, 35 }, { 25, 15, 5 }, { 3, 9, 15 }, { 21, 21, 15 }, { 9, 3, 1 }, { 3, 5, 7 }, { 7, 5, 3}, { 1 }

struct **ipu3_uapi_anr_config**
ANR config

Definition

```
struct ipu3_uapi_anr_config {
  struct ipu3_uapi_anr_transform_config transform ;
  struct ipu3_uapi_anr_stitch_config stitch ;
};
```

Members

transform
advanced noise reduction transform config as specified by *ipu3_uapi_anr_transform_config*

stitch
create 4x4 patch from 4 surrounding 8x8 patches.

struct **ipu3_uapi_acc_param**
Accelerator cluster parameters

Definition

```
struct ipu3_uapi_acc_param {
  struct ipu3_uapi_bnr_static_config bnr;
  struct ipu3_uapi_bnr_static_config_green_disparity green_
↪disparity ;
  struct ipu3_uapi_dm_config dm ;
  struct ipu3_uapi_ccm_mat_config ccm ;
  struct ipu3_uapi_gamma_config gamma ;
  struct ipu3_uapi_csc_mat_config csc ;
  struct ipu3_uapi_cds_params cds ;
  struct ipu3_uapi_shd_config shd ;
```

(continues on next page)

(continued from previous page)

```
  struct ipu3_uapi_yuvp1_iefd_config iefd ;
  struct ipu3_uapi_yuvp1_yds_config yds_c0 ;
  struct ipu3_uapi_yuvp1_chnr_config chnr_c0 ;
  struct ipu3_uapi_yuvp1_y_ee_nr_config y_ee_nr ;
  struct ipu3_uapi_yuvp1_yds_config yds ;
  struct ipu3_uapi_yuvp1_chnr_config chnr ;
  struct ipu3_uapi_yuvp1_yds_config yds2 ;
  struct ipu3_uapi_yuvp2_tcc_static_config tcc ;
  struct ipu3_uapi_anr_config anr;
  struct ipu3_uapi_awb_fr_config_s awb_fr;
  struct ipu3_uapi_ae_config ae;
  struct ipu3_uapi_af_config_s af;
  struct ipu3_uapi_awb_config awb;
};
```

Members

bnr
: parameters for bayer noise reduction static config. See *ipu3_uapi_bnr_static_config*

green_disparity
: disparity static config between gr and gb channel. See *ipu3_uapi_bnr_static_config_green_disparity*

dm
: de-mosaic config. See *ipu3_uapi_dm_config*

ccm
: color correction matrix. See *ipu3_uapi_ccm_mat_config*

gamma
: gamma correction config. See *ipu3_uapi_gamma_config*

csc
: color space conversion matrix. See *ipu3_uapi_csc_mat_config*

cds
: color down sample config. See *ipu3_uapi_cds_params*

shd
: lens shading correction config. See *ipu3_uapi_shd_config*

iefd
: Image enhancement filter and denoise config. *ipu3_uapi_yuvp1_iefd_config*

yds_c0
: y down scaler config. *ipu3_uapi_yuvp1_yds_config*

chnr_c0
: chroma noise reduction config. *ipu3_uapi_yuvp1_chnr_config*

y_ee_nr
: y edge enhancement and noise reduction config. *ipu3_uapi_yuvp1_y_ee_nr_config*

yds
y down scaler config. See *ipu3_uapi_yuvp1_yds_config*

chnr
chroma noise reduction config. See *ipu3_uapi_yuvp1_chnr_config*

yds2
y channel down scaler config. See *ipu3_uapi_yuvp1_yds_config*

tcc
total color correction config as defined in struct *ipu3_uapi_yuvp2_tcc_static_config*

anr
advanced noise reduction config.See *ipu3_uapi_anr_config*

awb_fr
AWB filter response config. See ipu3_uapi_awb_fr_config

ae
auto exposure config As specified by *ipu3_uapi_ae_config*

af
auto focus config. As specified by ipu3_uapi_af_config

awb
auto white balance config. As specified by *ipu3_uapi_awb_config*

Description

ACC refers to the HW cluster containing all Fixed Functions (FFs). Each FF implements a specific algorithm.

struct **ipu3_uapi_isp_lin_vmem_params**
Linearization parameters

Definition

```
struct ipu3_uapi_isp_lin_vmem_params {
  __s16 lin_lutlow_gr[IPU3_UAPI_LIN_LUT_SIZE];
  __s16 lin_lutlow_r[IPU3_UAPI_LIN_LUT_SIZE];
  __s16 lin_lutlow_b[IPU3_UAPI_LIN_LUT_SIZE];
  __s16 lin_lutlow_gb[IPU3_UAPI_LIN_LUT_SIZE];
  __s16 lin_lutdif_gr[IPU3_UAPI_LIN_LUT_SIZE];
  __s16 lin_lutdif_r[IPU3_UAPI_LIN_LUT_SIZE];
  __s16 lin_lutdif_b[IPU3_UAPI_LIN_LUT_SIZE];
  __s16 lin_lutdif_gb[IPU3_UAPI_LIN_LUT_SIZE];
};
```

Members

lin_lutlow_gr
linearization look-up table for GR channel interpolation.

lin_lutlow_r
linearization look-up table for R channel interpolation.

lin_lutlow_b
linearization look-up table for B channel interpolation.

lin_lutlow_gb
linearization look-up table for GB channel interpolation. lin_lutlow_gr / lin_lutlow_r / lin_lutlow_b / lin_lutlow_gb <= LIN_MAX_VALUE - 1.

lin_lutdif_gr
lin_lutlow_gr[i+1] - lin_lutlow_gr[i].

lin_lutdif_r
lin_lutlow_r[i+1] - lin_lutlow_r[i].

lin_lutdif_b
lin_lutlow_b[i+1] - lin_lutlow_b[i].

lin_lutdif_gb
lin_lutlow_gb[i+1] - lin_lutlow_gb[i].

struct **ipu3_uapi_isp_tnr3_vmem_params**
Temporal noise reduction vector memory parameters

Definition

```
struct ipu3_uapi_isp_tnr3_vmem_params {
  __u16 slope[IPU3_UAPI_ISP_TNR3_VMEM_LEN];
  __u16 reserved1[IPU3_UAPI_ISP_VEC_ELEMS - IPU3_UAPI_ISP_TNR3_VMEM_
→LEN];
  __u16 sigma[IPU3_UAPI_ISP_TNR3_VMEM_LEN];
  __u16 reserved2[IPU3_UAPI_ISP_VEC_ELEMS - IPU3_UAPI_ISP_TNR3_VMEM_
→LEN];
};
```

Members

slope
slope setting in interpolation curve for temporal noise reduction.

reserved1
reserved

sigma
knee point setting in interpolation curve for temporal noise reduction.

reserved2
reserved

struct **ipu3_uapi_isp_tnr3_params**
Temporal noise reduction v3 parameters

Definition

```
struct ipu3_uapi_isp_tnr3_params {
  __u32 knee_y1;
  __u32 knee_y2;
  __u32 maxfb_y;
  __u32 maxfb_u;
  __u32 maxfb_v;
  __u32 round_adj_y;
```

(continues on next page)

(continued from previous page)

```
  __u32 round_adj_u;
  __u32 round_adj_v;
  __u32 ref_buf_select;
};
```

Members

knee_y1
Knee point TNR3 assumes standard deviation of Y,U and V at Y1 are TnrY1_Sigma_Y, U and V.

knee_y2
Knee point TNR3 assumes standard deviation of Y,U and V at Y2 are TnrY2_Sigma_Y, U and V.

maxfb_y
Max feedback gain for Y

maxfb_u
Max feedback gain for U

maxfb_v
Max feedback gain for V

round_adj_y
rounding Adjust for Y

round_adj_u
rounding Adjust for U

round_adj_v
rounding Adjust for V

ref_buf_select
selection of the reference frame buffer to be used.

struct **ipu3_uapi_isp_xnr3_vmem_params**
Extreme noise reduction v3 vector memory parameters

Definition

```
struct ipu3_uapi_isp_xnr3_vmem_params {
  __u16 x[IPU3_UAPI_ISP_VEC_ELEMS];
  __u16 a[IPU3_UAPI_ISP_VEC_ELEMS];
  __u16 b[IPU3_UAPI_ISP_VEC_ELEMS];
  __u16 c[IPU3_UAPI_ISP_VEC_ELEMS];
};
```

Members

x
xnr3 parameters.

a
xnr3 parameters.

b
xnr3 parameters.

c
xnr3 parameters.

struct **ipu3_uapi_xnr3_alpha_params**
Extreme noise reduction v3 alpha tuning parameters

Definition

```
struct ipu3_uapi_xnr3_alpha_params {
  __u32 y0;
  __u32 u0;
  __u32 v0;
  __u32 ydiff;
  __u32 udiff;
  __u32 vdiff;
};
```

Members

y0
Sigma for Y range similarity in dark area.

u0
Sigma for U range similarity in dark area.

v0
Sigma for V range similarity in dark area.

ydiff
Sigma difference for Y between bright area and dark area.

udiff
Sigma difference for U between bright area and dark area.

vdiff
Sigma difference for V between bright area and dark area.

struct **ipu3_uapi_xnr3_coring_params**
Extreme noise reduction v3 coring parameters

Definition

```
struct ipu3_uapi_xnr3_coring_params {
  __u32 u0;
  __u32 v0;
  __u32 udiff;
  __u32 vdiff;
};
```

Members

u0
Coring Threshold of U channel in dark area.

v0
Coring Threshold of V channel in dark area.

udiff
Threshold difference of U channel between bright and dark area.

vdiff
Threshold difference of V channel between bright and dark area.

struct **ipu3_uapi_xnr3_blending_params**
Blending factor

Definition

```
struct ipu3_uapi_xnr3_blending_params {
  __u32 strength;
};
```

Members

strength
The factor for blending output with input. This is tuning parameterHigher values lead to more aggressive XNR operation.

struct **ipu3_uapi_isp_xnr3_params**
Extreme noise reduction v3 parameters

Definition

```
struct ipu3_uapi_isp_xnr3_params {
  struct ipu3_uapi_xnr3_alpha_params alpha;
  struct ipu3_uapi_xnr3_coring_params coring;
  struct ipu3_uapi_xnr3_blending_params blending;
};
```

Members

alpha
parameters for xnr3 alpha. See *ipu3_uapi_xnr3_alpha_params*

coring
parameters for xnr3 coring. See *ipu3_uapi_xnr3_coring_params*

blending
parameters for xnr3 blending. See *ipu3_uapi_xnr3_blending_params*

struct **ipu3_uapi_obgrid_param**
Optical black level compensation parameters

Definition

```
struct ipu3_uapi_obgrid_param {
  __u16 gr;
  __u16 r;
  __u16 b;
  __u16 gb;
};
```

Members

gr
    Grid table values for color GR

r
    Grid table values for color R

b
    Grid table values for color B

gb
    Grid table values for color GB

Description

Black level is different for red, green, and blue channels. So black level compensation is different per channel.

struct **ipu3_uapi_flags**
    bits to indicate which pipeline needs update

Definition

```
struct ipu3_uapi_flags {
  __u32 gdc:1;
  __u32 obgrid:1;
  __u32 reserved1:30;
  __u32 acc_bnr:1;
  __u32 acc_green_disparity:1;
  __u32 acc_dm:1;
  __u32 acc_ccm:1;
  __u32 acc_gamma:1;
  __u32 acc_csc:1;
  __u32 acc_cds:1;
  __u32 acc_shd:1;
  __u32 reserved2:2;
  __u32 acc_iefd:1;
  __u32 acc_yds_c0:1;
  __u32 acc_chnr_c0:1;
  __u32 acc_y_ee_nr:1;
  __u32 acc_yds:1;
  __u32 acc_chnr:1;
  __u32 acc_ytm:1;
  __u32 acc_yds2:1;
  __u32 acc_tcc:1;
  __u32 acc_dpc:1;
  __u32 acc_bds:1;
  __u32 acc_anr:1;
  __u32 acc_awb_fr:1;
  __u32 acc_ae:1;
  __u32 acc_af:1;
  __u32 acc_awb:1;
  __u32 reserved3:4;
```

(continues on next page)

(continued from previous page)

```
    __u32 lin_vmem_params:1;
    __u32 tnr3_vmem_params:1;
    __u32 xnr3_vmem_params:1;
    __u32 tnr3_dmem_params:1;
    __u32 xnr3_dmem_params:1;
    __u32 reserved4:1;
    __u32 obgrid_param:1;
    __u32 reserved5:25;
};
```

Members

gdc
    0 = no update, 1 = update.

obgrid
    0 = no update, 1 = update.

reserved1
    Not used.

acc_bnr
    0 = no update, 1 = update.

acc_green_disparity
    0 = no update, 1 = update.

acc_dm
    0 = no update, 1 = update.

acc_ccm
    0 = no update, 1 = update.

acc_gamma
    0 = no update, 1 = update.

acc_csc
    0 = no update, 1 = update.

acc_cds
    0 = no update, 1 = update.

acc_shd
    0 = no update, 1 = update.

reserved2
    Not used.

acc_iefd
    0 = no update, 1 = update.

acc_yds_c0
    0 = no update, 1 = update.

acc_chnr_c0
    0 = no update, 1 = update.

acc_y_ee_nr
0 = no update, 1 = update.

acc_yds
0 = no update, 1 = update.

acc_chnr
0 = no update, 1 = update.

acc_ytm
0 = no update, 1 = update.

acc_yds2
0 = no update, 1 = update.

acc_tcc
0 = no update, 1 = update.

acc_dpc
0 = no update, 1 = update.

acc_bds
0 = no update, 1 = update.

acc_anr
0 = no update, 1 = update.

acc_awb_fr
0 = no update, 1 = update.

acc_ae
0 = no update, 1 = update.

acc_af
0 = no update, 1 = update.

acc_awb
0 = no update, 1 = update.

reserved3
Not used.

lin_vmem_params
0 = no update, 1 = update.

tnr3_vmem_params
0 = no update, 1 = update.

xnr3_vmem_params
0 = no update, 1 = update.

tnr3_dmem_params
0 = no update, 1 = update.

xnr3_dmem_params
0 = no update, 1 = update.

reserved4
Not used.

obgrid_param
0 = no update, 1 = update.

reserved5
Not used.

struct **ipu3_uapi_params**

V4L2_META_FMT_IPU3_PARAMS

Definition

```
struct ipu3_uapi_params {
  struct ipu3_uapi_flags use ;
  struct ipu3_uapi_acc_param acc_param;
  struct ipu3_uapi_isp_lin_vmem_params lin_vmem_params;
  struct ipu3_uapi_isp_tnr3_vmem_params tnr3_vmem_params;
  struct ipu3_uapi_isp_xnr3_vmem_params xnr3_vmem_params;
  struct ipu3_uapi_isp_tnr3_params tnr3_dmem_params;
  struct ipu3_uapi_isp_xnr3_params xnr3_dmem_params;
  struct ipu3_uapi_obgrid_param obgrid_param;
};
```

Members

use
select which parameters to apply, see *ipu3_uapi_flags*

acc_param
ACC parameters, as specified by *ipu3_uapi_acc_param*

lin_vmem_params
linearization VMEM, as specified by *ipu3_uapi_isp_lin_vmem_params*

tnr3_vmem_params
tnr3 VMEM as specified by *ipu3_uapi_isp_tnr3_vmem_params*

xnr3_vmem_params
xnr3 VMEM as specified by *ipu3_uapi_isp_xnr3_vmem_params*

tnr3_dmem_params
tnr3 DMEM as specified by *ipu3_uapi_isp_tnr3_params*

xnr3_dmem_params
xnr3 DMEM as specified by *ipu3_uapi_isp_xnr3_params*

obgrid_param
obgrid parameters as specified by *ipu3_uapi_obgrid_param*

Description

The video queue "parameters" is of format V4L2_META_FMT_IPU3_PARAMS. This is a "single plane" v4l2_meta_format using V4L2_BUF_TYPE_META_OUTPUT.

struct ipu3_uapi_params as defined below contains a lot of parameters and ipu3_uapi_flags selects which parameters to apply.

V4L2_META_FMT_RK_ISP1_PARAMS ('rk1p'), V4L2_META_FMT_RK_ISP1_STAT_3A ('rk1s')

Configuration parameters

The configuration parameters are passed to the rkisp1_params metadata output video node, using the *v4l2_meta_format* interface. The buffer contains a single instance of the C structure *rkisp1_params_cfg* defined in `rkisp1-config.h`. So the structure can be obtained from the buffer by:

```
struct rkisp1_params_cfg *params = (struct rkisp1_params_cfg*)␣
↪buffer;
```

3A and histogram statistics

The ISP1 device collects different statistics over an input Bayer frame. Those statistics are obtained from the rkisp1_stats metadata capture video node, using the *v4l2_meta_format* interface. The buffer contains a single instance of the C structure *rkisp1_stat_buffer* defined in `rkisp1-config.h`. So the structure can be obtained from the buffer by:

```
struct rkisp1_stat_buffer *stats = (struct rkisp1_stat_buffer*)␣
↪buffer;
```

The statistics collected are Exposure, AWB (Auto-white balance), Histogram and AF (Auto-focus). See *rkisp1_stat_buffer* for details of the statistics.

The 3A statistics and configuration parameters described here are usually consumed and produced by dedicated user space libraries that comprise the important tuning tools using software control loop.

rkisp1 uAPI data types

enum **`rkisp1_cif_isp_exp_ctrl_autostop`**

stop modes

Constants

`RKISP1_CIF_ISP_EXP_CTRL_AUTOSTOP_0`
continuous measurement

`RKISP1_CIF_ISP_EXP_CTRL_AUTOSTOP_1`
stop measuring after a complete frame

enum **`rkisp1_cif_isp_exp_meas_mode`**

Exposure measure mode

Constants

`RKISP1_CIF_ISP_EXP_MEASURING_MODE_0`
Y = 16 + 0.25R + 0.5G + 0.1094B

RKISP1_CIF_ISP_EXP_MEASURING_MODE_1
Y = (R + G + B) x (85/256)

struct **rkisp1_cif_isp_window**
measurement window.

Definition

```
struct rkisp1_cif_isp_window {
  __u16 h_offs;
  __u16 v_offs;
  __u16 h_size;
  __u16 v_size;
};
```

Members

h_offs
the horizontal offset of the window from the left of the frame in pixels.

v_offs
the vertical offset of the window from the top of the frame in pixels.

h_size
the horizontal size of the window in pixels

v_size
the vertical size of the window in pixels.

Description

Measurements are calculated per window inside the frame. This struct represents a window for a measurement.

struct **rkisp1_cif_isp_bls_fixed_val**
BLS fixed subtraction values

Definition

```
struct rkisp1_cif_isp_bls_fixed_val {
  __s16 r;
  __s16 gr;
  __s16 gb;
  __s16 b;
};
```

Members

r
Fixed (signed!) subtraction value for Bayer pattern R

gr
Fixed (signed!) subtraction value for Bayer pattern Gr

gb
Fixed (signed!) subtraction value for Bayer pattern Gb

b
Fixed (signed!) subtraction value for Bayer pattern B

Description

The values will be subtracted from the sensor values. Therefore a negative value means addition instead of subtraction!

struct **rkisp1_cif_isp_bls_config**
Configuration used by black level subtraction

Definition

```
struct rkisp1_cif_isp_bls_config {
  __u8 enable_auto;
  __u8 en_windows;
  struct rkisp1_cif_isp_window bls_window1;
  struct rkisp1_cif_isp_window bls_window2;
  __u8 bls_samples;
  struct rkisp1_cif_isp_bls_fixed_val fixed_val;
};
```

Members

enable_auto
Automatic mode activated means that the measured values are subtracted. Otherwise the fixed subtraction values will be subtracted.

en_windows
enabled window

bls_window1
Measurement window 1 size

bls_window2
Measurement window 2 size

bls_samples
Set amount of measured pixels for each Bayer position (A, B,C and D) to 2^bls_samples.

fixed_val
Fixed subtraction values

struct **rkisp1_cif_isp_dpcc_methods_config**
Methods Configuration used by DPCC

Definition

```
struct rkisp1_cif_isp_dpcc_methods_config {
  __u32 method;
  __u32 line_thresh;
  __u32 line_mad_fac;
  __u32 pg_fac;
  __u32 rnd_thresh;
  __u32 rg_fac;
};
```

Members

method
Method enable bits

line_thresh
Line threshold

line_mad_fac
Line MAD factor

pg_fac
Peak gradient factor

rnd_thresh
Rank Neighbor Difference threshold

rg_fac
Rank gradient factor

Description

Methods Configuration used by Defect Pixel Cluster Correction

struct **rkisp1_cif_isp_dpcc_config**
Configuration used by DPCC

Definition

```
struct rkisp1_cif_isp_dpcc_config {
  __u32 mode;
  __u32 output_mode;
  __u32 set_use;
  struct rkisp1_cif_isp_dpcc_methods_config methods[RKISP1_CIF_ISP_
→DPCC_METHODS_MAX];
  __u32 ro_limits;
  __u32 rnd_offs;
};
```

Members

mode
dpcc output mode

output_mode
whether use hard coded methods

set_use
stage1 methods set

methods
methods config

ro_limits
rank order limits

rnd_offs
differential rank offsets for rank neighbor difference

Description

Configuration used by Defect Pixel Cluster Correction

struct **rkisp1_cif_isp_gamma_corr_curve**

gamma curve point definition y-axis (output).

Definition

```
struct rkisp1_cif_isp_gamma_corr_curve {
  __u16 gamma_y[RKISP1_CIF_ISP_DEGAMMA_CURVE_SIZE];
};
```

Members

gamma_y
the values for the y-axis of gamma curve points. Each value is 12 bit.

Description

The reset values define a linear curve which has the same effect as bypass. Reset values are: gamma_y[0] = 0x0000, gamma_y[1] = 0x0100, …gamma_y[15] = 0x0f00, gamma_y[16] = 0xfff

struct **rkisp1_cif_isp_gamma_curve_x_axis_pnts**

De-Gamma Curve definition x increments (sampling points). gamma_dx0 is for the lower samples (1-8), gamma_dx1 is for the higher samples (9-16). The reset values for both fields is 0x44444444. This means that each sample is 4 units away from the previous one on the x-axis.

Definition

```
struct rkisp1_cif_isp_gamma_curve_x_axis_pnts {
  __u32 gamma_dx0;
  __u32 gamma_dx1;
};
```

Members

gamma_dx0
gamma curve sample points definitions. Bits 0:2 for sample 1. Bit 3 unused. Bits 4:6 for sample 2. bit 7 unused …Bits 28:30 for sample 8. Bit 31 unused

gamma_dx1
gamma curve sample points definitions. Bits 0:2 for sample 9. Bit 3 unused. Bits 4:6 for sample 10. bit 7 unused …Bits 28:30 for sample 16. Bit 31 unused

struct **rkisp1_cif_isp_sdg_config**

Configuration used by sensor degamma

Definition

```
struct rkisp1_cif_isp_sdg_config {
  struct rkisp1_cif_isp_gamma_corr_curve curve_r;
  struct rkisp1_cif_isp_gamma_corr_curve curve_g;
  struct rkisp1_cif_isp_gamma_corr_curve curve_b;
```

(continues on next page)

(continued from previous page)

```
  struct rkisp1_cif_isp_gamma_curve_x_axis_pnts xa_pnts;
};
```

Members

curve_r
gamma curve point definition axis for red

curve_g
gamma curve point definition axis for green

curve_b
gamma curve point definition axis for blue

xa_pnts
x axis increments

struct **rkisp1_cif_isp_lsc_config**
Configuration used by Lens shading correction

Definition

```
struct rkisp1_cif_isp_lsc_config {
  __u16 r_data_tbl[RKISP1_CIF_ISP_LSC_SAMPLES_MAX][RKISP1_CIF_ISP_
→LSC_SAMPLES_MAX];
  __u16 gr_data_tbl[RKISP1_CIF_ISP_LSC_SAMPLES_MAX][RKISP1_CIF_ISP_
→LSC_SAMPLES_MAX];
  __u16 gb_data_tbl[RKISP1_CIF_ISP_LSC_SAMPLES_MAX][RKISP1_CIF_ISP_
→LSC_SAMPLES_MAX];
  __u16 b_data_tbl[RKISP1_CIF_ISP_LSC_SAMPLES_MAX][RKISP1_CIF_ISP_
→LSC_SAMPLES_MAX];
  __u16 x_grad_tbl[RKISP1_CIF_ISP_LSC_SECTORS_TBL_SIZE];
  __u16 y_grad_tbl[RKISP1_CIF_ISP_LSC_SECTORS_TBL_SIZE];
  __u16 x_size_tbl[RKISP1_CIF_ISP_LSC_SECTORS_TBL_SIZE];
  __u16 y_size_tbl[RKISP1_CIF_ISP_LSC_SECTORS_TBL_SIZE];
  __u16 config_width;
  __u16 config_height;
};
```

Members

r_data_tbl
sample table red

gr_data_tbl
sample table green (red)

gb_data_tbl
sample table green (blue)

b_data_tbl
sample table blue

x_grad_tbl
gradient table x

y_grad_tbl
gradient table y

x_size_tbl
size table x

y_size_tbl
size table y

config_width
not used at the moment

config_height
not used at the moment

struct **rkisp1_cif_isp_ie_config**

Configuration used by image effects

Definition

```
struct rkisp1_cif_isp_ie_config {
  __u16 effect;
  __u16 color_sel;
  __u16 eff_mat_1;
  __u16 eff_mat_2;
  __u16 eff_mat_3;
  __u16 eff_mat_4;
  __u16 eff_mat_5;
  __u16 eff_tint;
};
```

Members

effect
values from 'enum v4l2_colorfx'. Possible values are: V4L2_COLORFX_SEPIA, V4L2_COLORFX_SET_CBCR, V4L2_COLORFX_AQUA, V4L2_COLORFX_EMBOSS, V4L2_COLORFX_SKETCH, V4L2_COLORFX_BW, V4L2_COLORFX_NEGATIVE

color_sel
bits 0:2 - colors bitmask (001 - blue, 010 - green, 100 - red). bits 8:15 - Threshold value of the RGB colors for the color selection effect.

eff_mat_1
3x3 Matrix Coefficients for Emboss Effect 1

eff_mat_2
3x3 Matrix Coefficients for Emboss Effect 2

eff_mat_3
3x3 Matrix Coefficients for Emboss 3/Sketch 1

eff_mat_4
3x3 Matrix Coefficients for Sketch Effect 2

eff_mat_5
3x3 Matrix Coefficients for Sketch Effect 3

eff_tint
Chrominance increment values of tint (used for sepia effect)

struct **rkisp1_cif_isp_cproc_config**
Configuration used by Color Processing

Definition

```
struct rkisp1_cif_isp_cproc_config {
  __u8 c_out_range;
  __u8 y_in_range;
  __u8 y_out_range;
  __u8 contrast;
  __u8 brightness;
  __u8 sat;
  __u8 hue;
};
```

Members

c_out_range
Chrominance pixel clipping range at output. (0 for limit, 1 for full)

y_in_range
Luminance pixel clipping range at output.

y_out_range
Luminance pixel clipping range at output.

contrast
00~ff, 0.0~1.992

brightness
80~7F, -128~+127

sat
saturation, 00~FF, 0.0~1.992

hue
80~7F, -90~+87.188

struct **rkisp1_cif_isp_awb_meas_config**
Configuration used by auto white balance

Definition

```
struct rkisp1_cif_isp_awb_meas_config {
  struct rkisp1_cif_isp_window awb_wnd;
  __u32 awb_mode;
  __u8 max_y;
  __u8 min_y;
  __u8 max_csum;
  __u8 min_c;
  __u8 frames;
  __u8 awb_ref_cr;
  __u8 awb_ref_cb;
```

(continues on next page)

(continued from previous page)

```
  __u8 enable_ymax_cmp;
};
```

Members

awb_wnd
white balance measurement window (in pixels)

awb_mode
the awb meas mode. From enum rkisp1_cif_isp_awb_mode_type.

max_y
only pixels values < max_y contribute to awb measurement, set to 0 to disable this feature

min_y
only pixels values > min_y contribute to awb measurement

max_csum
Chrominance sum maximum value, only consider pixels with Cb+Cr, smaller than threshold for awb measurements

min_c
Chrominance minimum value, only consider pixels with Cb/Cr each greater than threshold value for awb measurements

frames
number of frames - 1 used for mean value calculation (ucFrames=0 means 1 Frame)

awb_ref_cr
reference Cr value for AWB regulation, target for AWB

awb_ref_cb
reference Cb value for AWB regulation, target for AWB

enable_ymax_cmp
enable Y_MAX compare (Not valid in RGB measurement mode.)

struct **rkisp1_cif_isp_awb_gain_config**

Configuration used by auto white balance gain

Definition

```
struct rkisp1_cif_isp_awb_gain_config {
  __u16 gain_red;
  __u16 gain_green_r;
  __u16 gain_blue;
  __u16 gain_green_b;
};
```

Members

gain_red
gain value for red component.

gain_green_r
gain value for green component in red line.

gain_blue
gain value for blue component.

gain_green_b
gain value for green component in blue line.

Description

All fields in this struct are 10 bit, where: 0x100h = 1, unsigned integer value, range 0 to 4 with 8 bit fractional part.

out_data_x = (AWB_GAIN_X * in_data + 128) >> 8

struct **rkisp1_cif_isp_flt_config**
Configuration used by ISP filtering

Definition

```
struct rkisp1_cif_isp_flt_config {
  __u32 mode;
  __u8 grn_stage1;
  __u8 chr_h_mode;
  __u8 chr_v_mode;
  __u32 thresh_bl0;
  __u32 thresh_bl1;
  __u32 thresh_sh0;
  __u32 thresh_sh1;
  __u32 lum_weight;
  __u32 fac_sh1;
  __u32 fac_sh0;
  __u32 fac_mid;
  __u32 fac_bl0;
  __u32 fac_bl1;
};
```

Members

mode
ISP_FILT_MODE register fields (from enum rkisp1_cif_isp_flt_mode)

grn_stage1
Green filter stage 1 select (range 0x0⋯0x8)

chr_h_mode
Chroma filter horizontal mode

chr_v_mode
Chroma filter vertical mode

thresh_bl0
If thresh_bl1 < sum_grad < thresh_bl0 then fac_bl0 is selected (blurring th)

thresh_bl1
If sum_grad < thresh_bl1 then fac_bl1 is selected (blurring th)

thresh_sh0
If thresh_sh0 < sum_grad < thresh_sh1 then thresh_sh0 is selected (sharpening th)

thresh_sh1
If thresh_sh1 < sum_grad then thresh_sh1 is selected (sharpening th)

lum_weight
Parameters for luminance weight function.

fac_sh1
filter factor for sharp1 level

fac_sh0
filter factor for sharp0 level

fac_mid
filter factor for mid level and for static filter mode

fac_bl0
filter factor for blur 0 level

fac_bl1
filter factor for blur 1 level (max blur)

Description

All 4 threshold fields (thresh_*) are 10 bits. All 6 factor fields (fac_*) are 6 bits.

struct **rkisp1_cif_isp_bdm_config**
Configuration used by Bayer DeMosaic

Definition

```
struct rkisp1_cif_isp_bdm_config {
  __u8 demosaic_th;
};
```

Members

demosaic_th
threshold for bayer demosaicing texture detection

struct **rkisp1_cif_isp_ctk_config**
Configuration used by Cross Talk correction

Definition

```
struct rkisp1_cif_isp_ctk_config {
  __u16 coeff[3][3];
  __u16 ct_offset[3];
};
```

Members

coeff
color correction matrix. Values are 11-bit signed fixed-point numbers with 4 bit integer and 7 bit fractional part, ranging from -8 (0x400) to +7.992 (0x3FF). 0 is represented by 0x000 and a coefficient value of 1 as 0x080.

ct_offset
Red, Green, Blue offsets for the crosstalk correction matrix

struct **rkisp1_cif_isp_goc_config**
Configuration used by Gamma Out correction

Definition

```
struct rkisp1_cif_isp_goc_config {
  __u32 mode;
  __u16 gamma_y[RKISP1_CIF_ISP_GAMMA_OUT_MAX_SAMPLES];
};
```

Members

mode
goc mode (from enum rkisp1_cif_isp_goc_mode)

gamma_y
gamma out curve y-axis for all color components

struct **rkisp1_cif_isp_hst_config**
Configuration used by Histogram

Definition

```
struct rkisp1_cif_isp_hst_config {
  __u32 mode;
  __u8 histogram_predivider;
  struct rkisp1_cif_isp_window meas_window;
  __u8 hist_weight[RKISP1_CIF_ISP_HISTOGRAM_WEIGHT_GRIDS_SIZE];
};
```

Members

mode
histogram mode (from enum rkisp1_cif_isp_histogram_mode)

histogram_predivider
process every stepsize pixel, all other pixels are skipped

meas_window
coordinates of the measure window

hist_weight
weighting factor for sub-windows

struct **rkisp1_cif_isp_aec_config**
Configuration used by Auto Exposure Control

Definition

```
struct rkisp1_cif_isp_aec_config {
  __u32 mode;
  __u32 autostop;
  struct rkisp1_cif_isp_window meas_window;
};
```

Members

mode
Exposure measure mode (from *enum rkisp1_cif_isp_exp_meas_mode*)

autostop
stop mode (from *enum rkisp1_cif_isp_exp_ctrl_autostop*)

meas_window
coordinates of the measure window

struct **rkisp1_cif_isp_afc_config**
Configuration used by Auto Focus Control

Definition

```
struct rkisp1_cif_isp_afc_config {
  __u8 num_afm_win;
  struct rkisp1_cif_isp_window afm_win[RKISP1_CIF_ISP_AFM_MAX_
↪WINDOWS];
  __u32 thres;
  __u32 var_shift;
};
```

Members

num_afm_win
max RKISP1_CIF_ISP_AFM_MAX_WINDOWS

afm_win
coordinates of the meas window

thres
threshold used for minimizing the influence of noise

var_shift
the number of bits for the shift operation at the end of the calculation chain.

enum **rkisp1_cif_isp_dpf_gain_usage**
dpf gain usage

Constants

RKISP1_CIF_ISP_DPF_GAIN_USAGE_DISABLED
don't use any gains in preprocessing stage

RKISP1_CIF_ISP_DPF_GAIN_USAGE_NF_GAINS
use only the noise function gains from registers DPF_NF_GAIN_R, …

RKISP1_CIF_ISP_DPF_GAIN_USAGE_LSC_GAINS
use only the gains from LSC module

RKISP1_CIF_ISP_DPF_GAIN_USAGE_NF_LSC_GAINS
use the noise function gains and the gains from LSC module

RKISP1_CIF_ISP_DPF_GAIN_USAGE_AWB_GAINS
use only the gains from AWB module

RKISP1_CIF_ISP_DPF_GAIN_USAGE_AWB_LSC_GAINS
use the gains from AWB and LSC module

RKISP1_CIF_ISP_DPF_GAIN_USAGE_MAX
upper border (only for an internal evaluation)

enum **rkisp1_cif_isp_dpf_rb_filtersize**
Red and blue filter sizes

Constants

RKISP1_CIF_ISP_DPF_RB_FILTERSIZE_13x9
red and blue filter kernel size 13x9 (means 7x5 active pixel)

RKISP1_CIF_ISP_DPF_RB_FILTERSIZE_9x9
red and blue filter kernel size 9x9 (means 5x5 active pixel)

enum **rkisp1_cif_isp_dpf_nll_scale_mode**
dpf noise level scale mode

Constants

RKISP1_CIF_ISP_NLL_SCALE_LINEAR
use a linear scaling

RKISP1_CIF_ISP_NLL_SCALE_LOGARITHMIC
use a logarithmic scaling

struct **rkisp1_cif_isp_dpf_nll**
Noise level lookup

Definition

```
struct rkisp1_cif_isp_dpf_nll {
  __u16 coeff[RKISP1_CIF_ISP_DPF_MAX_NLF_COEFFS];
  __u32 scale_mode;
};
```

Members

coeff
Noise level Lookup coefficient

scale_mode
dpf noise level scale mode (from *enum rkisp1_cif_isp_dpf_nll_scale_mode*)

struct **rkisp1_cif_isp_dpf_rb_flt**
Red blue filter config

Definition

```
struct rkisp1_cif_isp_dpf_rb_flt {
  __u32 fltsize;
  __u8 spatial_coeff[RKISP1_CIF_ISP_DPF_MAX_SPATIAL_COEFFS];
  __u8 r_enable;
  __u8 b_enable;
};
```

Members

fltsize
The filter size for the red and blue pixels (from *enum rkisp1_cif_isp_dpf_rb_filtersize*)

spatial_coeff
Spatial weights

r_enable
enable filter processing for red pixels

b_enable
enable filter processing for blue pixels

struct **rkisp1_cif_isp_dpf_g_flt**
Green filter Configuration

Definition

```
struct rkisp1_cif_isp_dpf_g_flt {
  __u8 spatial_coeff[RKISP1_CIF_ISP_DPF_MAX_SPATIAL_COEFFS];
  __u8 gr_enable;
  __u8 gb_enable;
};
```

Members

spatial_coeff
Spatial weights

gr_enable
enable filter processing for green pixels in green/red lines

gb_enable
enable filter processing for green pixels in green/blue lines

struct **rkisp1_cif_isp_dpf_gain**
Noise function Configuration

Definition

```
struct rkisp1_cif_isp_dpf_gain {
  __u32 mode;
  __u16 nf_r_gain;
  __u16 nf_b_gain;
  __u16 nf_gr_gain;
  __u16 nf_gb_gain;
};
```

Members

mode
dpf gain usage (from *enum rkisp1_cif_isp_dpf_gain_usage*)

nf_r_gain
Noise function Gain that replaces the AWB gain for red pixels

nf_b_gain
Noise function Gain that replaces the AWB gain for blue pixels

nf_gr_gain
Noise function Gain that replaces the AWB gain for green pixels in a red line

nf_gb_gain
Noise function Gain that replaces the AWB gain for green pixels in a blue line

struct **rkisp1_cif_isp_dpf_config**

Configuration used by De-noising pre-filter

Definition

```
struct rkisp1_cif_isp_dpf_config {
  struct rkisp1_cif_isp_dpf_gain gain;
  struct rkisp1_cif_isp_dpf_g_flt g_flt;
  struct rkisp1_cif_isp_dpf_rb_flt rb_flt;
  struct rkisp1_cif_isp_dpf_nll nll;
};
```

Members

gain
noise function gain

g_flt
green filter config

rb_flt
red blue filter config

nll
noise level lookup

struct **rkisp1_cif_isp_dpf_strength_config**

strength of the filter

Definition

```
struct rkisp1_cif_isp_dpf_strength_config {
  __u8 r;
  __u8 g;
  __u8 b;
};
```

Members

r
filter strength of the RED filter

g
filter strength of the GREEN filter

b
filter strength of the BLUE filter

struct **rkisp1_cif_isp_isp_other_cfg**

Parameters for some blocks in rockchip isp1

Definition

```
struct rkisp1_cif_isp_isp_other_cfg {
  struct rkisp1_cif_isp_dpcc_config dpcc_config;
  struct rkisp1_cif_isp_bls_config bls_config;
  struct rkisp1_cif_isp_sdg_config sdg_config;
  struct rkisp1_cif_isp_lsc_config lsc_config;
  struct rkisp1_cif_isp_awb_gain_config awb_gain_config;
  struct rkisp1_cif_isp_flt_config flt_config;
  struct rkisp1_cif_isp_bdm_config bdm_config;
  struct rkisp1_cif_isp_ctk_config ctk_config;
  struct rkisp1_cif_isp_goc_config goc_config;
  struct rkisp1_cif_isp_dpf_config dpf_config;
  struct rkisp1_cif_isp_dpf_strength_config dpf_strength_config;
  struct rkisp1_cif_isp_cproc_config cproc_config;
  struct rkisp1_cif_isp_ie_config ie_config;
};
```

Members

dpcc_config
Defect Pixel Cluster Correction config

bls_config
black level subtraction config

sdg_config
sensor degamma config

lsc_config
Lens Shade config

awb_gain_config
Auto White balance gain config

flt_config
filter config

bdm_config
demosaic config

ctk_config
cross talk config

goc_config
gamma out config

dpf_config
De-noising pre-filter config

dpf_strength_config
dpf strength config

cproc_config
color process config

ie_config
image effects config

struct **rkisp1_cif_isp_isp_meas_cfg**

Rockchip ISP1 Measure Parameters

Definition

```
struct rkisp1_cif_isp_isp_meas_cfg {
  struct rkisp1_cif_isp_awb_meas_config awb_meas_config;
  struct rkisp1_cif_isp_hst_config hst_config;
  struct rkisp1_cif_isp_aec_config aec_config;
  struct rkisp1_cif_isp_afc_config afc_config;
};
```

Members

awb_meas_config
auto white balance config

hst_config
histogram config

aec_config
auto exposure config

afc_config
auto focus config

struct **rkisp1_params_cfg**

Rockchip ISP1 Input Parameters Meta Data

Definition

```
struct rkisp1_params_cfg {
  __u32 module_en_update;
  __u32 module_ens;
  __u32 module_cfg_update;
  struct rkisp1_cif_isp_isp_meas_cfg meas;
  struct rkisp1_cif_isp_isp_other_cfg others;
};
```

Members

module_en_update
mask the enable bits of which module should be updated

module_ens
mask the enable value of each module, only update the module which correspond bit was set in module_en_update

module_cfg_update
mask the config bits of which module should be updated

meas
measurement config

others
other config

struct **rkisp1_cif_isp_awb_meas**

AWB measured values

Definition

```
struct rkisp1_cif_isp_awb_meas {
  __u32 cnt;
  __u8 mean_y_or_g;
  __u8 mean_cb_or_b;
  __u8 mean_cr_or_r;
};
```

Members

cnt
White pixel count, number of "white pixels" found during last measurement

mean_y_or_g
Mean value of Y within window and frames, Green if RGB is selected.

mean_cb_or_b
Mean value of Cb within window and frames, Blue if RGB is selected.

mean_cr_or_r
Mean value of Cr within window and frames, Red if RGB is selected.

struct **rkisp1_cif_isp_awb_stat**

statistics automatic white balance data

Definition

```
struct rkisp1_cif_isp_awb_stat {
  struct rkisp1_cif_isp_awb_meas awb_mean[RKISP1_CIF_ISP_AWB_MAX_
→GRID];
};
```

Members

awb_mean
Mean measured data

struct **rkisp1_cif_isp_bls_meas_val**

BLS measured values

Definition

```
struct rkisp1_cif_isp_bls_meas_val {
  __u16 meas_r;
  __u16 meas_gr;
  __u16 meas_gb;
  __u16 meas_b;
};
```

Members

meas_r
Mean measured value for Bayer pattern R

meas_gr
Mean measured value for Bayer pattern Gr

meas_gb
Mean measured value for Bayer pattern Gb

meas_b
Mean measured value for Bayer pattern B

struct **rkisp1_cif_isp_ae_stat**
statistics auto exposure data

Definition

```
struct rkisp1_cif_isp_ae_stat {
  __u8 exp_mean[RKISP1_CIF_ISP_AE_MEAN_MAX];
  struct rkisp1_cif_isp_bls_meas_val bls_val;
};
```

Members

exp_mean
Mean luminance value of block xx

bls_val
BLS measured values

Description

Image is divided into 5x5 blocks.

struct **rkisp1_cif_isp_af_meas_val**
AF measured values

Definition

```
struct rkisp1_cif_isp_af_meas_val {
  __u32 sum;
  __u32 lum;
};
```

Members

sum
sharpness value

lum
luminance value

struct **rkisp1_cif_isp_af_stat**
statistics auto focus data

Definition

```
struct rkisp1_cif_isp_af_stat {
  struct rkisp1_cif_isp_af_meas_val window[RKISP1_CIF_ISP_AFM_MAX_
→WINDOWS];
};
```

Members

window
AF measured value of window x

Description

The module measures the sharpness in 3 windows of selectable size via register settings(ISP_AFM_*_A/B/C)

struct **rkisp1_cif_isp_hist_stat**
statistics histogram data

Definition

```
struct rkisp1_cif_isp_hist_stat {
  __u16 hist_bins[RKISP1_CIF_ISP_HIST_BIN_N_MAX];
};
```

Members

hist_bins
measured bin counters

Description

Measurement window divided into 25 sub-windows, set with ISP_HIST_XXX

struct **rkisp1_cif_isp_stat**
Rockchip ISP1 Statistics Data

Definition

```
struct rkisp1_cif_isp_stat {
  struct rkisp1_cif_isp_awb_stat awb;
  struct rkisp1_cif_isp_ae_stat ae;
  struct rkisp1_cif_isp_af_stat af;
  struct rkisp1_cif_isp_hist_stat hist;
};
```

Members

awb
statistics data for automatic white balance

ae
statistics data for auto exposure

af
statistics data for auto focus

hist
statistics histogram data

struct **rkisp1_stat_buffer**
Rockchip ISP1 Statistics Meta Data

Definition

```
struct rkisp1_stat_buffer {
  __u32 meas_type;
  __u32 frame_id;
  struct rkisp1_cif_isp_stat params;
};
```

Members

meas_type
measurement types (RKISP1_CIF_ISP_STAT_* definitions)

frame_id
frame ID for sync

params
statistics data

V4L2_META_FMT_UVC ('UVCH')

UVC Payload Header Data

Description

This format describes standard UVC metadata, extracted from UVC packet headers and provided by the UVC driver through metadata video nodes. That data includes exact copies of the standard part of UVC Payload Header contents and auxiliary timing information, required for precise interpretation of timestamps, contained in those headers. See section "2.4.3.3 Video and Still Image Payload Headers" of the "UVC 1.5 Class specification" for details.

Each UVC payload header can be between 2 and 12 bytes large. Buffers can contain multiple headers, if multiple such headers have been transmitted by the camera for the respective frame. However, the driver may drop headers when the buffer is full, when they contain no useful information (e.g. those without the SCR field or with that field identical to the previous header), or generally to perform rate limiting when the device sends a large number of headers.

Each individual block contains the following fields:

Table 56: UVC Metadata Block

Field	Description
__u64 ts;	system timestamp in host byte order, measured by the driver upon reception of the payload
__u16 sof;	USB Frame Number in host byte order, also obtained by the driver as close as possible to the above timestamp to enable correlation between them
The rest is an exact copy of the UVC payload header:	
__u8 length;	length of the rest of the block, including this field
__u8 flags;	Flags, indicating presence of other standard UVC fields
__u8 buf[];	The rest of the header, possibly including UVC PTS and SCR fields

V4L2_META_FMT_VSP1_HGO ('VSPH')

Renesas R-Car VSP1 1-D Histogram Data

Description

This format describes histogram data generated by the Renesas R-Car VSP1 1-D Histogram (HGO) engine.

The VSP1 HGO is a histogram computation engine that can operate on RGB, YCrCb or HSV data. It operates on a possibly cropped and subsampled input image and computes the minimum, maximum and sum of all pixels as well as per-channel histograms.

The HGO can compute histograms independently per channel, on the maximum of the three channels (RGB data only) or on the Y channel only (YCbCr only). It can additionally output the histogram with 64 or 256 bins, resulting in four possible modes of operation.

- In *64 bins normal mode*, the HGO operates on the three channels independently to compute three 64-bins histograms. RGB, YCbCr and HSV image formats are supported.
- In *64 bins maximum mode*, the HGO operates on the maximum of the (R, G, B) channels to compute a single 64-bins histogram. Only the RGB image format is supported.
- In *256 bins normal mode*, the HGO operates on the Y channel to compute a single 256-bins histogram. Only the YCbCr image format is supported.
- In *256 bins maximum mode*, the HGO operates on the maximum of the (R, G, B) channels to compute a single 256-bins histogram. Only the RGB image format is supported.

Byte Order. All data is stored in memory in little endian format. Each cell in the tables contains one byte.

Table 57: VSP1 HGO Data - 64 Bins, Normal Mode (792 bytes)

Offset	Memory [31:24]	[23:16]	[15:8]	[7:0]
0		R/Cr/H max [7:0]		R/Cr/H min [7:0]
4		G/Y/S max [7:0]		G/Y/S min [7:0]
8		B/Cb/V max [7:0]		B/Cb/V min [7:0]
12	R/Cr/H sum [31:0]			
16	G/Y/S sum [31:0]			
20	B/Cb/V sum [31:0]			
24	R/Cr/H bin 0 [31:0]			
	…			
276	R/Cr/H bin 63 [31:0]			
280	G/Y/S bin 0 [31:0]			
	…			
532	G/Y/S bin 63 [31:0]			
536	B/Cb/V bin 0 [31:0]			
	…			
788	B/Cb/V bin 63 [31:0]			

Table 58: VSP1 HGO Data - 64 Bins, Max Mode (264 bytes)

Offset	Memory [31:24]	[23:16]	[15:8]	[7:0]
0		max(R,G,B) max [7:0]		max(R,G,B) min [7:0]
4	max(R,G,B) sum [31:0]			
8	max(R,G,B) bin 0 [31:0]			
	…			
260	max(R,G,B) bin 63 [31:0]			

Table 59: VSP1 HGO Data - 256 Bins, Normal Mode (1032 bytes)

Offset	Memory [31:24]	[23:16]	[15:8]	[7:0]
0		Y max [7:0]		Y min [7:0]
4	Y sum [31:0]			
8	Y bin 0 [31:0]			
	…			
1028	Y bin 255 [31:0]			

Table 60: VSP1 HGO Data - 256 Bins, Max Mode (1032 bytes)

Offset	Memory [31:24]	[23:16]	[15:8]	[7:0]
0		max(R,G,B) max [7:0]		max(R,G,B) min [7:0]
4	max(R,G,B) sum [31:0]			
8	max(R,G,B) bin 0 [31:0]			
	...			
1028	max(R,G,B) bin 255 [31:0]			

V4L2_META_FMT_VSP1_HGT ('VSPT')

Renesas R-Car VSP1 2-D Histogram Data

Description

This format describes histogram data generated by the Renesas R-Car VSP1 2-D Histogram (HGT) engine.

The VSP1 HGT is a histogram computation engine that operates on HSV data. It operates on a possibly cropped and subsampled input image and computes the sum, maximum and minimum of the S component as well as a weighted frequency histogram based on the H and S components.

The histogram is a matrix of 6 Hue and 32 Saturation buckets, 192 in total. Each HSV value is added to one or more buckets with a weight between 1 and 16 depending on the Hue areas configuration. Finding the corresponding buckets is done by inspecting the H and S value independently.

The Saturation position **n** (0 - 31) of the bucket in the matrix is found by the expression:

n = S / 8

The Hue position **m** (0 - 5) of the bucket in the matrix depends on how the HGT Hue areas are configured. There are 6 user configurable Hue Areas which can be configured to cover overlapping Hue values:

```
      Area 0       Area 1       Area 2       Area 3       Area 4       ␣
↪Area 5
     ________     ________     ________     ________     ________     ____
↪____
\   /|      |\   /|      |\   /|      |\   /|      |\   /|      |\   /|  ␣
↪   |\   /
 \ / |      | \ / |      | \ / |      | \ / |      | \ / |      | \ / |  ␣
↪   | \ /
  X  |      |  X  |      |  X  |      |  X  |      |  X  |      |  X  |  ␣
↪   |  X
 / \ |      | / \ |      | / \ |      | / \ |      | / \ |      | / \ |  ␣
↪   | / \
/   \|      |/   \|      |/   \|      |/   \|      |/   \|      |/   \|  ␣
```

(continues on next page)

(continued from previous page)

```
↪    |/   \
5U   0L      0U   1L      1U   2L      2U   3L      3U   4L      4U   5L  ␣
↪    5U   0L
      <0............................Hue Value............................
↪255>
```

When two consecutive areas don' t overlap (n+1L is equal to nU) the boundary value is considered as part of the lower area.

Pixels with a hue value included in the centre of an area (between nL and nU included) are attributed to that single area and given a weight of 16. Pixels with a hue value included in the overlapping region between two areas (between n+1L and nU excluded) are attributed to both areas and given a weight for each of these areas proportional to their position along the diagonal lines (rounded down).

The Hue area setup must match one of the following constrains:

```
0L <= 0U <= 1L <= 1U <= 2L <= 2U <= 3L <= 3U <= 4L <= 4U <= 5L <= 5U
```

```
0U <= 1L <= 1U <= 2L <= 2U <= 3L <= 3U <= 4L <= 4U <= 5L <= 5U <= 0L
```

Byte Order. All data is stored in memory in little endian format. Each cell in the tables contains one byte.

Table 61: VSP1 HGT Data - (776 bytes)

Offset	Memory [31:24]	[23:16]	[15:8]	[7:0]
0	•	S max [7:0]	•	S min [7:0]
4	S sum [31:0]			
8	Histogram bucket (m=0, n=0) [31:0]			
12	Histogram bucket (m=0, n=1) [31:0]			
	…			
132	Histogram bucket (m=0, n=31) [31:0]			
136	Histogram bucket (m=1, n=0) [31:0]			
	…			
264	Histogram bucket (m=2, n=0) [31:0]			
	…			
392	Histogram bucket (m=3, n=0) [31:0]			
	…			
520	Histogram bucket (m=4, n=0) [31:0]			
	…			
648	Histogram bucket (m=5, n=0) [31:0]			
	…			
772	Histogram bucket (m=5, n=31) [31:0]			

V4L2_META_FMT_VIVID ('VIVD')

VIVID Metadata Format

Description

This describes metadata format used by the vivid driver.

It sets Brightness, Saturation, Contrast and Hue, each of which maps to corresponding controls of the vivid driver with respect to the range and default values.

It contains the following fields:

Table 62: VIVID Metadata

Field	Description
u16 brightness;	Image brightness, the value is in the range 0 to 255, with the default value as 128.
u16 contrast;	Image contrast, the value is in the range 0 to 255, with the default value as 128.
u16 saturation;	Image color saturation, the value is in the range 0 to 255, with the default value as 128.
s16 hue;	Image color balance, the value is in the range -128 to 128, with the default value as 0.

Reserved Format Identifiers

These formats are not defined by this specification, they are just listed for reference and to avoid naming conflicts. If you want to register your own format, send an e-mail to the linux-media mailing list https://linuxtv.org/lists.php for inclusion in the `videodev2.h` file. If you want to share your format with other developers add a link to your documentation and send a copy to the linux-media mailing list for inclusion in this section. If you think your format should be listed in a standard format section please make a proposal on the linux-media mailing list.

Table 63: Reserved Image Formats

Identifier	Code	Details
`V4L2_PIX_FMT_DV`	'dvsd'	unknown
`V4L2_PIX_FMT_ET61X251`	'E625'	Compressed format of the ET61X251 driver
`V4L2_PIX_FMT_HI240`	'HI24'	8 bit RGB format used by the BTTV driver.
`V4L2_PIX_FMT_HM12`	'HM12'	YUV 4:2:0 format used by the IVTV driver. The format is documented in the kernel sources in the file `Documentation/userspace-api/media/drivers/cx2341x-uapi.rst`

continues on next page

Table 63 – continued from previous page

Identifier	Code	Details
V4L2_PIX_FMT_CPIA1	'CPIA'	YUV format used by the gspca cpia1 driver.
V4L2_PIX_FMT_JPGL	'JPGL'	JPEG-Light format (Pegasus Lossless JPEG) used in Divio webcams NW 80x.
V4L2_PIX_FMT_SPCA501	'S501'	YUYV per line used by the gspca driver.
V4L2_PIX_FMT_SPCA505	'S505'	YYUV per line used by the gspca driver.
V4L2_PIX_FMT_SPCA508	'S508'	YUVY per line used by the gspca driver.
V4L2_PIX_FMT_SPCA561	'S561'	Compressed GBRG Bayer format used by the gspca driver.
V4L2_PIX_FMT_PAC207	'P207'	Compressed BGGR Bayer format used by the gspca driver.
V4L2_PIX_FMT_MR97310A	'M310'	Compressed BGGR Bayer format used by the gspca driver.
V4L2_PIX_FMT_JL2005BCD	'JL20'	JPEG compressed RGGB Bayer format used by the gspca driver.
V4L2_PIX_FMT_OV511	'O511'	OV511 JPEG format used by the gspca driver.
V4L2_PIX_FMT_OV518	'O518'	OV518 JPEG format used by the gspca driver.
V4L2_PIX_FMT_PJPG	'PJPG'	Pixart 73xx JPEG format used by the gspca driver.
V4L2_PIX_FMT_SE401	'S401'	Compressed RGB format used by the gspca se401 driver
V4L2_PIX_FMT_SQ905C	'905C'	Compressed RGGB bayer format used by the gspca driver.
V4L2_PIX_FMT_MJPEG	'MJPG'	Compressed format used by the Zoran driver
V4L2_PIX_FMT_PWC1	'PWC1'	Compressed format of the PWC driver.
V4L2_PIX_FMT_PWC2	'PWC2'	Compressed format of the PWC driver.
V4L2_PIX_FMT_SN9C10X	'S910'	Compressed format of the SN9C102 driver.
V4L2_PIX_FMT_SN9C20X_I420	'S920'	YUV 4:2:0 format of the gspca sn9c20x driver.
V4L2_PIX_FMT_SN9C2028	'SONX'	Compressed GBRG bayer format of the gspca sn9c2028 driver.
V4L2_PIX_FMT_STV0680	'S680'	Bayer format of the gspca stv0680 driver.

continues on next page

Table 63 – continued from previous page

Identifier	Code	Details
V4L2_PIX_FMT_WNVA	'WNVA'	Used by the Winnov Videum driver, http:/ www.thedirks.org/winnov/
V4L2_PIX_FMT_TM6000	'TM60'	Used by Trident tm6000
V4L2_PIX_FMT_CIT_YYVYUY	'CITV'	Used by xirlink CIT, found at IBM webcams. Uses one line of Y then 1 line of VYUY
V4L2_PIX_FMT_KONICA420	'KONI'	Used by Konica webcams. YUV420 planar in blocks of 256 pixels.
V4L2_PIX_FMT_YYUV	'YYUV'	unknown
V4L2_PIX_FMT_Y4	'Y04 '	Old 4-bit greyscale format. Only the mos significant 4 bits of each byte are used, the other bits are set to 0.
V4L2_PIX_FMT_Y6	'Y06 '	Old 6-bit greyscale format. Only the mos significant 6 bits of each byte are used, the other bits are set to 0.

continues on next page

Table 63 – continued from previous page

Identifier	Code	Details
V4L2_PIX_FMT_S5C_UYVY_JPG	'S5CI'	Two-planar format used by Samsung S5C73MX cameras. The first plane contains interleaved JPEG and UYVY image data, followed by meta data in form of an array of offsets to the UYVY data blocks. The actual pointer array follows immediately the interleaved JPEG/UYVY data, the number of entries in this array equals the height of the UYVY image. Each entry is a 4-byte unsigned integer in big endian order and it's an offset to a single pixel line of the UYVY image. The first plane can start either with JPEG or UYVY data chunk. The size of a single UYVY block equals the UYVY image s width multiplied by 2. The size of a JPEG chunk depends on the image and can vary with each line. The second plane, at an offset of 4084 bytes contains a 4-byte offset to the pointer array in the first plane. This offset is followed by a 4-byte value indicating size of the pointer array. All numbers in the second plane are also in big endian order. Remaining data in the second plane is undefined. The information in the second plane allows to easily find location of the pointer array, which can be different for each frame. The size of the pointer array is constant for given UYVY image height. In order to extract UYVY and JPEG frames an application can initially set a data pointer to the start of first plane and then add an offset from the first entry of the pointers table. Such a pointer indicates start of an UYVY image pixel line. Whole UYVY line can be copied to a separate buffer. These steps should be repeated for each line, i.e. the number of entries in the pointer array. Anything what's in between the UYVY lines is JPEG data and should be concatenated to form the JPEG stream.

continues on next page

Table 63 – continued from previous page

Identifier	Code	Details
V4L2_PIX_FMT_MT21C	'MT21'	Compressed two-planar YVU420 forma used by Mediatek MT8173. The compres sion is lossless. It is an opaque interme diate format and the MDP hardware mus be used to convert V4L2_PIX_FMT_MT21C to V4L2_PIX_FMT_NV12M V4L2_PIX_FMT_YUV420M or V4L2_PIX_FMT_YVU420.
V4L2_PIX_FMT_SUNXI_TILED_NV12	'ST12'	Two-planar NV12-based format used by the video engine found on Allwinner (codenamed sunxi) platforms, with 32x32 tiles for the luminance plane and 32x64 tiles for the chrominance plane. The data in each tile is stored in linear order, within the tile bounds Each tile follows the previous one linearly in memory (from left to right, top to bottom). The associated buffer dimensions are aligned to match an integer number of tiles, result ing in 32-aligned resolutions for the lumi nance plane and 16-aligned resolutions for the chrominance plane (with 2x2 subsam pling).

Colorspaces

'Color' is a very complex concept and depends on physics, chemistry and biology. Just because you have three numbers that describe the 'red', 'green' and 'blue' components of the color of a pixel does not mean that you can accurately display that color. A colorspace defines what it actually *means* to have an RGB value of e.g. (255, 0, 0). That is, which color should be reproduced on the screen in a perfectly calibrated environment.

In order to do that we first need to have a good definition of color, i.e. some way to uniquely and unambiguously define a color so that someone else can reproduce it. Human color vision is trichromatic since the human eye has color receptors that are sensitive to three different wavelengths of light. Hence the need to use three numbers to describe color. Be glad you are not a mantis shrimp as those are sensitive to 12 different wavelengths, so instead of RGB we would be using the ABCDEFGHIJKL colorspace…

Color exists only in the eye and brain and is the result of how strongly color receptors are stimulated. This is based on the Spectral Power Distribution (SPD) which is a graph showing the intensity (radiant power) of the light at wavelengths covering the visible spectrum as it enters the eye. The science of colorimetry is about the relationship between the SPD and color as perceived by the human brain.

Since the human eye has only three color receptors it is perfectly possible that different SPDs will result in the same stimulation of those receptors and are perceived as the same color, even though the SPD of the light is different.

In the 1920s experiments were devised to determine the relationship between SPDs and the perceived color and that resulted in the CIE 1931 standard that defines spectral weighting functions that model the perception of color. Specifically that standard defines functions that can take an SPD and calculate the stimulus for each color receptor. After some further mathematical transforms these stimuli are known as the *CIE XYZ tristimulus* values and these X, Y and Z values describe a color as perceived by a human unambiguously. These X, Y and Z values are all in the range [0…1].

The Y value in the CIE XYZ colorspace corresponds to luminance. Often the CIE XYZ colorspace is transformed to the normalized CIE xyY colorspace:

x = X / (X + Y + Z)

y = Y / (X + Y + Z)

The x and y values are the chromaticity coordinates and can be used to define a color without the luminance component Y. It is very confusing to have such similar names for these colorspaces. Just be aware that if colors are specified with lower case 'x' and 'y', then the CIE xyY colorspace is used. Upper case 'X' and 'Y' refer to the CIE XYZ colorspace. Also, y has nothing to do with luminance. Together x and y specify a color, and Y the luminance. That is really all you need to remember from a practical point of view. At the end of this section you will find reading resources that go into much more detail if you are interested.

A monitor or TV will reproduce colors by emitting light at three different wavelengths, the combination of which will stimulate the color receptors in the eye and thus cause the perception of color. Historically these wavelengths were defined by the red, green and blue phosphors used in the displays. These *color primaries* are part of what defines a colorspace.

Different display devices will have different primaries and some primaries are more suitable for some display technologies than others. This has resulted in a variety of colorspaces that are used for different display technologies or uses. To define a colorspace you need to define the three color primaries (these are typically defined as x, y chromaticity coordinates from the CIE xyY colorspace) but also the white reference: that is the color obtained when all three primaries are at maximum power. This determines the relative power or energy of the primaries. This is usually chosen to be close to daylight which has been defined as the CIE D65 Illuminant.

To recapitulate: the CIE XYZ colorspace uniquely identifies colors. Other colorspaces are defined by three chromaticity coordinates defined in the CIE xyY colorspace. Based on those a 3x3 matrix can be constructed that transforms CIE XYZ colors to colors in the new colorspace.

Both the CIE XYZ and the RGB colorspace that are derived from the specific chromaticity primaries are linear colorspaces. But neither the eye, nor display technology is linear. Doubling the values of all components in the linear colorspace will not be perceived as twice the intensity of the color. So each colorspace also defines a transfer function that takes a linear color component value and transforms it to the non-linear component value, which is a closer match to the non-linear performance of both the eye and displays. Linear component values are denoted RGB, non-linear are denoted as R' G' B'. In general colors used in graphics are all R' G' B', except in openGL which uses linear RGB. Special care should be taken when dealing with openGL to provide linear RGB colors or to use the built-in openGL support to apply the inverse transfer function.

The final piece that defines a colorspace is a function that transforms non-linear R' G' B' to non-linear Y' CbCr. This function is determined by the so-called luma coefficients. There may be multiple possible Y' CbCr encodings allowed for the same colorspace. Many encodings of color prefer to use luma (Y') and chroma (CbCr) instead of R' G' B'. Since the human eye is more sensitive to differences in luminance than in color this encoding allows one to reduce the amount of color information compared to the luma data. Note that the luma (Y') is unrelated to the Y in the CIE XYZ colorspace. Also note that Y' CbCr is often called YCbCr or YUV even though these are strictly speaking wrong.

Sometimes people confuse Y' CbCr as being a colorspace. This is not correct, it is just an encoding of an R' G' B' color into luma and chroma values. The underlying colorspace that is associated with the R' G' B' color is also associated with the Y' CbCr color.

The final step is how the RGB, R' G' B' or Y' CbCr values are quantized. The CIE XYZ colorspace where X, Y and Z are in the range [0…1] describes all colors that humans can perceive, but the transform to another colorspace will produce colors that are outside the [0…1] range. Once clamped to the [0…1] range those colors can no longer be reproduced in that colorspace. This clamping is what reduces the extent or gamut of the colorspace. How the range of [0…1] is translated to integer values in the range of [0…255] (or higher, depending on the color depth) is called the quantization. This is *not* part of the colorspace definition. In practice RGB or R' G' B' values are full range, i.e. they use the full [0…255] range. Y' CbCr values on the other hand are limited range with Y' using [16…235] and Cb and Cr using [16…240].

Unfortunately, in some cases limited range RGB is also used where the components use the range [16…235]. And full range Y' CbCr also exists using the [0…255] range.

In order to correctly interpret a color you need to know the quantization range, whether it is R' G' B' or Y' CbCr, the used Y' CbCr encoding and the colorspace. From that information you can calculate the corresponding CIE XYZ color and map that again to whatever colorspace your display device uses.

The colorspace definition itself consists of the three chromaticity primaries, the white reference chromaticity, a transfer function and the luma coefficients needed to transform R' G' B' to Y' CbCr. While some colorspace standards correctly define all four, quite often the colorspace standard only defines some, and you have to rely on other standards for the missing pieces. The fact that colorspaces are often a mix of different standards also led to very confusing naming conventions where the name of a standard was used to name a colorspace when in fact that standard was part of various other colorspaces as well.

If you want to read more about colors and colorspaces, then the following resources are useful: *poynton* is a good practical book for video engineers, *colimg* has a much broader scope and describes many more aspects of color (physics, chemistry, biology, etc.). The http://www.brucelindbloom.com website is an excellent resource, especially with respect to the mathematics behind colorspace conversions. The wikipedia CIE 1931 colorspace article is also very useful.

Defining Colorspaces in V4L2

In V4L2 colorspaces are defined by four values. The first is the colorspace identifier (enum *v4l2_colorspace*) which defines the chromaticities, the default transfer function, the default Y'CbCr encoding and the default quantization method. The second is the transfer function identifier (enum *v4l2_xfer_func*) to specify non-standard transfer functions. The third is the Y'CbCr encoding identifier (enum *v4l2_ycbcr_encoding*) to specify non-standard Y'CbCr encodings and the fourth is the quantization identifier (enum *v4l2_quantization*) to specify non-standard quantization methods. Most of the time only the colorspace field of struct *v4l2_pix_format* or struct *v4l2_pix_format_mplane* needs to be filled in.

On *HSV formats* the *Hue* is defined as the angle on the cylindrical color representation. Usually this angle is measured in degrees, i.e. 0-360. When we map this angle value into 8 bits, there are two basic ways to do it: Divide the angular value by 2 (0-179), or use the whole range, 0-255, dividing the angular value by 1.41. The enum *v4l2_hsv_encoding* specifies which encoding is used.

Note: The default R'G'B' quantization is full range for all colorspaces. HSV formats are always full range.

type **v4l2_colorspace**

Table 64: V4L2 Colorspaces

Identifier	Details
V4L2_COLORSPACE_DEFAULT	The default colorspace. This can be used by applications to let the driver fill in the colorspace.
V4L2_COLORSPACE_SMPTE170M	See *Colorspace SMPTE 170M (V4L2_COLORSPACE_SMPTE170M)*.
V4L2_COLORSPACE_REC709	See *Colorspace Rec. 709 (V4L2_COLORSPACE_REC709)*.
V4L2_COLORSPACE_SRGB	See *Colorspace sRGB (V4L2_COLORSPACE_SRGB)*.
V4L2_COLORSPACE_OPRGB	See *Colorspace opRGB (V4L2_COLORSPACE_OPRGB)*.
V4L2_COLORSPACE_BT2020	See *Colorspace BT.2020 (V4L2_COLORSPACE_BT2020)*.
V4L2_COLORSPACE_DCI_P3	See *Colorspace DCI-P3 (V4L2_COLORSPACE_DCI_P3)*.
V4L2_COLORSPACE_SMPTE240M	See *Colorspace SMPTE 240M (V4L2_COLORSPACE_SMPTE240M)*.
V4L2_COLORSPACE_470_SYSTEM_M	See *Colorspace NTSC 1953 (V4L2_COLORSPACE_470_SYSTEM_M)*.
V4L2_COLORSPACE_470_SYSTEM_BG	See *Colorspace EBU Tech. 3213 (V4L2_COLORSPACE_470_SYSTEM_BG)*.
V4L2_COLORSPACE_JPEG	See *Colorspace JPEG (V4L2_COLORSPACE_JPEG)*.
V4L2_COLORSPACE_RAW	The raw colorspace. This is used for raw image capture where the image is minimally processed and is using the internal colorspace of the device. The software that processes an image using this 'colorspace' will have to know the internals of the capture device.

type **v4l2_xfer_func**

Table 65: V4L2 Transfer Function

Identifier	Details
V4L2_XFER_FUNC_DEFAULT	Use the default transfer function as defined by the colorspace.
V4L2_XFER_FUNC_709	Use the Rec. 709 transfer function.
V4L2_XFER_FUNC_SRGB	Use the sRGB transfer function.
V4L2_XFER_FUNC_OPRGB	Use the opRGB transfer function.
V4L2_XFER_FUNC_SMPTE240M	Use the SMPTE 240M transfer function.
V4L2_XFER_FUNC_NONE	Do not use a transfer function (i.e. use linear RGB values).
V4L2_XFER_FUNC_DCI_P3	Use the DCI-P3 transfer function.
V4L2_XFER_FUNC_SMPTE2084	Use the SMPTE 2084 transfer function. See *Transfer Function SMPTE 2084 (V4L2_XFER_FUNC_SMPTE2084)*.

type **v4l2_ycbcr_encoding**

Table 66: V4L2 Y'CbCr Encodings

Identifier	Details
V4L2_YCBCR_ENC_DEFAULT	Use the default Y'CbCr encoding as defined by the colorspace.
V4L2_YCBCR_ENC_601	Use the BT.601 Y'CbCr encoding.
V4L2_YCBCR_ENC_709	Use the Rec. 709 Y'CbCr encoding.
V4L2_YCBCR_ENC_XV601	Use the extended gamut xvYCC BT.601 encoding.
V4L2_YCBCR_ENC_XV709	Use the extended gamut xvYCC Rec. 709 encoding.
V4L2_YCBCR_ENC_BT2020	Use the default non-constant luminance BT.2020 Y'CbCr encoding.
V4L2_YCBCR_ENC_BT2020_CONST_LUM	Use the constant luminance BT.2020 Yc'CbcCrc encoding.
V4L2_YCBCR_ENC_SMPTE_240M	Use the SMPTE 240M Y'CbCr encoding.

type **v4l2_hsv_encoding**

Table 67: V4L2 HSV Encodings

Identifier	Details
V4L2_HSV_ENC_180	For the Hue, each LSB is two degrees.
V4L2_HSV_ENC_256	For the Hue, the 360 degrees are mapped into 8 bits, i.e. each LSB is roughly 1.41 degrees.

type **v4l2_quantization**

Table 68: V4L2 Quantization Methods

Identifier	Details
V4L2_QUANTIZATION_DEFAULT	Use the default quantization encoding as defined by the colorspace. This is always full range for R' G' B' and HSV. It is usually limited range for Y' CbCr.
V4L2_QUANTIZATION_FULL_RANGE	Use the full range quantization encoding. I.e. the range [0…1] is mapped to [0…255] (with possible clipping to [1…254] to avoid the 0x00 and 0xff values). Cb and Cr are mapped from [-0.5…0.5] to [0…255] (with possible clipping to [1…254] to avoid the 0x00 and 0xff values).
V4L2_QUANTIZATION_LIM_RANGE	Use the limited range quantization encoding. I.e. the range [0…1] is mapped to [16…235]. Cb and Cr are mapped from [-0.5…0.5] to [16…240]. Limited Range cannot be used with HSV.

Detailed Colorspace Descriptions

Colorspace SMPTE 170M (V4L2_COLORSPACE_SMPTE170M)

The *SMPTE 170M* standard defines the colorspace used by NTSC and PAL and by SDTV in general. The default transfer function is `V4L2_XFER_FUNC_709`. The default Y'CbCr encoding is `V4L2_YCBCR_ENC_601`. The default Y'CbCr quantization is limited range. The chromaticities of the primary colors and the white reference are:

Table 69: SMPTE 170M Chromaticities

Color	x	y
Red	0.630	0.340
Green	0.310	0.595
Blue	0.155	0.070
White Reference (D65)	0.3127	0.3290

The red, green and blue chromaticities are also often referred to as the SMPTE C set, so this colorspace is sometimes called SMPTE C as well.

The transfer function defined for SMPTE 170M is the same as the one defined in Rec. 709.

$$L' = -1.099(-L)^{0.45} + 0.099, \text{ for } L \leq -0.018$$
$$L' = 4.5L, \text{ for } -0.018 < L < 0.018$$
$$L' = 1.099L^{0.45} - 0.099, \text{ for } L \geq 0.018$$

Inverse Transfer function:

$$L = -\left(\frac{L' - 0.099}{-1.099}\right)^{\frac{1}{0.45}}, \text{ for } L' \le -0.081$$
$$L = \frac{L'}{4.5}, \text{ for } -0.081 < L' < 0.081$$
$$L = \left(\frac{L' + 0.099}{1.099}\right)^{\frac{1}{0.45}}, \text{ for } L' \ge 0.081$$

The luminance (Y') and color difference (Cb and Cr) are obtained with the following `V4L2_YCBCR_ENC_601` encoding:

$$Y' = 0.2990R' + 0.5870G' + 0.1140B'$$
$$Cb = -0.1687R' - 0.3313G' + 0.5B'$$
$$Cr = 0.5R' - 0.4187G' - 0.0813B'$$

Y' is clamped to the range [0…1] and Cb and Cr are clamped to the range [-0.5…0.5]. This conversion to Y' CbCr is identical to the one defined in the *ITU BT.601* standard and this colorspace is sometimes called BT.601 as well, even though BT.601 does not mention any color primaries.

The default quantization is limited range, but full range is possible although rarely seen.

Colorspace Rec. 709 (V4L2_COLORSPACE_REC709)

The *ITU BT.709* standard defines the colorspace used by HDTV in general. The default transfer function is `V4L2_XFER_FUNC_709`. The default Y' CbCr encoding is `V4L2_YCBCR_ENC_709`. The default Y' CbCr quantization is limited range. The chromaticities of the primary colors and the white reference are:

Table 70: Rec. 709 Chromaticities

Color	x	y
Red	0.640	0.330
Green	0.300	0.600
Blue	0.150	0.060
White Reference (D65)	0.3127	0.3290

The full name of this standard is Rec. ITU-R BT.709-5.

Transfer function. Normally L is in the range [0…1], but for the extended gamut xvYCC encoding values outside that range are allowed.

$$L' = -1.099(-L)^{0.45} + 0.099, \text{ for } L \le -0.018$$
$$L' = 4.5L, \text{ for } -0.018 < L < 0.018$$
$$L' = 1.099L^{0.45} - 0.099, \text{ for } L \ge 0.018$$

Inverse Transfer function:

$$L = -\left(\frac{L' - 0.099}{-1.099}\right)^{\frac{1}{0.45}}, \text{ for } L' \le -0.081$$
$$L = \frac{L'}{4.5}, \text{ for } -0.081 < L' < 0.081$$
$$L = \left(\frac{L' + 0.099}{1.099}\right)^{\frac{1}{0.45}}, \text{ for } L' \ge 0.081$$

The luminance (Y') and color difference (Cb and Cr) are obtained with the following `V4L2_YCBCR_ENC_709` encoding:

$$Y' = 0.2126R' + 0.7152G' + 0.0722B'$$
$$Cb = -0.1146R' - 0.3854G' + 0.5B'$$
$$Cr = 0.5R' - 0.4542G' - 0.0458B'$$

Y' is clamped to the range [0…1] and Cb and Cr are clamped to the range [-0.5…0.5].

The default quantization is limited range, but full range is possible although rarely seen.

The `V4L2_YCBCR_ENC_709` encoding described above is the default for this colorspace, but it can be overridden with `V4L2_YCBCR_ENC_601`, in which case the BT.601 Y' CbCr encoding is used.

Two additional extended gamut Y' CbCr encodings are also possible with this colorspace:

The xvYCC 709 encoding (`V4L2_YCBCR_ENC_XV709`, *xvYCC*) is similar to the Rec. 709 encoding, but it allows for R', G' and B' values that are outside the range [0…1]. The resulting Y', Cb and Cr values are scaled and offset according to the limited range formula:

$$Y' = \frac{219}{256} * (0.2126R' + 0.7152G' + 0.0722B') + \frac{16}{256}$$
$$Cb = \frac{224}{256} * (-0.1146R' - 0.3854G' + 0.5B')$$
$$Cr = \frac{224}{256} * (0.5R' - 0.4542G' - 0.0458B')$$

The xvYCC 601 encoding (`V4L2_YCBCR_ENC_XV601`, *xvYCC*) is similar to the BT.601 encoding, but it allows for R', G' and B' values that are outside the range [0…1]. The resulting Y', Cb and Cr values are scaled and offset according to the limited range formula:

$$Y' = \frac{219}{256} * (0.2990R' + 0.5870G' + 0.1140B') + \frac{16}{256}$$
$$Cb = \frac{224}{256} * (-0.1687R' - 0.3313G' + 0.5B')$$
$$Cr = \frac{224}{256} * (0.5R' - 0.4187G' - 0.0813B')$$

Y' is clamped to the range [0…1] and Cb and Cr are clamped to the range [-0.5…0.5] and quantized without further scaling or offsets. The non-standard xvYCC

709 or xvYCC 601 encodings can be used by selecting V4L2_YCBCR_ENC_XV709 or V4L2_YCBCR_ENC_XV601. As seen by the xvYCC formulas these encodings always use limited range quantization, there is no full range variant. The whole point of these extended gamut encodings is that values outside the limited range are still valid, although they map to R', G' and B' values outside the [0⋯1] range and are therefore outside the Rec. 709 colorspace gamut.

Colorspace sRGB (V4L2_COLORSPACE_SRGB)

The *sRGB* standard defines the colorspace used by most webcams and computer graphics. The default transfer function is V4L2_XFER_FUNC_SRGB. The default Y'CbCr encoding is V4L2_YCBCR_ENC_601. The default Y'CbCr quantization is limited range.

Note that the *sYCC* standard specifies full range quantization, however all current capture hardware supported by the kernel convert R'G'B' to limited range Y'CbCr. So choosing full range as the default would break how applications interpret the quantization range.

The chromaticities of the primary colors and the white reference are:

Table 71: sRGB Chromaticities

Color	x	y
Red	0.640	0.330
Green	0.300	0.600
Blue	0.150	0.060
White Reference (D65)	0.3127	0.3290

These chromaticities are identical to the Rec. 709 colorspace.

Transfer function. Note that negative values for L are only used by the Y' CbCr conversion.

$$L' = -1.055(-L)^{\frac{1}{2.4}} + 0.055, \text{ for } L < -0.0031308$$
$$L' = 12.92L, \text{ for } -0.0031308 \le L \le 0.0031308$$
$$L' = 1.055L^{\frac{1}{2.4}} - 0.055, \text{ for } 0.0031308 < L \le 1$$

Inverse Transfer function:

$$L = -((-L' + 0.055)/1.055)^{2.4}, \text{ for } L' < -0.04045$$
$$L = L'/12.92, \text{ for } -0.04045 \le L' \le 0.04045$$
$$L = ((L' + 0.055)/1.055)^{2.4}, \text{ for } L' > 0.04045$$

The luminance (Y') and color difference (Cb and Cr) are obtained with the following V4L2_YCBCR_ENC_601 encoding as defined by *sYCC*:

$$Y' = 0.2990R' + 0.5870G' + 0.1140B'$$
$$Cb = -0.1687R' - 0.3313G' + 0.5B'$$
$$Cr = 0.5R' - 0.4187G' - 0.0813B'$$

Y' is clamped to the range [0…1] and Cb and Cr are clamped to the range [-0.5 …0.5]. This transform is identical to one defined in SMPTE 170M/BT.601. The Y' CbCr quantization is limited range.

Colorspace opRGB (V4L2_COLORSPACE_OPRGB)

The *opRGB* standard defines the colorspace used by computer graphics that use the opRGB colorspace. The default transfer function is `V4L2_XFER_FUNC_OPRGB`. The default Y' CbCr encoding is `V4L2_YCBCR_ENC_601`. The default Y' CbCr quantization is limited range.

Note that the *opRGB* standard specifies full range quantization, however all current capture hardware supported by the kernel convert R' G' B' to limited range Y' CbCr. So choosing full range as the default would break how applications interpret the quantization range.

The chromaticities of the primary colors and the white reference are:

Table 72: opRGB Chromaticities

Color	x	y
Red	0.6400	0.3300
Green	0.2100	0.7100
Blue	0.1500	0.0600
White Reference (D65)	0.3127	0.3290

Transfer function:

$$L' = L^{\frac{1}{2.19921875}}$$

Inverse Transfer function:

$$L = L'^{(2.19921875)}$$

The luminance (Y') and color difference (Cb and Cr) are obtained with the following `V4L2_YCBCR_ENC_601` encoding:

$$Y' = 0.2990R' + 0.5870G' + 0.1140B'$$
$$Cb = -0.1687R' - 0.3313G' + 0.5B'$$
$$Cr = 0.5R' - 0.4187G' - 0.0813B'$$

Y' is clamped to the range [0…1] and Cb and Cr are clamped to the range [-0.5 …0.5]. This transform is identical to one defined in SMPTE 170M/BT.601. The Y' CbCr quantization is limited range.

Colorspace BT.2020 (V4L2_COLORSPACE_BT2020)

The *ITU BT.2020* standard defines the colorspace used by Ultra-high definition television (UHDTV). The default transfer function is `V4L2_XFER_FUNC_709`. The default Y'CbCr encoding is `V4L2_YCBCR_ENC_BT2020`. The default Y'CbCr quantization is limited range. The chromaticities of the primary colors and the white reference are:

Table 73: BT.2020 Chromaticities

Color	x	y
Red	0.708	0.292
Green	0.170	0.797
Blue	0.131	0.046
White Reference (D65)	0.3127	0.3290

Transfer function (same as Rec. 709):

$$L' = 4.5L, \text{ for } 0 \le L < 0.018$$
$$L' = 1.099L^{0.45} - 0.099, \text{ for } 0.018 \le L \le 1$$

Inverse Transfer function:

$$L = L'/4.5, \text{ for } L' < 0.081$$
$$L = \left(\frac{L' + 0.099}{1.099}\right)^{\frac{1}{0.45}}, \text{ for } L' \ge 0.081$$

Please note that while Rec. 709 is defined as the default transfer function by the *ITU BT.2020* standard, in practice this colorspace is often used with the *Transfer Function SMPTE 2084 (V4L2_XFER_FUNC_SMPTE2084)*. In particular Ultra HD Blu-ray discs use this combination.

The luminance (Y') and color difference (Cb and Cr) are obtained with the following `V4L2_YCBCR_ENC_BT2020` encoding:

$$Y' = 0.2627R' + 0.6780G' + 0.0593B'$$
$$Cb = -0.1396R' - 0.3604G' + 0.5B'$$
$$Cr = 0.5R' - 0.4598G' - 0.0402B'$$

Y' is clamped to the range [0…1] and Cb and Cr are clamped to the range [-0.5…0.5]. The Y'CbCr quantization is limited range.

There is also an alternate constant luminance R'G'B' to Yc'CbcCrc (`V4L2_YCBCR_ENC_BT2020_CONST_LUM`) encoding:

Luma:

$$Yc' = (0.2627R + 0.6780G + 0.0593B)'$$
$$B' - Yc' \leq 0:$$
$$Cbc = (B' - Yc')/1.9404$$
$$B' - Yc' > 0:$$
$$Cbc = (B' - Yc')/1.5816$$
$$R' - Yc' \leq 0:$$
$$Crc = (R' - Y')/1.7184$$
$$R' - Yc' > 0:$$
$$Crc = (R' - Y')/0.9936$$

Yc' is clamped to the range [0…1] and Cbc and Crc are clamped to the range [-0.5 …0.5]. The Yc' CbcCrc quantization is limited range.

Colorspace DCI-P3 (V4L2_COLORSPACE_DCI_P3)

The *SMPTE RP 431-2* standard defines the colorspace used by cinema projectors that use the DCI-P3 colorspace. The default transfer function is `V4L2_XFER_FUNC_DCI_P3`. The default Y' CbCr encoding is `V4L2_YCBCR_ENC_709`. The default Y' CbCr quantization is limited range.

Note: Note that this colorspace standard does not specify a Y' CbCr encoding since it is not meant to be encoded to Y' CbCr. So this default Y' CbCr encoding was picked because it is the HDTV encoding.

The chromaticities of the primary colors and the white reference are:

Table 74: DCI-P3 Chromaticities

Color	x	y
Red	0.6800	0.3200
Green	0.2650	0.6900
Blue	0.1500	0.0600
White Reference	0.3140	0.3510

Transfer function:

$$L' = L^{\frac{1}{2.6}}$$

Inverse Transfer function:

$$L = L'^{(2.6)}$$

Y' CbCr encoding is not specified. V4L2 defaults to Rec. 709.

Colorspace SMPTE 240M (V4L2_COLORSPACE_SMPTE240M)

The *SMPTE 240M* standard was an interim standard used during the early days of HDTV (1988-1998). It has been superseded by Rec. 709. The default transfer function is `V4L2_XFER_FUNC_SMPTE240M`. The default Y' CbCr encoding is `V4L2_YCBCR_ENC_SMPTE240M`. The default Y' CbCr quantization is limited range. The chromaticities of the primary colors and the white reference are:

Table 75: SMPTE 240M Chromaticities

Color	x	y
Red	0.630	0.340
Green	0.310	0.595
Blue	0.155	0.070
White Reference (D65)	0.3127	0.3290

These chromaticities are identical to the SMPTE 170M colorspace.

Transfer function:

$$L' = 4L, \text{ for } 0 \leq L < 0.0228$$
$$L' = 1.1115L^{0.45} - 0.1115, \text{ for } 0.0228 \leq L \leq 1$$

Inverse Transfer function:

$$L = \frac{L'}{4}, \text{ for } 0 \leq L' < 0.0913$$
$$L = \left(\frac{L' + 0.1115}{1.1115}\right)^{\frac{1}{0.45}}, \text{ for } L' \geq 0.0913$$

The luminance (Y') and color difference (Cb and Cr) are obtained with the following `V4L2_YCBCR_ENC_SMPTE240M` encoding:

$$Y' = 0.2122R' + 0.7013G' + 0.0865B'$$
$$Cb = -0.1161R' - 0.3839G' + 0.5B'$$
$$Cr = 0.5R' - 0.4451G' - 0.0549B'$$

Y' is clamped to the range [0…1] and Cb and Cr are clamped to the range [-0.5…0.5]. The Y' CbCr quantization is limited range.

Colorspace NTSC 1953 (V4L2_COLORSPACE_470_SYSTEM_M)

This standard defines the colorspace used by NTSC in 1953. In practice this colorspace is obsolete and SMPTE 170M should be used instead. The default transfer function is `V4L2_XFER_FUNC_709`. The default Y' CbCr encoding is `V4L2_YCBCR_ENC_601`. The default Y' CbCr quantization is limited range. The chromaticities of the primary colors and the white reference are:

Table 76: NTSC 1953 Chromaticities

Color	x	y
Red	0.67	0.33
Green	0.21	0.71
Blue	0.14	0.08
White Reference (C)	0.310	0.316

Note: This colorspace uses Illuminant C instead of D65 as the white reference. To correctly convert an image in this colorspace to another that uses D65 you need to apply a chromatic adaptation algorithm such as the Bradford method.

The transfer function was never properly defined for NTSC 1953. The Rec. 709 transfer function is recommended in the literature:

$$L' = 4.5L\text{, for } 0 \le L < 0.018$$
$$L' = 1.099L^{0.45} - 0.099\text{, for } 0.018 \le L \le 1$$

Inverse Transfer function:

$$L = \frac{L'}{4.5}\text{, for } L' < 0.081$$
$$L = \left(\frac{L' + 0.099}{1.099}\right)^{\frac{1}{0.45}}\text{, for } L' \ge 0.081$$

The luminance (Y') and color difference (Cb and Cr) are obtained with the following `V4L2_YCBCR_ENC_601` encoding:

$$Y' = 0.2990R' + 0.5870G' + 0.1140B'$$
$$Cb = -0.1687R' - 0.3313G' + 0.5B'$$
$$Cr = 0.5R' - 0.4187G' - 0.0813B'$$

Y' is clamped to the range [0…1] and Cb and Cr are clamped to the range [-0.5…0.5]. The Y' CbCr quantization is limited range. This transform is identical to one defined in SMPTE 170M/BT.601.

Colorspace EBU Tech. 3213 (V4L2_COLORSPACE_470_SYSTEM_BG)

The *EBU Tech 3213* standard defines the colorspace used by PAL/SECAM in 1975. In practice this colorspace is obsolete and SMPTE 170M should be used instead. The default transfer function is `V4L2_XFER_FUNC_709`. The default Y' CbCr encoding is `V4L2_YCBCR_ENC_601`. The default Y' CbCr quantization is limited range. The chromaticities of the primary colors and the white reference are:

Table 77: EBU Tech. 3213 Chromaticities

Color	x	y
Red	0.64	0.33
Green	0.29	0.60
Blue	0.15	0.06
White Reference (D65)	0.3127	0.3290

The transfer function was never properly defined for this colorspace. The Rec. 709 transfer function is recommended in the literature:

$$L' = 4.5L, \text{ for } 0 \leq L < 0.018$$
$$L' = 1.099L^{0.45} - 0.099, \text{ for } 0.018 \leq L \leq 1$$

Inverse Transfer function:

$$L = \frac{L'}{4.5}, \text{ for } L' < 0.081$$
$$L = \left(\frac{L' + 0.099}{1.099}\right)^{\frac{1}{0.45}}, \text{ for } L' \geq 0.081$$

The luminance (Y') and color difference (Cb and Cr) are obtained with the following V4L2_YCBCR_ENC_601 encoding:

$$Y' = 0.2990R' + 0.5870G' + 0.1140B'$$
$$Cb = -0.1687R' - 0.3313G' + 0.5B'$$
$$Cr = 0.5R' - 0.4187G' - 0.0813B'$$

Y' is clamped to the range [0…1] and Cb and Cr are clamped to the range [-0.5…0.5]. The Y' CbCr quantization is limited range. This transform is identical to one defined in SMPTE 170M/BT.601.

Colorspace JPEG (V4L2_COLORSPACE_JPEG)

This colorspace defines the colorspace used by most (Motion-)JPEG formats. The chromaticities of the primary colors and the white reference are identical to sRGB. The transfer function use is V4L2_XFER_FUNC_SRGB. The Y' CbCr encoding is V4L2_YCBCR_ENC_601 with full range quantization where Y' is scaled to [0…255] and Cb/Cr are scaled to [-128…128] and then clipped to [-128…127].

Note: The JPEG standard does not actually store colorspace information. So if something other than sRGB is used, then the driver will have to set that information explicitly. Effectively V4L2_COLORSPACE_JPEG can be considered to be an abbreviation for V4L2_COLORSPACE_SRGB, V4L2_XFER_FUNC_SRGB, V4L2_YCBCR_ENC_601 and V4L2_QUANTIZATION_FULL_RANGE.

Detailed Transfer Function Descriptions

Transfer Function SMPTE 2084 (V4L2_XFER_FUNC_SMPTE2084)

The *SMPTE ST 2084* standard defines the transfer function used by High Dynamic Range content.

Constants:

m1 = (2610 / 4096) / 4

m2 = (2523 / 4096) * 128

c1 = 3424 / 4096

c2 = (2413 / 4096) * 32

c3 = (2392 / 4096) * 32

Transfer function:

$L' = ((c1 + c2 * L^{m1}) / (1 + c3 * L^{m1}))^{m2}$

Inverse Transfer function:

$L = (\max(L'^{1/m2} - c1, 0) / (c2 - c3 * L'^{1/m2}))^{1/m1}$

Take care when converting between this transfer function and non-HDR transfer functions: the linear RGB values [0…1] of HDR content map to a luminance range of 0 to 10000 cd/m^2 whereas the linear RGB values of non-HDR (aka Standard Dynamic Range or SDR) map to a luminance range of 0 to 100 cd/m^2.

To go from SDR to HDR you will have to divide L by 100 first. To go in the other direction you will have to multiply L by 100. Of course, this clamps all luminance values over 100 cd/m^2 to 100 cd/m^2.

There are better methods, see e.g. *colimg* for more in-depth information about this.

8.2.3 Input/Output

The V4L2 API defines several different methods to read from or write to a device. All drivers exchanging data with applications must support at least one of them.

The classic I/O method using the `read()` and `write()` function is automatically selected after opening a V4L2 device. When the driver does not support this method attempts to read or write will fail at any time.

Other methods must be negotiated. To select the streaming I/O method with memory mapped or user buffers applications call the *ioctl VIDIOC_REQBUFS* ioctl. The asynchronous I/O method is not defined yet.

Video overlay can be considered another I/O method, although the application does not directly receive the image data. It is selected by initiating video overlay with the *VIDIOC_S_FMT* ioctl. For more information see *Video Overlay Interface*.

Generally exactly one I/O method, including overlay, is associated with each file descriptor. The only exceptions are applications not exchanging data with a driver (“panel applications”, see *Opening and Closing Devices*) and drivers permitting simultaneous video capturing and overlay using the same file descriptor, for compatibility with V4L and earlier versions of V4L2.

VIDIOC_S_FMT and *ioctl VIDIOC_REQBUFS* would permit this to some degree, but for simplicity drivers need not support switching the I/O method (after first switching away from read/write) other than by closing and reopening the device.

The following sections describe the various I/O methods in more detail.

Read/Write

Input and output devices support the *read()* and *write()* function, respectively, when the `V4L2_CAP_READWRITE` flag in the `capabilities` field of struct *v4l2_capability* returned by the *ioctl VIDIOC_QUERYCAP* ioctl is set.

Drivers may need the CPU to copy the data, but they may also support DMA to or from user memory, so this I/O method is not necessarily less efficient than other methods merely exchanging buffer pointers. It is considered inferior though because no meta-information like frame counters or timestamps are passed. This information is necessary to recognize frame dropping and to synchronize with other data streams. However this is also the simplest I/O method, requiring little or no setup to exchange data. It permits command line stunts like this (the vidctrl tool is fictitious):

```
$ vidctrl /dev/video --input=0 --format=YUYV --size=352x288
$ dd if=/dev/video of=myimage.422 bs=202752 count=1
```

To read from the device applications use the *read()* function, to write the *write()* function. Drivers must implement one I/O method if they exchange data with applications, but it need not be this.[1] When reading or writing is supported, the driver must also support the *select()* and *poll()* function.[2]

Streaming I/O (Memory Mapping)

Input and output devices support this I/O method when the `V4L2_CAP_STREAMING` flag in the `capabilities` field of struct *v4l2_capability* returned by the *ioctl VIDIOC_QUERYCAP* ioctl is set. There are two streaming methods, to determine if the memory mapping flavor is supported applications must call the *ioctl VIDIOC_REQBUFS* ioctl with the memory type set to `V4L2_MEMORY_MMAP`.

Streaming is an I/O method where only pointers to buffers are exchanged between application and driver, the data itself is not copied. Memory mapping is primarily intended to map buffers in device memory into the application's address space. Device memory can be for example the video memory on a graphics card with a video capture add-on. However, being the most efficient I/O method available for a long time, many other drivers support streaming as well, allocating buffers in DMA-able main memory.

[1] It would be desirable if applications could depend on drivers supporting all I/O interfaces, but as much as the complex memory mapping I/O can be inadequate for some devices we have no reason to require this interface, which is most useful for simple applications capturing still images.

[2] At the driver level *select()* and *poll()* are the same, and *select()* is too important to be optional.

A driver can support many sets of buffers. Each set is identified by a unique buffer type value. The sets are independent and each set can hold a different type of data. To access different sets at the same time different file descriptors must be used.[1]

To allocate device buffers applications call the *ioctl VIDIOC_REQBUFS* ioctl with the desired number of buffers and buffer type, for example `V4L2_BUF_TYPE_VIDEO_CAPTURE`. This ioctl can also be used to change the number of buffers or to free the allocated memory, provided none of the buffers are still mapped.

Before applications can access the buffers they must map them into their address space with the `mmap()` function. The location of the buffers in device memory can be determined with the *ioctl VIDIOC_QUERYBUF* ioctl. In the single-planar API case, the `m.offset` and `length` returned in a struct `v4l2_buffer` are passed as sixth and second parameter to the `mmap()` function. When using the multi-planar API, struct `v4l2_buffer` contains an array of struct `v4l2_plane` structures, each containing its own `m.offset` and `length`. When using the multi-planar API, every plane of every buffer has to be mapped separately, so the number of calls to `mmap()` should be equal to number of buffers times number of planes in each buffer. The offset and length values must not be modified. Remember, the buffers are allocated in physical memory, as opposed to virtual memory, which can be swapped out to disk. Applications should free the buffers as soon as possible with the `munmap()` function.

Example: Mapping buffers in the single-planar API

```
struct v4l2_requestbuffers reqbuf;
struct {
    void *start;
    size_t length;
} *buffers;
unsigned int i;

memset(&reqbuf, 0, sizeof(reqbuf));
reqbuf.type = V4L2_BUF_TYPE_VIDEO_CAPTURE;
reqbuf.memory = V4L2_MEMORY_MMAP;
reqbuf.count = 20;

if (-1 == ioctl (fd, VIDIOC_REQBUFS, &reqbuf)) {
    if (errno == EINVAL)
        printf("Video capturing or mmap-streaming is not supported\\
↪n");
    else
        perror("VIDIOC_REQBUFS");
```

(continues on next page)

[1] One could use one file descriptor and set the buffer type field accordingly when calling *ioctl VIDIOC_QBUF, VIDIOC_DQBUF* etc., but it makes the `select()` function ambiguous. We also like the clean approach of one file descriptor per logical stream. Video overlay for example is also a logical stream, although the CPU is not needed for continuous operation.

(continued from previous page)

```
    exit(EXIT_FAILURE);
}

/* We want at least five buffers. */

if (reqbuf.count < 5) {
    /* You may need to free the buffers here. */
    printf("Not enough buffer memory\\n");
    exit(EXIT_FAILURE);
}

buffers = calloc(reqbuf.count, sizeof(*buffers));
assert(buffers != NULL);

for (i = 0; i < reqbuf.count; i++) {
    struct v4l2_buffer buffer;

    memset(&buffer, 0, sizeof(buffer));
    buffer.type = reqbuf.type;
    buffer.memory = V4L2_MEMORY_MMAP;
    buffer.index = i;

    if (-1 == ioctl (fd, VIDIOC_QUERYBUF, &buffer)) {
        perror("VIDIOC_QUERYBUF");
        exit(EXIT_FAILURE);
    }

    buffers[i].length = buffer.length; /* remember for munmap() */

    buffers[i].start = mmap(NULL, buffer.length,
                PROT_READ | PROT_WRITE, /* recommended */
                MAP_SHARED,             /* recommended */
                fd, buffer.m.offset);

    if (MAP_FAILED == buffers[i].start) {
        /* If you do not exit here you should unmap() and free()
           the buffers mapped so far. */
        perror("mmap");
        exit(EXIT_FAILURE);
    }
}

/* Cleanup. */

for (i = 0; i < reqbuf.count; i++)
    munmap(buffers[i].start, buffers[i].length);
```

Example: Mapping buffers in the multi-planar API

```
struct v4l2_requestbuffers reqbuf;
/* Our current format uses 3 planes per buffer */
#define FMT_NUM_PLANES = 3

struct {
    void *start[FMT_NUM_PLANES];
    size_t length[FMT_NUM_PLANES];
} *buffers;
unsigned int i, j;

memset(&reqbuf, 0, sizeof(reqbuf));
reqbuf.type = V4L2_BUF_TYPE_VIDEO_CAPTURE_MPLANE;
reqbuf.memory = V4L2_MEMORY_MMAP;
reqbuf.count = 20;

if (ioctl(fd, VIDIOC_REQBUFS, &reqbuf) < 0) {
    if (errno == EINVAL)
        printf("Video capturing or mmap-streaming is not supported\\
↪n");
    else
        perror("VIDIOC_REQBUFS");

    exit(EXIT_FAILURE);
}

/* We want at least five buffers. */

if (reqbuf.count < 5) {
    /* You may need to free the buffers here. */
    printf("Not enough buffer memory\\n");
    exit(EXIT_FAILURE);
}

buffers = calloc(reqbuf.count, sizeof(*buffers));
assert(buffers != NULL);

for (i = 0; i < reqbuf.count; i++) {
    struct v4l2_buffer buffer;
    struct v4l2_plane planes[FMT_NUM_PLANES];

    memset(&buffer, 0, sizeof(buffer));
    buffer.type = reqbuf.type;
    buffer.memory = V4L2_MEMORY_MMAP;
    buffer.index = i;
    /* length in struct v4l2_buffer in multi-planar API stores the␣
↪size
     * of planes array. */
    buffer.length = FMT_NUM_PLANES;
```

(continues on next page)

(continued from previous page)

```
    buffer.m.planes = planes;

    if (ioctl(fd, VIDIOC_QUERYBUF, &buffer) < 0) {
        perror("VIDIOC_QUERYBUF");
        exit(EXIT_FAILURE);
    }

    /* Every plane has to be mapped separately */
    for (j = 0; j < FMT_NUM_PLANES; j++) {
        buffers[i].length[j] = buffer.m.planes[j].length; /*␣
↪remember for munmap() */

        buffers[i].start[j] = mmap(NULL, buffer.m.planes[j].length,
                 PROT_READ | PROT_WRITE, /* recommended */
                 MAP_SHARED,             /* recommended */
                 fd, buffer.m.planes[j].m.offset);

        if (MAP_FAILED == buffers[i].start[j]) {
            /* If you do not exit here you should unmap() and free()
               the buffers and planes mapped so far. */
            perror("mmap");
            exit(EXIT_FAILURE);
        }
    }
}

/* Cleanup. */

for (i = 0; i < reqbuf.count; i++)
    for (j = 0; j < FMT_NUM_PLANES; j++)
        munmap(buffers[i].start[j], buffers[i].length[j]);
```

Conceptually streaming drivers maintain two buffer queues, an incoming and an outgoing queue. They separate the synchronous capture or output operation locked to a video clock from the application which is subject to random disk or network delays and preemption by other processes, thereby reducing the probability of data loss. The queues are organized as FIFOs, buffers will be output in the order enqueued in the incoming FIFO, and were captured in the order dequeued from the outgoing FIFO.

The driver may require a minimum number of buffers enqueued at all times to function, apart of this no limit exists on the number of buffers applications can enqueue in advance, or dequeue and process. They can also enqueue in a different order than buffers have been dequeued, and the driver can *fill* enqueued *empty* buffers in any order.[2] The index number of a buffer (struct `v4l2_buffer` index) plays no role here, it only identifies the buffer.

[2] Random enqueue order permits applications processing images out of order (such as video codecs) to return buffers earlier, reducing the probability of data loss. Random fill order allows drivers to reuse buffers on a LIFO-basis, taking advantage of caches holding scatter-gather lists and the like.

Initially all mapped buffers are in dequeued state, inaccessible by the driver. For capturing applications it is customary to first enqueue all mapped buffers, then to start capturing and enter the read loop. Here the application waits until a filled buffer can be dequeued, and re-enqueues the buffer when the data is no longer needed. Output applications fill and enqueue buffers, when enough buffers are stacked up the output is started with *VIDIOC_STREAMON*. In the write loop, when the application runs out of free buffers, it must wait until an empty buffer can be dequeued and reused.

To enqueue and dequeue a buffer applications use the *VIVIOC_QBUF* and *VIDIOC_DQBUF* ioctl. The status of a buffer being mapped, enqueued, full or empty can be determined at any time using the *ioctl VIDIOC_QUERYBUF* ioctl. Two methods exist to suspend execution of the application until one or more buffers can be dequeued. By default *VIDIOC_DQBUF* blocks when no buffer is in the outgoing queue. When the `O_NONBLOCK` flag was given to the *`open()`* function, *VIDIOC_DQBUF* returns immediately with an `EAGAIN` error code when no buffer is available. The *`select()`* or *`poll()`* functions are always available.

To start and stop capturing or output applications call the *VIDIOC_STREAMON* and *VIDIOC_STREAMOFF* ioctl.

Drivers implementing memory mapping I/O must support the *VIDIOC_REQBUFS*, *VIDIOC_QUERYBUF*, *VIDIOC_QBUF*, *VIDIOC_DQBUF*, *VIDIOC_STREAMON* and *VIDIOC_STREAMOFF* ioctls, the *mmap()*, *`munmap()`*, *select()* and *`poll()`* function.[3]

[capture example]

Streaming I/O (User Pointers)

Input and output devices support this I/O method when the `V4L2_CAP_STREAMING` flag in the `capabilities` field of struct *`v4l2_capability`* returned by the *ioctl VIDIOC_QUERYCAP* ioctl is set. If the particular user pointer method (not only memory mapping) is supported must be determined by calling the *ioctl VIDIOC_REQBUFS* ioctl with the memory type set to `V4L2_MEMORY_USERPTR`.

This I/O method combines advantages of the read/write and memory mapping methods. Buffers (planes) are allocated by the application itself, and can reside for example in virtual or shared memory. Only pointers to data are exchanged, these pointers and meta-information are passed in struct *`v4l2_buffer`* (or in struct *`v4l2_plane`* in the multi-planar API case). The driver must be switched into user pointer I/O mode by calling the *ioctl VIDIOC_REQBUFS* with the desired buffer type. No buffers (planes) are allocated beforehand, consequently they are not indexed and cannot be queried like mapped buffers with the *VIDIOC_QUERYBUF* ioctl.

[3] At the driver level *`select()`* and *`poll()`* are the same, and *`select()`* is too important to be optional. The rest should be evident.

Example: Initiating streaming I/O with user pointers

```
struct v4l2_requestbuffers reqbuf;

memset (&reqbuf, 0, sizeof (reqbuf));
reqbuf.type = V4L2_BUF_TYPE_VIDEO_CAPTURE;
reqbuf.memory = V4L2_MEMORY_USERPTR;

if (ioctl (fd, VIDIOC_REQBUFS, &reqbuf) == -1) {
    if (errno == EINVAL)
        printf ("Video capturing or user pointer streaming is not␣
↪supported\\n");
    else
        perror ("VIDIOC_REQBUFS");

    exit (EXIT_FAILURE);
}
```

Buffer (plane) addresses and sizes are passed on the fly with the *VIDIOC_QBUF* ioctl. Although buffers are commonly cycled, applications can pass different addresses and sizes at each *VIDIOC_QBUF* call. If required by the hardware the driver swaps memory pages within physical memory to create a continuous area of memory. This happens transparently to the application in the virtual memory subsystem of the kernel. When buffer pages have been swapped out to disk they are brought back and finally locked in physical memory for DMA.[1]

Filled or displayed buffers are dequeued with the *VIDIOC_DQBUF* ioctl. The driver can unlock the memory pages at any time between the completion of the DMA and this ioctl. The memory is also unlocked when *VIDIOC_STREAMOFF* is called, *ioctl VIDIOC_REQBUFS*, or when the device is closed. Applications must take care not to free buffers without dequeuing. Firstly, the buffers remain locked for longer, wasting physical memory. Secondly the driver will not be notified when the memory is returned to the application's free list and subsequently reused for other purposes, possibly completing the requested DMA and overwriting valuable data.

For capturing applications it is customary to enqueue a number of empty buffers, to start capturing and enter the read loop. Here the application waits until a filled buffer can be dequeued, and re-enqueues the buffer when the data is no longer needed. Output applications fill and enqueue buffers, when enough buffers are stacked up output is started. In the write loop, when the application runs out of free buffers it must wait until an empty buffer can be dequeued and reused. Two methods exist to suspend execution of the application until one or more buffers can be dequeued. By default *VIDIOC_DQBUF* blocks when no buffer is in the outgoing queue. When the `O_NONBLOCK` flag was given to the *`open()`* function,

[1] We expect that frequently used buffers are typically not swapped out. Anyway, the process of swapping, locking or generating scatter-gather lists may be time consuming. The delay can be masked by the depth of the incoming buffer queue, and perhaps by maintaining caches assuming a buffer will be soon enqueued again. On the other hand, to optimize memory usage drivers can limit the number of buffers locked in advance and recycle the most recently used buffers first. Of course, the pages of empty buffers in the incoming queue need not be saved to disk. Output buffers must be saved on the incoming and outgoing queue because an application may share them with other processes.

VIDIOC_DQBUF returns immediately with an EAGAIN error code when no buffer is available. The *select()* or *poll()* function are always available.

To start and stop capturing or output applications call the *VIDIOC_STREAMON* and *VIDIOC_STREAMOFF* ioctl.

Note: *VIDIOC_STREAMOFF* removes all buffers from both queues and unlocks all buffers as a side effect. Since there is no notion of doing anything "now" on a multitasking system, if an application needs to synchronize with another event it should examine the struct *v4l2_buffer* timestamp of captured or outputted buffers.

Drivers implementing user pointer I/O must support the *VIDIOC_REQBUFS*, *VIDIOC_QBUF*, *VIDIOC_DQBUF*, *VIDIOC_STREAMON* and *VIDIOC_STREAMOFF* ioctls, the *select()* and *poll()* function.[2]

Streaming I/O (DMA buffer importing)

The DMABUF framework provides a generic method for sharing buffers between multiple devices. Device drivers that support DMABUF can export a DMA buffer to userspace as a file descriptor (known as the exporter role), import a DMA buffer from userspace using a file descriptor previously exported for a different or the same device (known as the importer role), or both. This section describes the DMABUF importer role API in V4L2.

Refer to *DMABUF exporting* for details about exporting V4L2 buffers as DMABUF file descriptors.

Input and output devices support the streaming I/O method when the V4L2_CAP_STREAMING flag in the capabilities field of struct *v4l2_capability* returned by the *VIDIOC_QUERYCAP* ioctl is set. Whether importing DMA buffers through DMABUF file descriptors is supported is determined by calling the *VIDIOC_REQBUFS* ioctl with the memory type set to V4L2_MEMORY_DMABUF.

This I/O method is dedicated to sharing DMA buffers between different devices, which may be V4L devices or other video-related devices (e.g. DRM). Buffers (planes) are allocated by a driver on behalf of an application. Next, these buffers are exported to the application as file descriptors using an API which is specific for an allocator driver. Only such file descriptor are exchanged. The descriptors and meta-information are passed in struct *v4l2_buffer* (or in struct *v4l2_plane* in the multi-planar API case). The driver must be switched into DMABUF I/O mode by calling the *VIDIOC_REQBUFS* with the desired buffer type.

[2] At the driver level *select()* and *poll()* are the same, and *select()* is too important to be optional. The rest should be evident.

Example: Initiating streaming I/O with DMABUF file descriptors

```
struct v4l2_requestbuffers reqbuf;

memset(&reqbuf, 0, sizeof (reqbuf));
reqbuf.type = V4L2_BUF_TYPE_VIDEO_CAPTURE;
reqbuf.memory = V4L2_MEMORY_DMABUF;
reqbuf.count = 1;

if (ioctl(fd, VIDIOC_REQBUFS, &reqbuf) == -1) {
    if (errno == EINVAL)
        printf("Video capturing or DMABUF streaming is not␣
↪supported\\n");
    else
        perror("VIDIOC_REQBUFS");

    exit(EXIT_FAILURE);
}
```

The buffer (plane) file descriptor is passed on the fly with the *VIDIOC_QBUF* ioctl. In case of multiplanar buffers, every plane can be associated with a different DMABUF descriptor. Although buffers are commonly cycled, applications can pass a different DMABUF descriptor at each *VIDIOC_QBUF* call.

Example: Queueing DMABUF using single plane API

```
int buffer_queue(int v4lfd, int index, int dmafd)
{
    struct v4l2_buffer buf;

    memset(&buf, 0, sizeof buf);
    buf.type = V4L2_BUF_TYPE_VIDEO_CAPTURE;
    buf.memory = V4L2_MEMORY_DMABUF;
    buf.index = index;
    buf.m.fd = dmafd;

    if (ioctl(v4lfd, VIDIOC_QBUF, &buf) == -1) {
        perror("VIDIOC_QBUF");
        return -1;
    }

    return 0;
}
```

Example 3.6. Queueing DMABUF using multi plane API

```
int buffer_queue_mp(int v4lfd, int index, int dmafd[], int n_planes)
{
    struct v4l2_buffer buf;
    struct v4l2_plane planes[VIDEO_MAX_PLANES];
    int i;

    memset(&buf, 0, sizeof buf);
    buf.type = V4L2_BUF_TYPE_VIDEO_CAPTURE_MPLANE;
    buf.memory = V4L2_MEMORY_DMABUF;
    buf.index = index;
    buf.m.planes = planes;
    buf.length = n_planes;

    memset(&planes, 0, sizeof planes);

    for (i = 0; i < n_planes; ++i)
        buf.m.planes[i].m.fd = dmafd[i];

    if (ioctl(v4lfd, VIDIOC_QBUF, &buf) == -1) {
        perror("VIDIOC_QBUF");
        return -1;
    }

    return 0;
}
```

Captured or displayed buffers are dequeued with the *VIDIOC_DQBUF* ioctl. The driver can unlock the buffer at any time between the completion of the DMA and this ioctl. The memory is also unlocked when *VIDIOC_STREAMOFF* is called, *VIDIOC_REQBUFS*, or when the device is closed.

For capturing applications it is customary to enqueue a number of empty buffers, to start capturing and enter the read loop. Here the application waits until a filled buffer can be dequeued, and re-enqueues the buffer when the data is no longer needed. Output applications fill and enqueue buffers, when enough buffers are stacked up output is started. In the write loop, when the application runs out of free buffers it must wait until an empty buffer can be dequeued and reused. Two methods exist to suspend execution of the application until one or more buffers can be dequeued. By default *VIDIOC_DQBUF* blocks when no buffer is in the outgoing queue. When the `O_NONBLOCK` flag was given to the `open()` function, *VIDIOC_DQBUF* returns immediately with an `EAGAIN` error code when no buffer is available. The `select()` and `poll()` functions are always available.

To start and stop capturing or displaying applications call the *VIDIOC_STREAMON* and *VIDIOC_STREAMOFF* ioctls.

Note: *VIDIOC_STREAMOFF* removes all buffers from both queues and unlocks all buffers as a side effect. Since there is no notion of doing anything "now" on a multitasking system, if an application needs to synchronize with another event it should examine the struct *v4l2_buffer* timestamp of captured or outputted buffers.

Drivers implementing DMABUF importing I/O must support the *VIDIOC_REQBUFS*, *VIDIOC_QBUF*, *VIDIOC_DQBUF*, *VIDIOC_STREAMON* and *VIDIOC_STREAMOFF* ioctls, and the *select()* and *poll()* functions.

Asynchronous I/O

This method is not defined yet.

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 *v4l2_buffer*, argument to the *ioctl VIDIOC_QUERYBUF*, *VIDIOC_QBUF* and *VIDIOC_DQBUF* ioctl. In the multi-planar API, some plane-specific members of struct *v4l2_buffer*, such as pointers and sizes for each plane, are stored in struct *v4l2_plane* instead. In that case, struct *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 *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 *VIDIOC_S_INPUT* or *VIDIOC_S_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 VIDIOC_QUERYCTRL() or *VIDIOC_QUERY_EXT_CTRL()* for such a control while the stream is active.

Note: The *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 *VIDIOC_S_INPUT()*, *VIDIOC_S_OUTPUT()*, *VIDIOC_S_STD()* and *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 *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 *VIDIOC_STREAMOFF()* ioctl if it is running and free all buffers with the *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 *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 *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 *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 (*VIDIOC_S_FMT()*) or control changes (*VIDIOC_S_CTRL()* or *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.

Additionally, drivers shall return a EINVAL error from the *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.

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 *VIDIOC_TRY_FMT()* ioctl will return the required buffer size.

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

The *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).

type **v4l2_buffer**

struct v4l2_buffer

Table 78: struct v4l2_buffer

__u32	index	Number of the buffer, set by the application except when calling *VIDIOC_DQBUF*, then it is set by the driver. This field can range from zero to the number of buffers allocated with the *ioctl VIDIOC_REQBUFS* ioctl (struct *v4l2_requestbuffers* count), plus any buffers allocated with *ioctl VIDIOC_CREATE_BUFS* minus one.

continues on next page

Table 78 – continued from previous page

__u32	type	Type of the buffer, same as struct *v4l2_format* type or struct *v4l2_requestbuffers* type, set by the application. See *v4l2_buf_type*

continues on next page

Table 78 – continued from previous page

__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

Table 78 – continued from previous page

__u32	flags	Flags set by the application or driver, see *Buffer Flags*.
__u32	field	Indicates the field order of the image in the buffer, see *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.

continues on next page

Table 78 – continued from previous page

struct timeval	`timestamp`	For capture streams this is time when the first data byte was captured, as returned by the `clock_gettime()` function for the relevant clock id; see `V4L2_BUF_FLAG_TIMESTAMP_*` in *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

Table 78 – continued from previous page

struct *v4l2_timecode*	timecode	When the V4L2_BUF_FLAG_TIMECODE flag is set in flags, this structure contains a frame timecode. In *V4L2_FIELD_ALTERNATE* 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.

Table 78 – continued from previous page

__u32	sequence	Set by the driver, counting the frames (not fields!) in sequence. This field is set for both input and output devices.

In *V4L2_FIELD_ALTERNATE* 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 *Video Standards*.

continues on next page

Table 78 – continued from previous page

__u32	memory	This field must be set by applications and/or drivers in accordance with the selected I/O method. See v4l2_memory
union {	m	

continues on next page

Table 78 – continued from previous page

__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 the *mmap()* function not useful for applications. See *Streaming I/O (Memory Mapping)* for details

continues on next page

Table 78 – continued from previous page

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 *Streaming I/O (User Pointers)* for details.

continues on next page

Table 78 – continued from previous page

struct v4l2_plane	*planes	When using the multi-planar API, contains a userspace pointer to an array of struct *v4l2_plane*. The size of the array should be put in the length field of this struct *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.
}		

continues on next page

Table 78 – continued from previous page

__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 *ioctl VIDIOC_REQBUFS* and/or *ioctl 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.

continues on next page

Table 78 – continued from previous page

__u32	reserved2	A place holder for future extensions. Drivers and applications must set this to 0.

continues on next page

Table 78 – continued from previous page

__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 *ioctl VIDIOC_QBUF* and ignored by other ioctls that take a *v4l2_buffer* as argument. Applications should not set V4L2_BUF_FLAG_REQUEST_FD for any ioctls other than

Table 78 – continued from previous page

type **v4l2_plane**

struct v4l2_plane

__u32	bytesused	The number of bytes occupied by data in the plane (its payload). 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 plane (see length field of this struct) by the driver. **Note:** Note that the actual image data starts at data_offset which may not be 0.
__u32	length	Size in bytes of the plane (not its payload). This is set by the driver based on the calls to *ioctl VIDIOC_REQBUFS* and/or *ioctl VIDIOC_CREATE_BUFS*.
union {	m	

continues on next page

Table 79 – continued from previous page

__u32	mem_offset	When the memory type in the containing struct *v4l2_buffer* is V4L2_MEMORY_MMAP, this is the value that should be passed to *mmap()*, similar to the offset field in struct *v4l2_buffer*.
unsigned long	userptr	When the memory type in the containing struct *v4l2_buffer* is V4L2_MEMORY_USERPTR, this is a userspace pointer to the memory allocated for this plane by an application.
int	fd	When the memory type in the containing struct *v4l2_buffer* is V4L2_MEMORY_DMABUF, this is a file descriptor associated with a DMABUF buffer, similar to the fd field in struct *v4l2_buffer*.
}		

continues on next page

Table 79 – continued from previous page

__u32	data_offset	Offset in bytes to video data in the plane. Drivers must set this field when type refers to a capture stream, applications when it refers to an output stream. **Note:** That data_offset is included in bytesused. So the size of the image in the plane is bytesused-data_offset at offset data_offset from the start of the plane.
__u32	reserved[11]	Reserved for future use. Should be zeroed by drivers and applications.

type **v4l2_buf_type**

enum v4l2_buf_type

V4L2_BUF_TYPE_VIDEO_CAPTURE	1	Buffer of a single-planar video capture stream, see *Video Capture Interface*.
V4L2_BUF_TYPE_VIDEO_CAPTURE_MPLANE	9	Buffer of a multi-planar video capture stream, see *Video Capture Interface*.
V4L2_BUF_TYPE_VIDEO_OUTPUT	2	Buffer of a single-planar video output stream, see *Video Output Interface*.
V4L2_BUF_TYPE_VIDEO_OUTPUT_MPLANE	10	Buffer of a multi-planar video output stream, see *Video Output Interface*.
V4L2_BUF_TYPE_VIDEO_OVERLAY	3	Buffer for video overlay, see *Video Overlay Interface*.
V4L2_BUF_TYPE_VBI_CAPTURE	4	Buffer of a raw VBI capture stream, see *Raw VBI Data Interface*.
V4L2_BUF_TYPE_VBI_OUTPUT	5	Buffer of a raw VBI output stream, see *Raw VBI Data Interface*.
V4L2_BUF_TYPE_SLICED_VBI_CAPTURE	6	Buffer of a sliced VBI capture stream, see *Sliced VBI Data Interface*.
V4L2_BUF_TYPE_SLICED_VBI_OUTPUT	7	Buffer of a sliced VBI output stream, see *Sliced VBI Data Interface*.
V4L2_BUF_TYPE_VIDEO_OUTPUT_OVERLAY	8	Buffer for video output overlay (OSD), see *Video Output Overlay Interface*.
V4L2_BUF_TYPE_SDR_CAPTURE	11	Buffer for Software Defined Radio (SDR) capture stream, see *Software Defined Radio Interface (SDR)*.
V4L2_BUF_TYPE_SDR_OUTPUT	12	Buffer for Software Defined Radio (SDR) output stream, see *Software Defined Radio Interface (SDR)*.
V4L2_BUF_TYPE_META_CAPTURE	13	Buffer for metadata capture, see *Metadata Interface*.
V4L2_BUF_TYPE_META_OUTPUT	14	Buffer for metadata output, see *Metadata Interface*.

Buffer Flags

V4L2_BUF_FLAG_MAPPED	0x00000001	The buffer resides in device memory and has been mapped into the application's address space, see *Streaming I/O (Memory Mapping)* for details. Drivers set or clear this flag when the *ioctl VIDIOC_QUERYBUF*, *ioctl VIDIOC_QBUF, VIDIOC_DQBUF* or *VIDIOC_DQBUF* ioctl is called. Set by the driver.

continues on next page

Table 80 – continued from previous page

V4L2_BUF_FLAG_QUEUED	0x00000002	Internally drivers maintain two buffer queues an incoming and outgoing queue. When this flag is set, the buffer is currently on the incoming queue. It automatically moves to the outgoing queue after the buffer has been filled (capture devices) or displayed (output devices). Drivers set or clear this flag when the VIDIOC_QUERYBUF ioctl is called. After (successful) calling the VIDIOC_QBUFioctl it is always set and after VIDIOC_DQBUF always cleared.
V4L2_BUF_FLAG_DONE	0x00000004	When this flag is set, the buffer is currently on the outgoing queue, ready to be dequeued from the driver. Drivers set or clear this flag when the VIDIOC_QUERYBUF ioctl is called. After calling the VIDIOC_QBUF or VIDIOC_DQBUF it is always cleared. Of course a buffer cannot be on both queues at the same time, the V4L2_BUF_FLAG_QUEUED and V4L2_BUF_FLAG_DONE flag are mutually exclusive. They can be both cleared however, then the buffer is in "dequeued" state, in the application domain so to say.
V4L2_BUF_FLAG_ERROR	0x00000040	When this flag is set, the buffer has been dequeued successfully, although the data might have been corrupted. This is recoverable, streaming may continue as normal and the buffer may be reused normally. Drivers set this flag when the VIDIOC_DQBUF ioctl is called.
V4L2_BUF_FLAG_IN_REQUEST	0x00000080	This buffer is part of a request that hasn't been queued yet.
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	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	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.

continues on next page

Table 80 – continued from previous page

V4L2_BUF_FLAG_TIMECODE	0x00000100	The timecode field is valid. Drivers set o clear this flag when the VIDIOC_DQBUF ioctl i called. Applications can set this bit and th corresponding timecode structure when typ refers to an output stream.
V4L2_BUF_FLAG_PREPARED	0x00000400	The buffer has been prepared for I/O and can b queued by the application. Drivers set or clea this flag when the *ioctl VIDIOC_QUERYBUF* *VIDIOC_PREPARE_BUF*, *ioctl VIDIOC_QBUF* *VIDIOC_DQBUF* or *VIDIOC_DQBUF* ioctl i called.
V4L2_BUF_FLAG_NO_CACHE_INVALIDATE	0x00000800	Caches do not have to be invalidated for thi buffer. Typically applications shall use this fla if the data captured in the buffer is not goin to be touched by the CPU, instead the buffe will, probably, be passed on to a DMA-capabl hardware unit for further processing or outpu This flag is ignored unless the queue is use for *memory mapping* streaming I/O and report *V4L2_BUF_CAP_SUPPORTS_MMAP_CACHE_H* capability.
V4L2_BUF_FLAG_NO_CACHE_CLEAN	0x00001000	Caches do not have to be cleaned for this buffe Typically applications shall use this flag for out put buffers if the data in this buffer has not bee created by the CPU but by some DMA-capabl unit, in which case caches have not been use This flag is ignored unless the queue is use for *memory mapping* streaming I/O and report *V4L2_BUF_CAP_SUPPORTS_MMAP_CACHE_H* capability.
V4L2_BUF_FLAG_M2M_HOLD_CAPTURE_BUF	0x00000200	Only valid if V4L2_BUF_CAP_SUPPORTS_M2M_HOL is set. It is typically used with stateless de coders where multiple output buffers eac decode to a slice of the decoded frame. Ap plications can set this flag when queuein the output buffer to prevent the driver fron dequeueing the capture buffer after the outpu buffer has been decoded (i.e. the captur buffer is 'held'). If the timestamp of thi output buffer differs from that of the previou output buffer, then that indicates the start o a new frame and the previously held captur buffer is dequeued.

continues on next page

Table 80 – continued from previous page

V4L2_BUF_FLAG_LAST	0x00100000	Last buffer produced by the hardware mem2mem codec drivers set this flag or the capture queue for the last buffer when the *ioctl VIDIOC_QUERYBUF* or *VIDIOC_DQBUF* 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 Any subsequent call to the *VIDIOC_DQBUF* ioctl will not block anymore, but return an EPIPE error code.
V4L2_BUF_FLAG_REQUEST_FD	0x00800000	The request_fd field contains a valid file descriptor.
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	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 clock_gettime() using clock IDs CLOCK_MONOTONIC and CLOCK_REALTIME, respectively.
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 clock_gettime().
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	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	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.

continues on next page

Table 80 – continued from previous page

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

V4L2_MEMORY_MMAP	1	The buffer is used for *memory mapping* I/O.
V4L2_MEMORY_USERPTR	2	The buffer is used for *user pointer* I/O.
V4L2_MEMORY_OVERLAY	3	[to do]
V4L2_MEMORY_DMABUF	4	The buffer is used for *DMA shared buffer* I/O.

Timecodes

The `v4l2_buffer_timecode` structure is designed to hold a *SMPTE 12M* or similar timecode. (struct `timeval` timestamps are stored in the struct *`v4l2_buffer`* `timestamp` field.)

type **`v4l2_timecode`**

struct v4l2_timecode

__u32	type	Frame rate the timecodes are based on, see *Timecode Types*.
__u32	flags	Timecode flags, see *Timecode Flags*.
__u8	frames	Frame count, 0 …23/24/29/49/59, depending on the type of timecode.
__u8	seconds	Seconds count, 0 …59. This is a binary, not BCD number.
__u8	minutes	Minutes count, 0 …59. This is a binary, not BCD number.
__u8	hours	Hours count, 0 …29. This is a binary, not BCD number.
__u8	userbits[4]	The “user group” bits from the timecode.

Timecode Types

V4L2_TC_TYPE_24FPS	1	24 frames per second, i. e. film.
V4L2_TC_TYPE_25FPS	2	25 frames per second, i. e. PAL or SECAM video.
V4L2_TC_TYPE_30FPS	3	30 frames per second, i. e. NTSC video.
V4L2_TC_TYPE_50FPS	4	
V4L2_TC_TYPE_60FPS	5	

Timecode Flags

V4L2_TC_FLAG_DROPFRAME	0x0001	Indicates "drop frame" semantics for countin frames in 29.97 fps material. When set, fram numbers 0 and 1 at the start of each minute, ex cept minutes 0, 10, 20, 30, 40, 50 are omitte from the count.
V4L2_TC_FLAG_COLORFRAME	0x0002	The "color frame" flag.
V4L2_TC_USERBITS_field	0x000C	Field mask for the "binary group flags".
V4L2_TC_USERBITS_USERDEFINED	0x0000	Unspecified format.
V4L2_TC_USERBITS_8BITCHARS	0x0008	8-bit ISO characters.

Field Order

We have to distinguish between progressive and interlaced video. Progressive video transmits all lines of a video image sequentially. Interlaced video divides an image into two fields, containing only the odd and even lines of the image, respectively. Alternating the so called odd and even field are transmitted, and due to a small delay between fields a cathode ray TV displays the lines interleaved, yielding the original frame. This curious technique was invented because at refresh rates similar to film the image would fade out too quickly. Transmitting fields reduces the flicker without the necessity of doubling the frame rate and with it the bandwidth required for each channel.

It is important to understand a video camera does not expose one frame at a time, merely transmitting the frames separated into fields. The fields are in fact captured at two different instances in time. An object on screen may well move between one field and the next. For applications analysing motion it is of paramount importance to recognize which field of a frame is older, the *temporal order*.

When the driver provides or accepts images field by field rather than interleaved, it is also important applications understand how the fields combine to frames. We distinguish between top (aka odd) and bottom (aka even) fields, the *spatial order*: The first line of the top field is the first line of an interlaced frame, the first line of the bottom field is the second line of that frame.

However because fields were captured one after the other, arguing whether a frame commences with the top or bottom field is pointless. Any two successive top and bottom, or bottom and top fields yield a valid frame. Only when the source was progressive to begin with, e. g. when transferring film to video, two fields may come from the same frame, creating a natural order.

Counter to intuition the top field is not necessarily the older field. Whether the older field contains the top or bottom lines is a convention determined by the video standard. Hence the distinction between temporal and spatial order of fields. The diagrams below should make this clearer.

In V4L it is assumed that all video cameras transmit fields on the media bus in the same order they were captured, so if the top field was captured first (is the older field), the top field is also transmitted first on the bus.

All video capture and output devices must report the current field order. Some drivers may permit the selection of a different order, to this end applications initialize the `field` field of struct *v4l2_pix_format* before calling the *VIDIOC_S_FMT* ioctl. If this is not desired it should have the value `V4L2_FIELD_ANY` (0).

enum v4l2_field

type **v4l2_field**

V4L2_FIELD_ANY	0	Applications request this field order when any field format is acceptable. Drivers choose depending on hardware capabilities or e.g. the requested image size, and return the actual field order. Drivers must never return V4L2_FIELD_ANY. If multiple field orders are possible the driver must choose one of the possible field orders during *VIDIOC_S_FMT* or *VIDIOC_TRY_FMT*. struct v4l2_buffer field can never be V4L2_FIELD_ANY.
V4L2_FIELD_NONE	1	Images are in progressive (frame-based) format, not interlaced (field-based).
V4L2_FIELD_TOP	2	Images consist of the top (aka odd) field only.
V4L2_FIELD_BOTTOM	3	Images consist of the bottom (aka even) field only. Applications may wish to prevent a device from capturing interlaced images because they will have "comb"or "feathering" artefacts around moving objects.
V4L2_FIELD_INTERLACED	4	Images contain both fields, interleaved line by line. The temporal order of the fields (whether the top or bottom field is older) depends on the current video standard. In M/NTSC the bottom field is the older field. In all other standards the top field is the older field.
V4L2_FIELD_SEQ_TB	5	Images contain both fields, the top field lines are stored first in memory, immediately followed by the bottom field lines. Fields are always stored in temporal order, the older one first in memory. Image sizes refer to the frame, not fields.
V4L2_FIELD_SEQ_BT	6	Images contain both fields, the bottom field lines are stored first in memory, immediately followed by the top field lines. Fields are always stored in temporal order, the older one first in memory. Image sizes refer to the frame, not fields.

continues on next page

Table 81 – continued from previous page

V4L2_FIELD_ALTERNATE	7	The two fields of a frame are passed in separate buffers in temporal order, i. e. the older one first. To indicate the field parity (whether the current field is a top or bottom field) the driver or application, depending on data direction, must set struct v4l2_buffer field to V4L2_FIELD_TOP or V4L2_FIELD_BOTTOM. Any two successive fields pair to build a frame. If fields are successive, without any dropped fields between them (fields can drop individually), can be determined from the struct v4l2_buffer sequence field. This format cannot be selected when using the read/write I/O method since there is no way to communicate if a field was a top or bottom field.
V4L2_FIELD_INTERLACED_TB	8	Images contain both fields, interleaved line by line, top field first. The top field is the older field.
V4L2_FIELD_INTERLACED_BT	9	Images contain both fields, interleaved line by line, top field first. The bottom field is the older field.

Field Order, Top Field First Transmitted

Field Order, Bottom Field First Transmitted

8.2.4 Interfaces

Video Capture Interface

Video capture devices sample an analog video signal and store the digitized images in memory. Today nearly all devices can capture at full 25 or 30 frames/second. With this interface applications can control the capture process and move images from the driver into user space.

Conventionally V4L2 video capture devices are accessed through character device special files named `/dev/video` and `/dev/video0` to `/dev/video63` with major number 81 and minor numbers 0 to 63. `/dev/video` is typically a symbolic link to the preferred video device.

Note: The same device file names are used for video output devices.

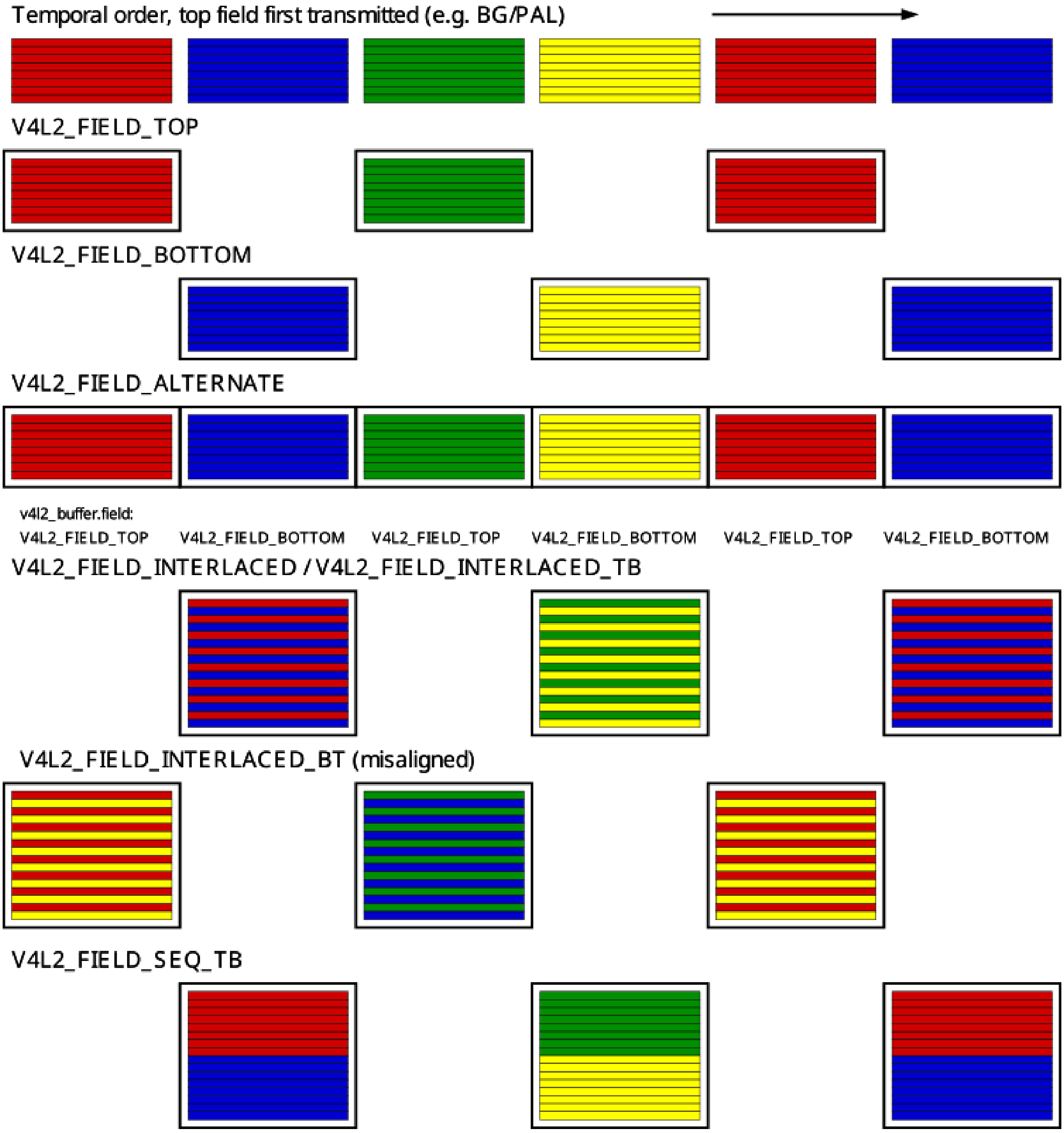

Fig. 6: Field Order, Top Field First Transmitted

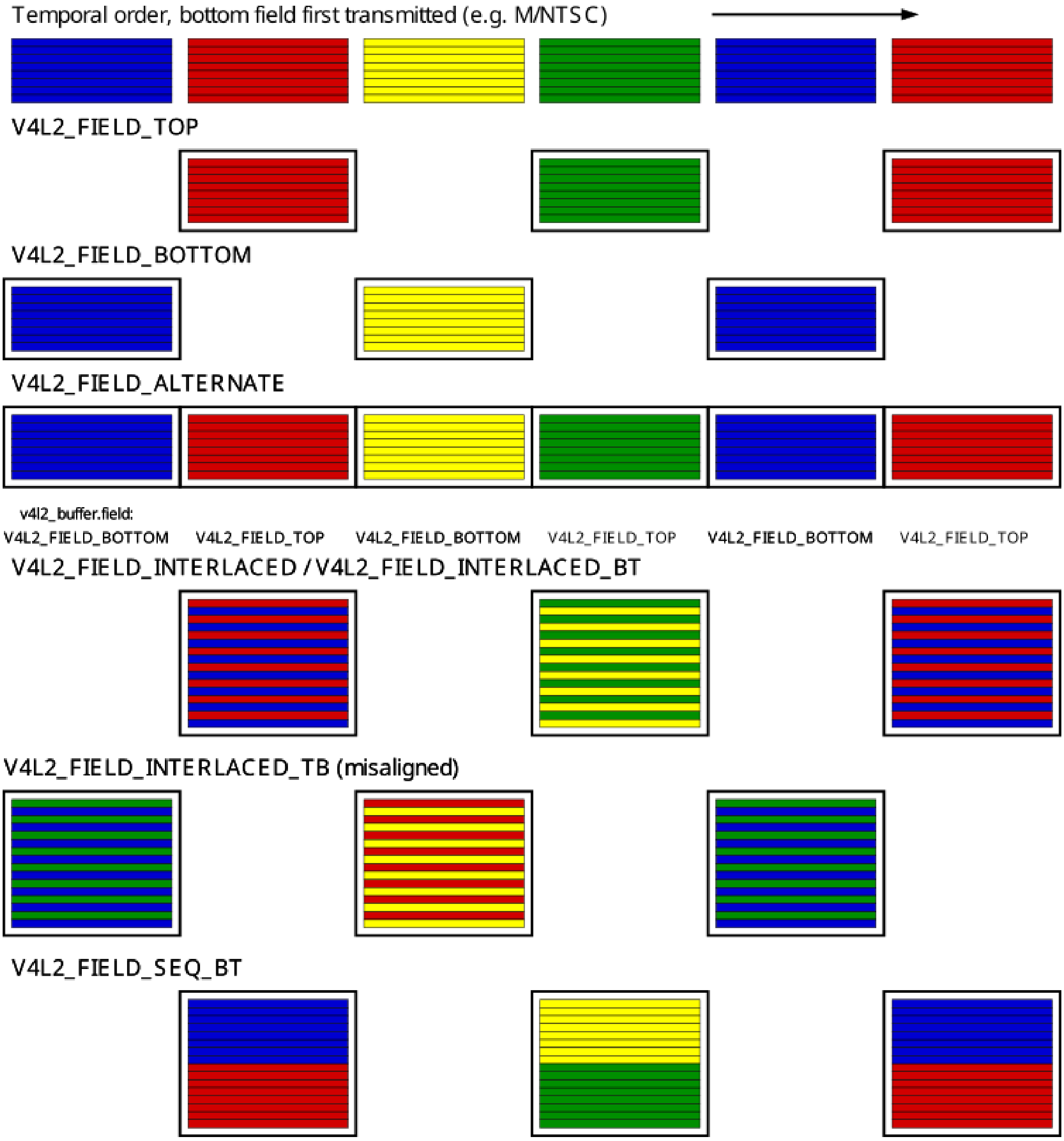

Fig. 7: Field Order, Bottom Field First Transmitted

Querying Capabilities

Devices supporting the video capture interface set the V4L2_CAP_VIDEO_CAPTURE or V4L2_CAP_VIDEO_CAPTURE_MPLANE flag in the capabilities field of struct *v4l2_capability* returned by the *ioctl VIDIOC_QUERYCAP* ioctl. As secondary device functions they may also support the *video overlay* (V4L2_CAP_VIDEO_OVERLAY) and the *raw VBI capture* (V4L2_CAP_VBI_CAPTURE) interface. At least one of the read/write or streaming I/O methods must be supported. Tuners and audio inputs are optional.

Supplemental Functions

Video capture devices shall support *audio input*, *Tuners and Modulators*, *controls*, *cropping and scaling* and *streaming parameter* ioctls as needed. The *video input* ioctls must be supported by all video capture devices.

Image Format Negotiation

The result of a capture operation is determined by cropping and image format parameters. The former select an area of the video picture to capture, the latter how images are stored in memory, i. e. in RGB or YUV format, the number of bits per pixel or width and height. Together they also define how images are scaled in the process.

As usual these parameters are *not* reset at *open()* time to permit Unix tool chains, programming a device and then reading from it as if it was a plain file. Well written V4L2 applications ensure they really get what they want, including cropping and scaling.

Cropping initialization at minimum requires to reset the parameters to defaults. An example is given in *Image Cropping, Insertion and Scaling – the CROP API*.

To query the current image format applications set the type field of a struct *v4l2_format* to V4L2_BUF_TYPE_VIDEO_CAPTURE or V4L2_BUF_TYPE_VIDEO_CAPTURE_MPLANE and call the *VIDIOC_G_FMT* ioctl with a pointer to this structure. Drivers fill the struct *v4l2_pix_format* pix or the struct *v4l2_pix_format_mplane* pix_mp member of the fmt union.

To request different parameters applications set the type field of a struct *v4l2_format* as above and initialize all fields of the struct *v4l2_pix_format* vbi member of the fmt union, or better just modify the results of *VIDIOC_G_FMT*, and call the *VIDIOC_S_FMT* ioctl with a pointer to this structure. Drivers may adjust the parameters and finally return the actual parameters as *VIDIOC_G_FMT* does.

Like *VIDIOC_S_FMT* the *VIDIOC_TRY_FMT* ioctl can be used to learn about hardware limitations without disabling I/O or possibly time consuming hardware preparations.

The contents of struct *v4l2_pix_format* and struct *v4l2_pix_format_mplane* are discussed in *Image Formats*. See also the specification of the *VIDIOC_G_FMT*, *VIDIOC_S_FMT* and *VIDIOC_TRY_FMT* ioctls for details. Video capture devices must implement both the *VIDIOC_G_FMT* and *VIDIOC_S_FMT* ioctl, even if *VIDIOC_S_FMT* ignores all requests and always returns default parameters as *VIDIOC_G_FMT* does. *VIDIOC_TRY_FMT* is optional.

Reading Images

A video capture device may support the *read() function* and/or streaming (*memory mapping* or *user pointer*) I/O. See *Input/Output* for details.

Video Overlay Interface

Also known as Framebuffer Overlay or Previewing.

Video overlay devices have the ability to genlock (TV-)video into the (VGA-)video signal of a graphics card, or to store captured images directly in video memory of a graphics card, typically with clipping. This can be considerable more efficient than capturing images and displaying them by other means. In the old days when only nuclear power plants needed cooling towers this used to be the only way to put live video into a window.

Video overlay devices are accessed through the same character special files as *video capture* devices.

Note: The default function of a `/dev/video` device is video capturing. The overlay function is only available after calling the *VIDIOC_S_FMT* ioctl.

The driver may support simultaneous overlay and capturing using the read/write and streaming I/O methods. If so, operation at the nominal frame rate of the video standard is not guaranteed. Frames may be directed away from overlay to capture, or one field may be used for overlay and the other for capture if the capture parameters permit this.

Applications should use different file descriptors for capturing and overlay. This must be supported by all drivers capable of simultaneous capturing and overlay. Optionally these drivers may also permit capturing and overlay with a single file descriptor for compatibility with V4L and earlier versions of V4L2.[1]

[1] A common application of two file descriptors is the XFree86 *Xv/V4L* interface driver and a V4L2 application. While the X server controls video overlay, the application can take advantage of memory mapping and DMA.

In the opinion of the designers of this API, no driver writer taking the efforts to support simultaneous capturing and overlay will restrict this ability by requiring a single file descriptor, as in V4L and earlier versions of V4L2. Making this optional means applications depending on two file descriptors need backup routines to be compatible with all drivers, which is considerable more work than using two fds in applications which do not. Also two fd's fit the general concept of one file descriptor for each logical stream. Hence as a complexity trade-off drivers *must* support two file descriptors and *may* support single fd operation.

Querying Capabilities

Devices supporting the video overlay interface set the `V4L2_CAP_VIDEO_OVERLAY` flag in the `capabilities` field of struct `v4l2_capability` returned by the *ioctl VIDIOC_QUERYCAP* ioctl. The overlay I/O method specified below must be supported. Tuners and audio inputs are optional.

Supplemental Functions

Video overlay devices shall support *audio input*, *Tuners and Modulators*, *controls*, *cropping and scaling* and *streaming parameter* ioctls as needed. The *video input* and *video standard* ioctls must be supported by all video overlay devices.

Setup

Before overlay can commence applications must program the driver with frame buffer parameters, namely the address and size of the frame buffer and the image format, for example RGB 5:6:5. The *VIDIOC_G_FBUF* and *VIDIOC_S_FBUF* ioctls are available to get and set these parameters, respectively. The *VIDIOC_S_FBUF* ioctl is privileged because it allows to set up DMA into physical memory, bypassing the memory protection mechanisms of the kernel. Only the superuser can change the frame buffer address and size. Users are not supposed to run TV applications as root or with SUID bit set. A small helper application with suitable privileges should query the graphics system and program the V4L2 driver at the appropriate time.

Some devices add the video overlay to the output signal of the graphics card. In this case the frame buffer is not modified by the video device, and the frame buffer address and pixel format are not needed by the driver. The *VIDIOC_S_FBUF* ioctl is not privileged. An application can check for this type of device by calling the *VIDIOC_G_FBUF* ioctl.

A driver may support any (or none) of five clipping/blending methods:

1. Chroma-keying displays the overlaid image only where pixels in the primary graphics surface assume a certain color.
2. A bitmap can be specified where each bit corresponds to a pixel in the overlaid image. When the bit is set, the corresponding video pixel is displayed, otherwise a pixel of the graphics surface.
3. A list of clipping rectangles can be specified. In these regions *no* video is displayed, so the graphics surface can be seen here.
4. The framebuffer has an alpha channel that can be used to clip or blend the framebuffer with the video.
5. A global alpha value can be specified to blend the framebuffer contents with video images.

When simultaneous capturing and overlay is supported and the hardware prohibits different image and frame buffer formats, the format requested first takes precedence. The attempt to capture (*VIDIOC_S_FMT*) or overlay (*VIDIOC_S_FBUF*) may fail with an EBUSY error code or return accordingly modified parameters..

Overlay Window

The overlaid image is determined by cropping and overlay window parameters. The former select an area of the video picture to capture, the latter how images are overlaid and clipped. Cropping initialization at minimum requires to reset the parameters to defaults. An example is given in *Image Cropping, Insertion and Scaling – the CROP API*.

The overlay window is described by a struct *v4l2_window*. It defines the size of the image, its position over the graphics surface and the clipping to be applied. To get the current parameters applications set the `type` field of a struct `v4l2_format` to `V4L2_BUF_TYPE_VIDEO_OVERLAY` and call the *VIDIOC_G_FMT* ioctl. The driver fills the struct *v4l2_window* substructure named `win`. It is not possible to retrieve a previously programmed clipping list or bitmap.

To program the overlay window applications set the `type` field of a struct `v4l2_format` to `V4L2_BUF_TYPE_VIDEO_OVERLAY`, initialize the `win` substructure and call the *VIDIOC_S_FMT* ioctl. The driver adjusts the parameters against hardware limits and returns the actual parameters as *VIDIOC_G_FMT* does. Like *VIDIOC_S_FMT*, the *VIDIOC_TRY_FMT* ioctl can be used to learn about driver capabilities without actually changing driver state. Unlike *VIDIOC_S_FMT* this also works after the overlay has been enabled.

The scaling factor of the overlaid image is implied by the width and height given in struct *v4l2_window* and the size of the cropping rectangle. For more information see *Image Cropping, Insertion and Scaling – the CROP API*.

When simultaneous capturing and overlay is supported and the hardware prohibits different image and window sizes, the size requested first takes precedence. The attempt to capture or overlay as well (*VIDIOC_S_FMT*) may fail with an EBUSY error code or return accordingly modified parameters.

type **v4l2_window**

struct v4l2_window

struct v4l2_rect w
: Size and position of the window relative to the top, left corner of the frame buffer defined with *VIDIOC_S_FBUF*. The window can extend the frame buffer width and height, the `x` and `y` coordinates can be negative, and it can lie completely outside the frame buffer. The driver clips the window accordingly, or if that is not possible, modifies its size and/or position.

enum v4l2_field field
: Applications set this field to determine which video field shall be overlaid, typically one of `V4L2_FIELD_ANY` (0), `V4L2_FIELD_TOP`, `V4L2_FIELD_BOTTOM` or `V4L2_FIELD_INTERLACED`. Drivers may have to choose a different field order and return the actual setting here.

__u32 chromakey
: When chroma-keying has been negotiated with *VIDIOC_S_FBUF* applications set this field to the desired pixel value for the chroma key. The format is the same as the pixel format of the framebuffer (struct `v4l2_framebuffer fmt.pixelformat` field), with bytes in host order. E. g. for *V4L2_PIX_FMT_BGR24* the value should be 0xRRGGBB on a little endian, 0xBBGGRR on a big endian host.

struct v4l2_clip * clips
: When chroma-keying has *not* been negotiated and *VIDIOC_G_FBUF* indicated this capability, applications can set this field to point to an array of clipping rectangles.

  Like the window coordinates w, clipping rectangles are defined relative to the top, left corner of the frame buffer. However clipping rectangles must not extend the frame buffer width and height, and they must not overlap. If possible applications should merge adjacent rectangles. Whether this must create x-y or y-x bands, or the order of rectangles, is not defined. When clip lists are not supported the driver ignores this field. Its contents after calling *VIDIOC_S_FMT* are undefined.

__u32 clipcount
: When the application set the `clips` field, this field must contain the number of clipping rectangles in the list. When clip lists are not supported the driver ignores this field, its contents after calling *VIDIOC_S_FMT* are undefined. When clip lists are supported but no clipping is desired this field must be set to zero.

void * bitmap
: When chroma-keying has *not* been negotiated and *VIDIOC_G_FBUF* indicated this capability, applications can set this field to point to a clipping bit mask.

It must be of the same size as the window, `w.width` and `w.height`. Each bit corresponds to a pixel in the overlaid image, which is displayed only when the bit is *set*. Pixel coordinates translate to bits like:

```
((__u8 *) bitmap)[w.width * y + x / 8] & (1 << (x & 7))
```

where `0` ≤ x < `w.width` and `0` ≤ y <`w.height`.[2]

When a clipping bit mask is not supported the driver ignores this field, its contents after calling *VIDIOC_S_FMT* are undefined. When a bit mask is supported but no clipping is desired this field must be set to `NULL`.

Applications need not create a clip list or bit mask. When they pass both, or despite negotiating chroma-keying, the results are undefined. Regardless of the chosen method, the clipping abilities of the hardware may be limited in quantity or quality. The results when these limits are exceeded are undefined.[3]

__u8 global_alpha
: The global alpha value used to blend the framebuffer with video images, if

[2] Should we require `w.width` to be a multiple of eight?

[3] When the image is written into frame buffer memory it will be undesirable if the driver clips out less pixels than expected, because the application and graphics system are not aware these regions need to be refreshed. The driver should clip out more pixels or not write the image at all.

global alpha blending has been negotiated (V4L2_FBUF_FLAG_GLOBAL_ALPHA, see *VIDIOC_S_FBUF*, *Frame Buffer Flags*).

Note: This field was added in Linux 2.6.23, extending the structure. However the *VIDIOC_[G|S|TRY]_FMT* ioctls, which take a pointer to a v4l2_format parent structure with padding bytes at the end, are not affected.

---

type **v4l2_clip**

struct v4l2_clip[4]

struct v4l2_rect c
Coordinates of the clipping rectangle, relative to the top, left corner of the frame buffer. Only window pixels *outside* all clipping rectangles are displayed.

struct v4l2_clip * next
Pointer to the next clipping rectangle, NULL when this is the last rectangle. Drivers ignore this field, it cannot be used to pass a linked list of clipping rectangles.

type **v4l2_rect**

struct v4l2_rect

__s32 left
Horizontal offset of the top, left corner of the rectangle, in pixels.

__s32 top
Vertical offset of the top, left corner of the rectangle, in pixels. Offsets increase to the right and down.

__u32 width
Width of the rectangle, in pixels.

__u32 height
Height of the rectangle, in pixels.

Enabling Overlay

To start or stop the frame buffer overlay applications call the *ioctl VIDIOC_OVERLAY* ioctl.

[4] The X Window system defines "regions" which are vectors of `struct BoxRec { short x1, y1, x2, y2; }` with `width = x2 - x1` and `height = y2 - y1`, so one cannot pass X11 clip lists directly.

Video Output Interface

Video output devices encode stills or image sequences as analog video signal. With this interface applications can control the encoding process and move images from user space to the driver.

Conventionally V4L2 video output devices are accessed through character device special files named `/dev/video` and `/dev/video0` to `/dev/video63` with major number 81 and minor numbers 0 to 63. `/dev/video` is typically a symbolic link to the preferred video device.

Note: The same device file names are used also for video capture devices.

Querying Capabilities

Devices supporting the video output interface set the `V4L2_CAP_VIDEO_OUTPUT` or `V4L2_CAP_VIDEO_OUTPUT_MPLANE` flag in the `capabilities` field of struct *`v4l2_capability`* returned by the *ioctl VIDIOC_QUERYCAP* ioctl. As secondary device functions they may also support the *raw VBI output* (`V4L2_CAP_VBI_OUTPUT`) interface. At least one of the read/write or streaming I/O methods must be supported. Modulators and audio outputs are optional.

Supplemental Functions

Video output devices shall support *audio output*, *modulator*, *controls*, *cropping and scaling* and *streaming parameter* ioctls as needed. The *video output* ioctls must be supported by all video output devices.

Image Format Negotiation

The output is determined by cropping and image format parameters. The former select an area of the video picture where the image will appear, the latter how images are stored in memory, i. e. in RGB or YUV format, the number of bits per pixel or width and height. Together they also define how images are scaled in the process.

As usual these parameters are *not* reset at *`open()`* time to permit Unix tool chains, programming a device and then writing to it as if it was a plain file. Well written V4L2 applications ensure they really get what they want, including cropping and scaling.

Cropping initialization at minimum requires to reset the parameters to defaults. An example is given in *Image Cropping, Insertion and Scaling – the CROP API*.

To query the current image format applications set the `type` field of a struct *`v4l2_format`* to `V4L2_BUF_TYPE_VIDEO_OUTPUT` or `V4L2_BUF_TYPE_VIDEO_OUTPUT_MPLANE` and call the *VIDIOC_G_FMT* ioctl with a pointer to this structure. Drivers fill the struct *`v4l2_pix_format`* `pix` or the struct *`v4l2_pix_format_mplane`* `pix_mp` member of the `fmt` union.

To request different parameters applications set the `type` field of a struct *v4l2_format* as above and initialize all fields of the struct *v4l2_pix_format* `vbi` member of the `fmt` union, or better just modify the results of *VIDIOC_G_FMT*, and call the *VIDIOC_S_FMT* ioctl with a pointer to this structure. Drivers may adjust the parameters and finally return the actual parameters as *VIDIOC_G_FMT* does.

Like *VIDIOC_S_FMT* the *VIDIOC_TRY_FMT* ioctl can be used to learn about hardware limitations without disabling I/O or possibly time consuming hardware preparations.

The contents of struct *v4l2_pix_format* and struct *v4l2_pix_format_mplane* are discussed in *Image Formats*. See also the specification of the *VIDIOC_G_FMT*, *VIDIOC_S_FMT* and *VIDIOC_TRY_FMT* ioctls for details. Video output devices must implement both the *VIDIOC_G_FMT* and *VIDIOC_S_FMT* ioctl, even if *VIDIOC_S_FMT* ignores all requests and always returns default parameters as *VIDIOC_G_FMT* does. *VIDIOC_TRY_FMT* is optional.

Writing Images

A video output device may support the *write() function* and/or streaming (*memory mapping* or *user pointer*) I/O. See *Input/Output* for details.

Video Output Overlay Interface

Also known as On-Screen Display (OSD)

Some video output devices can overlay a framebuffer image onto the outgoing video signal. Applications can set up such an overlay using this interface, which borrows structures and ioctls of the *Video Overlay* interface.

The OSD function is accessible through the same character special file as the *Video Output* function.

Note: The default function of such a `/dev/video` device is video capturing or output. The OSD function is only available after calling the *VIDIOC_S_FMT* ioctl.

Querying Capabilities

Devices supporting the *Video Output Overlay* interface set the `V4L2_CAP_VIDEO_OUTPUT_OVERLAY` flag in the `capabilities` field of struct `v4l2_capability` returned by the *ioctl VIDIOC_QUERYCAP* ioctl.

Framebuffer

Contrary to the *Video Overlay* interface the framebuffer is normally implemented on the TV card and not the graphics card. On Linux it is accessible as a framebuffer device (`/dev/fbN`). Given a V4L2 device, applications can find the corresponding framebuffer device by calling the *VIDIOC_G_FBUF* ioctl. It returns, amongst other information, the physical address of the framebuffer in the `base` field of struct `v4l2_framebuffer`. The framebuffer device ioctl `FBIOGET_FSCREENINFO` returns the same address in the `smem_start` field of struct `fb_fix_screeninfo`. The `FBIOGET_FSCREENINFO` ioctl and struct `fb_fix_screeninfo` are defined in the `linux/fb.h` header file.

The width and height of the framebuffer depends on the current video standard. A V4L2 driver may reject attempts to change the video standard (or any other ioctl which would imply a framebuffer size change) with an `EBUSY` error code until all applications closed the framebuffer device.

Example: Finding a framebuffer device for OSD

```
#include <linux/fb.h>

struct v4l2_framebuffer fbuf;
unsigned int i;
int fb_fd;

if (-1 == ioctl(fd, VIDIOC_G_FBUF, &fbuf)) {
    perror("VIDIOC_G_FBUF");
    exit(EXIT_FAILURE);
}

for (i = 0; i < 30; i++) {
    char dev_name[16];
    struct fb_fix_screeninfo si;

    snprintf(dev_name, sizeof(dev_name), "/dev/fb%u", i);

    fb_fd = open(dev_name, O_RDWR);
    if (-1 == fb_fd) {
        switch (errno) {
        case ENOENT: /* no such file */
        case ENXIO:  /* no driver */
            continue;

        default:
            perror("open");
            exit(EXIT_FAILURE);
        }
    }
```

(continues on next page)

(continued from previous page)

```
    if (0 == ioctl(fb_fd, FBIOGET_FSCREENINFO, &si)) {
        if (si.smem_start == (unsigned long)fbuf.base)
            break;
    } else {
        /* Apparently not a framebuffer device. */
    }

    close(fb_fd);
    fb_fd = -1;
}

/* fb_fd is the file descriptor of the framebuffer device
   for the video output overlay, or -1 if no device was found. */
```

Overlay Window and Scaling

The overlay is controlled by source and target rectangles. The source rectangle selects a subsection of the framebuffer image to be overlaid, the target rectangle an area in the outgoing video signal where the image will appear. Drivers may or may not support scaling, and arbitrary sizes and positions of these rectangles. Further drivers may support any (or none) of the clipping/blending methods defined for the *Video Overlay* interface.

A struct *v4l2_window* defines the size of the source rectangle, its position in the framebuffer and the clipping/blending method to be used for the overlay. To get the current parameters applications set the type field of a struct v4l2_format to V4L2_BUF_TYPE_VIDEO_OUTPUT_OVERLAY and call the *VIDIOC_G_FMT* ioctl. The driver fills the struct *v4l2_window* substructure named win. It is not possible to retrieve a previously programmed clipping list or bitmap.

To program the source rectangle applications set the type field of a struct v4l2_format to V4L2_BUF_TYPE_VIDEO_OUTPUT_OVERLAY, initialize the win substructure and call the *VIDIOC_S_FMT* ioctl. The driver adjusts the parameters against hardware limits and returns the actual parameters as *VIDIOC_G_FMT* does. Like *VIDIOC_S_FMT*, the *VIDIOC_TRY_FMT* ioctl can be used to learn about driver capabilities without actually changing driver state. Unlike *VIDIOC_S_FMT* this also works after the overlay has been enabled.

A struct v4l2_crop defines the size and position of the target rectangle. The scaling factor of the overlay is implied by the width and height given in struct *v4l2_window* and struct v4l2_crop. The cropping API applies to *Video Output* and *Video Output Overlay* devices in the same way as to *Video Capture* and *Video Overlay* devices, merely reversing the direction of the data flow. For more information see *Image Cropping, Insertion and Scaling – the CROP API*.

Enabling Overlay

There is no V4L2 ioctl to enable or disable the overlay, however the framebuffer interface of the driver may support the FBIOBLANK ioctl.

Video Memory-To-Memory Interface

A V4L2 memory-to-memory device can compress, decompress, transform, or otherwise convert video data from one format into another format, in memory. Such memory-to-memory devices set the V4L2_CAP_VIDEO_M2M or V4L2_CAP_VIDEO_M2M_MPLANE capability. Examples of memory-to-memory devices are codecs, scalers, deinterlacers or format converters (i.e. converting from YUV to RGB).

A memory-to-memory video node acts just like a normal video node, but it supports both output (sending frames from memory to the hardware) and capture (receiving the processed frames from the hardware into memory) stream I/O. An application will have to setup the stream I/O for both sides and finally call *VIDIOC_STREAMON* for both capture and output to start the hardware.

Memory-to-memory devices function as a shared resource: you can open the video node multiple times, each application setting up their own properties that are local to the file handle, and each can use it independently from the others. The driver will arbitrate access to the hardware and reprogram it whenever another file handler gets access. This is different from the usual video node behavior where the video properties are global to the device (i.e. changing something through one file handle is visible through another file handle).

One of the most common memory-to-memory device is the codec. Codecs are more complicated than most and require additional setup for their codec parameters. This is done through codec controls. See *Codec Control Reference*. More details on how to use codec memory-to-memory devices are given in the following sections.

Memory-to-Memory Stateful Video Decoder Interface

A stateful video decoder takes complete chunks of the bytestream (e.g. Annex-B H.264/HEVC stream, raw VP8/9 stream) and decodes them into raw video frames in display order. The decoder is expected not to require any additional information from the client to process these buffers.

Performing software parsing, processing etc. of the stream in the driver in order to support this interface is strongly discouraged. In case such operations are needed, use of the Stateless Video Decoder Interface (in development) is strongly advised.

Conventions and Notations Used in This Document

1. The general V4L2 API rules apply if not specified in this document otherwise.
2. The meaning of words "must", "may", "should", etc. is as per RFC 2119.
3. All steps not marked "optional" are required.
4. `VIDIOC_G_EXT_CTRLS()` and `VIDIOC_S_EXT_CTRLS()` may be used interchangeably with `VIDIOC_G_CTRL()` and `VIDIOC_S_CTRL()`, unless specified otherwise.
5. Single-planar API (see *Single- and multi-planar APIs*) and applicable structures may be used interchangeably with multi-planar API, unless specified otherwise, depending on decoder capabilities and following the general V4L2 guidelines.
6. i = [a..b]: sequence of integers from a to b, inclusive, i.e. i = [0..2]: i = 0, 1, 2.
7. Given an `OUTPUT` buffer A, then A' represents a buffer on the `CAPTURE` queue containing data that resulted from processing buffer A.

Glossary

CAPTURE
: the destination buffer queue; for decoders, the queue of buffers containing decoded frames; for encoders, the queue of buffers containing an encoded bytestream; `V4L2_BUF_TYPE_VIDEO_CAPTURE` or `V4L2_BUF_TYPE_VIDEO_CAPTURE_MPLANE`; data is captured from the hardware into `CAPTURE` buffers.

client
: the application communicating with the decoder or encoder implementing this interface.

coded format
: encoded/compressed video bytestream format (e.g. H.264, VP8, etc.); see also: raw format.

coded height
: height for given coded resolution.

coded resolution
: stream resolution in pixels aligned to codec and hardware requirements; typically visible resolution rounded up to full macroblocks; see also: visible resolution.

coded width
: width for given coded resolution.

decode order
: the order in which frames are decoded; may differ from display order if the coded format includes a feature of frame reordering; for decoders, `OUTPUT` buffers must be queued by the client in decode order; for encoders `CAPTURE` buffers must be returned by the encoder in decode order.

destination
: data resulting from the decode process; see CAPTURE.

display order
: the order in which frames must be displayed; for encoders, OUTPUT buffers must be queued by the client in display order; for decoders, CAPTURE buffers must be returned by the decoder in display order.

DPB
: Decoded Picture Buffer; an H.264/HEVC term for a buffer that stores a decoded raw frame available for reference in further decoding steps.

EOS
: end of stream.

IDR
: Instantaneous Decoder Refresh; a type of a keyframe in an H.264/HEVC-encoded stream, which clears the list of earlier reference frames (DPBs).

keyframe
: an encoded frame that does not reference frames decoded earlier, i.e. can be decoded fully on its own.

macroblock
: a processing unit in image and video compression formats based on linear block transforms (e.g. H.264, VP8, VP9); codec-specific, but for most of popular codecs the size is 16x16 samples (pixels).

OUTPUT
: the source buffer queue; for decoders, the queue of buffers containing an encoded bytestream; for encoders, the queue of buffers containing raw frames; `V4L2_BUF_TYPE_VIDEO_OUTPUT` or `V4L2_BUF_TYPE_VIDEO_OUTPUT_MPLANE`; the hardware is fed with data from OUTPUT buffers.

PPS
: Picture Parameter Set; a type of metadata entity in an H.264/HEVC bytestream.

raw format
: uncompressed format containing raw pixel data (e.g. YUV, RGB formats).

resume point
: a point in the bytestream from which decoding may start/continue, without any previous state/data present, e.g.: a keyframe (VP8/VP9) or SPS/PPS/IDR sequence (H.264/HEVC); a resume point is required to start decode of a new stream, or to resume decoding after a seek.

source
: data fed to the decoder or encoder; see OUTPUT.

source height
: height in pixels for given source resolution; relevant to encoders only.

source resolution
: resolution in pixels of source frames being source to the encoder and subject to further cropping to the bounds of visible resolution; relevant to encoders only.

source width
: width in pixels for given source resolution; relevant to encoders only.

SPS
: Sequence Parameter Set; a type of metadata entity in an H.264/HEVC bytestream.

stream metadata
: additional (non-visual) information contained inside encoded bytestream; for example: coded resolution, visible resolution, codec profile.

visible height
: height for given visible resolution; display height.

visible resolution
: stream resolution of the visible picture, in pixels, to be used for display purposes; must be smaller or equal to coded resolution; display resolution.

visible width
: width for given visible resolution; display width.

State Machine

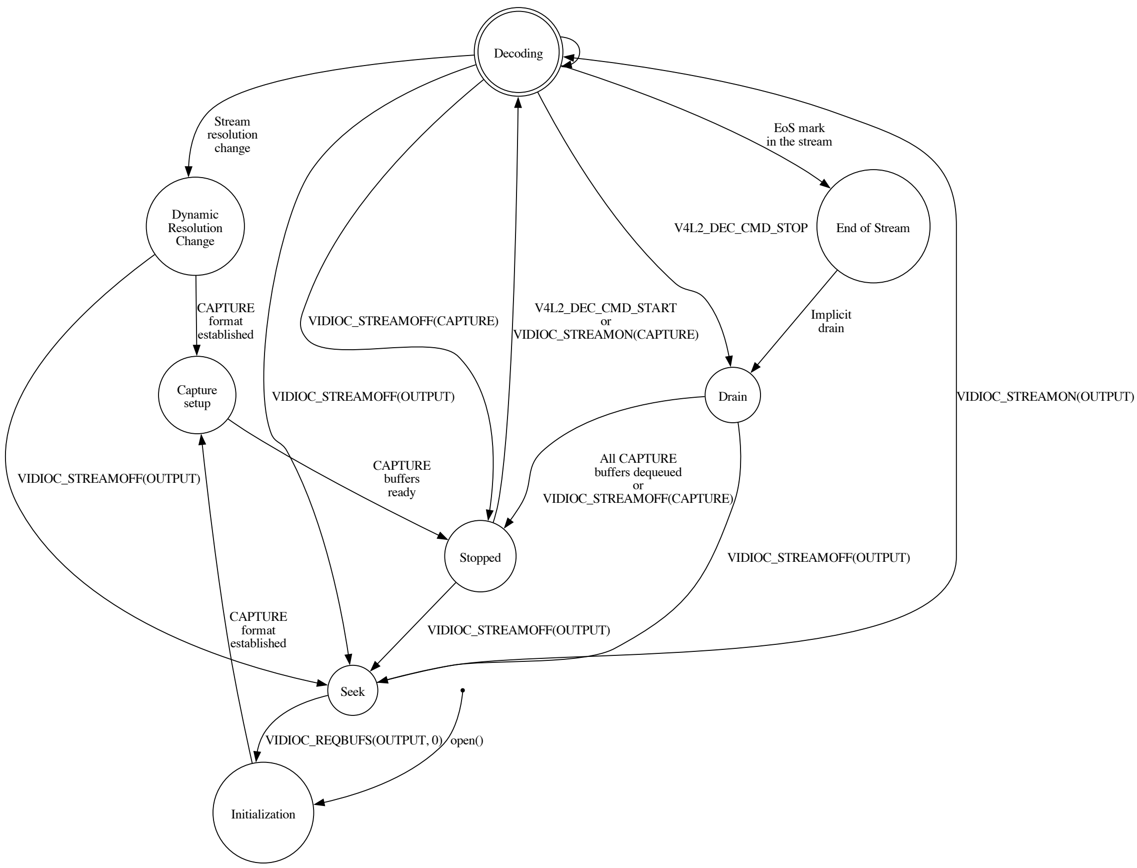

Fig. 8: Decoder State Machine

Querying Capabilities

1. To enumerate the set of coded formats supported by the decoder, the client may call `VIDIOC_ENUM_FMT()` on `OUTPUT`.
   - The full set of supported formats will be returned, regardless of the format set on `CAPTURE`.
   - Check the flags field of `v4l2_fmtdesc` for more information about the decoder' s capabilities with respect to each coded format. In particular whether or not the decoder has a full-fledged bytestream parser and if the decoder supports dynamic resolution changes.
2. To enumerate the set of supported raw formats, the client may call `VIDIOC_ENUM_FMT()` on `CAPTURE`.
   - Only the formats supported for the format currently active on `OUTPUT` will be returned.
   - In order to enumerate raw formats supported by a given coded format, the client must first set that coded format on `OUTPUT` and then enumerate formats on `CAPTURE`.
3. The client may use `VIDIOC_ENUM_FRAMESIZES()` to detect supported resolutions for a given format, passing desired pixel format in `v4l2_frmsizeenum` `pixel_format`.
   - Values returned by `VIDIOC_ENUM_FRAMESIZES()` for a coded pixel format will include all possible coded resolutions supported by the decoder for given coded pixel format.
   - Values returned by `VIDIOC_ENUM_FRAMESIZES()` for a raw pixel format will include all possible frame buffer resolutions supported by the decoder for given raw pixel format and the coded format currently set on `OUTPUT`.
4. Supported profiles and levels for the coded format currently set on `OUTPUT`, if applicable, may be queried using their respective controls via `VIDIOC_QUERYCTRL()`.

Initialization

1. Set the coded format on `OUTPUT` via `VIDIOC_S_FMT()`.
   - **Required fields:**

     `type`
        a `V4L2_BUF_TYPE_*` enum appropriate for `OUTPUT`.

     `pixelformat`
        a coded pixel format.

     `width`, `height`
        coded resolution of the stream; required only if it cannot be parsed from the stream for the given coded format; otherwise the decoder will use this resolution as a placeholder resolution that will likely

change as soon as it can parse the actual coded resolution from the stream.

sizeimage
desired size of OUTPUT buffers; the decoder may adjust it to match hardware requirements.

other fields
follow standard semantics.

- **Return fields:**

  sizeimage
  adjusted size of OUTPUT buffers.

- The CAPTURE format will be updated with an appropriate frame buffer resolution instantly based on the width and height returned by VIDIOC_S_FMT(). However, for coded formats that include stream resolution information, after the decoder is done parsing the information from the stream, it will update the CAPTURE format with new values and signal a source change event, regardless of whether they match the values set by the client or not.

Important: Changing the OUTPUT format may change the currently set CAPTURE format. How the new CAPTURE format is determined is up to the decoder and the client must ensure it matches its needs afterwards.

---

2. Allocate source (bytestream) buffers via VIDIOC_REQBUFS() on OUTPUT.

   - **Required fields:**

     count
     requested number of buffers to allocate; greater than zero.

     type
     a V4L2_BUF_TYPE_* enum appropriate for OUTPUT.

     memory
     follows standard semantics.

   - **Return fields:**

     count
     the actual number of buffers allocated.

   Warning: The actual number of allocated buffers may differ from the count given. The client must check the updated value of count after the call returns.

   Alternatively, VIDIOC_CREATE_BUFS() on the OUTPUT queue can be used to have more control over buffer allocation.

   - **Required fields:**

count
requested number of buffers to allocate; greater than zero.

type
a V4L2_BUF_TYPE_* enum appropriate for OUTPUT.

memory
follows standard semantics.

format
follows standard semantics.

- **Return fields:**

  count
  adjusted to the number of allocated buffers.

> **Warning:** The actual number of allocated buffers may differ from the count given. The client must check the updated value of count after the call returns.

3. Start streaming on the OUTPUT queue via VIDIOC_STREAMON().

4. **This step only applies to coded formats that contain resolution information in the stream.** Continue queuing/dequeuing bytestream buffers to/from the OUTPUT queue via VIDIOC_QBUF() and VIDIOC_DQBUF(). The buffers will be processed and returned to the client in order, until required metadata to configure the CAPTURE queue are found. This is indicated by the decoder sending a V4L2_EVENT_SOURCE_CHANGE event with changes set to V4L2_EVENT_SRC_CH_RESOLUTION.

   - It is not an error if the first buffer does not contain enough data for this to occur. Processing of the buffers will continue as long as more data is needed.
   - If data in a buffer that triggers the event is required to decode the first frame, it will not be returned to the client, until the initialization sequence completes and the frame is decoded.
   - If the client has not set the coded resolution of the stream on its own, calling VIDIOC_G_FMT(), VIDIOC_S_FMT(), VIDIOC_TRY_FMT() or VIDIOC_REQBUFS() on the CAPTURE queue will not return the real values for the stream until a V4L2_EVENT_SOURCE_CHANGE event with changes set to V4L2_EVENT_SRC_CH_RESOLUTION is signaled.

> **Important:** Any client query issued after the decoder queues the event will return values applying to the just parsed stream, including queue formats, selection rectangles and controls.

> **Note:** A client capable of acquiring stream parameters from the bytestream on its own may attempt to set the width and height of the OUTPUT format to non-zero values matching the coded size of the stream, skip this step and

continue with the *Capture Setup* sequence. However, it must not rely on any driver queries regarding stream parameters, such as selection rectangles and controls, since the decoder has not parsed them from the stream yet. If the values configured by the client do not match those parsed by the decoder, a *Dynamic Resolution Change* will be triggered to reconfigure them.

Note: No decoded frames are produced during this phase.

---

5. Continue with the *Capture Setup* sequence.

Capture Setup

1. Call `VIDIOC_G_FMT()` on the `CAPTURE` queue to get format for the destination buffers parsed/decoded from the bytestream.

   - **Required fields:**

     `type`
     a `V4L2_BUF_TYPE_*` enum appropriate for `CAPTURE`.

   - **Return fields:**

     `width`, `height`
     frame buffer resolution for the decoded frames.

     `pixelformat`
     pixel format for decoded frames.

     `num_planes` (for _MPLANE `type` only)
     number of planes for pixelformat.

     `sizeimage`, `bytesperline`
     as per standard semantics; matching frame buffer format.

   Note: The value of `pixelformat` may be any pixel format supported by the decoder for the current stream. The decoder should choose a preferred/optimal format for the default configuration. For example, a YUV format may be preferred over an RGB format if an additional conversion step would be required for the latter.

   ---

2. **Optional.** Acquire the visible resolution via `VIDIOC_G_SELECTION()`.

   - **Required fields:**

     `type`
     a `V4L2_BUF_TYPE_*` enum appropriate for `CAPTURE`.

     `target`
     set to `V4L2_SEL_TGT_COMPOSE`.

   - **Return fields:**

r.left, r.top, r.width, r.height
: the visible rectangle; it must fit within the frame buffer resolution returned by VIDIOC_G_FMT() on CAPTURE.

- The following selection targets are supported on CAPTURE:

  V4L2_SEL_TGT_CROP_BOUNDS
  corresponds to the coded resolution of the stream.

  V4L2_SEL_TGT_CROP_DEFAULT
  the rectangle covering the part of the CAPTURE buffer that contains meaningful picture data (visible area); width and height will be equal to the visible resolution of the stream.

  V4L2_SEL_TGT_CROP
  the rectangle within the coded resolution to be output to CAPTURE; defaults to V4L2_SEL_TGT_CROP_DEFAULT; read-only on hardware without additional compose/scaling capabilities.

  V4L2_SEL_TGT_COMPOSE_BOUNDS
  the maximum rectangle within a CAPTURE buffer, which the cropped frame can be composed into; equal to V4L2_SEL_TGT_CROP if the hardware does not support compose/scaling.

  V4L2_SEL_TGT_COMPOSE_DEFAULT
  equal to V4L2_SEL_TGT_CROP.

  V4L2_SEL_TGT_COMPOSE
  the rectangle inside a CAPTURE buffer into which the cropped frame is written; defaults to V4L2_SEL_TGT_COMPOSE_DEFAULT; read-only on hardware without additional compose/scaling capabilities.

  V4L2_SEL_TGT_COMPOSE_PADDED
  the rectangle inside a CAPTURE buffer which is overwritten by the hardware; equal to V4L2_SEL_TGT_COMPOSE if the hardware does not write padding pixels.

> **Warning:** The values are guaranteed to be meaningful only after the decoder successfully parses the stream metadata. The client must not rely on the query before that happens.

3. **Optional.** Enumerate CAPTURE formats via VIDIOC_ENUM_FMT() on the CAPTURE queue. Once the stream information is parsed and known, the client may use this ioctl to discover which raw formats are supported for given stream and select one of them via VIDIOC_S_FMT().

   Important: The decoder will return only formats supported for the currently established coded format, as per the OUTPUT format and/or stream metadata parsed in this initialization sequence, even if more formats may be supported by the decoder in general. In other words, the set returned will be a subset of the initial query mentioned in the *Querying Capabilities* section.

   For example, a decoder may support YUV and RGB formats for resolutions 1920x1088 and lower, but only YUV for higher resolutions (due to hardware limitations). After parsing a resolution of 1920x1088 or lower, `VIDIOC_ENUM_FMT()` may return a set of YUV and RGB pixel formats, but after parsing resolution higher than 1920x1088, the decoder will not return RGB, unsupported for this resolution.

However, subsequent resolution change event triggered after discovering a resolution change within the same stream may switch the stream into a lower resolution and `VIDIOC_ENUM_FMT()` would return RGB formats again in that case.

---

4. **Optional.** Set the `CAPTURE` format via `VIDIOC_S_FMT()` on the `CAPTURE` queue. The client may choose a different format than selected/suggested by the decoder in `VIDIOC_G_FMT()`.

   * **Required fields:**

     `type`
     a `V4L2_BUF_TYPE_*` enum appropriate for `CAPTURE`.

     `pixelformat`
     a raw pixel format.

     `width`, `height`
     frame buffer resolution of the decoded stream; typically unchanged from what was returned with `VIDIOC_G_FMT()`, but it may be different if the hardware supports composition and/or scaling.

 * Setting the `CAPTURE` format will reset the compose selection rectangles to their default values, based on the new resolution, as described in the previous step.

5. **Optional.** Set the compose rectangle via `VIDIOC_S_SELECTION()` on the `CAPTURE` queue if it is desired and if the decoder has compose and/or scaling capabilities.

   * **Required fields:**

     `type`
     a `V4L2_BUF_TYPE_*` enum appropriate for `CAPTURE`.

     `target`
     set to `V4L2_SEL_TGT_COMPOSE`.

     `r.left`, `r.top`, `r.width`, `r.height`
     the rectangle inside a `CAPTURE` buffer into which the cropped frame is written; defaults to `V4L2_SEL_TGT_COMPOSE_DEFAULT`; read-only on hardware without additional compose/scaling capabilities.

   * **Return fields:**

     `r.left`, `r.top`, `r.width`, `r.height`
     the visible rectangle; it must fit within the frame buffer resolution returned by `VIDIOC_G_FMT()` on `CAPTURE`.

> **Warning:** The decoder may adjust the compose rectangle to the nearest supported one to meet codec and hardware requirements. The client needs to check the adjusted rectangle returned by `VIDIOC_S_SELECTION()`.

6. If all the following conditions are met, the client may resume the decoding instantly:

   - `sizeimage` of the new format (determined in previous steps) is less than or equal to the size of currently allocated buffers,
   - the number of buffers currently allocated is greater than or equal to the minimum number of buffers acquired in previous steps. To fulfill this requirement, the client may use `VIDIOC_CREATE_BUFS()` to add new buffers.

   In that case, the remaining steps do not apply and the client may resume the decoding by one of the following actions:

   - if the `CAPTURE` queue is streaming, call `VIDIOC_DECODER_CMD()` with the `V4L2_DEC_CMD_START` command,
   - if the `CAPTURE` queue is not streaming, call `VIDIOC_STREAMON()` on the `CAPTURE` queue.

   However, if the client intends to change the buffer set, to lower memory usage or for any other reasons, it may be achieved by following the steps below.

7. **If the** `CAPTURE` **queue is streaming,** keep queuing and dequeuing buffers on the `CAPTURE` queue until a buffer marked with the `V4L2_BUF_FLAG_LAST` flag is dequeued.

8. **If the** `CAPTURE` **queue is streaming,** call `VIDIOC_STREAMOFF()` on the `CAPTURE` queue to stop streaming.

> **Warning:** The `OUTPUT` queue must remain streaming. Calling `VIDIOC_STREAMOFF()` on it would abort the sequence and trigger a seek.

9. **If the** `CAPTURE` **queue has buffers allocated,** free the `CAPTURE` buffers using `VIDIOC_REQBUFS()`.

   - **Required fields:**

     `count`
     set to 0.

     `type`
     a `V4L2_BUF_TYPE_*` enum appropriate for `CAPTURE`.

     `memory`
     follows standard semantics.

10. Allocate `CAPTURE` buffers via `VIDIOC_REQBUFS()` on the `CAPTURE` queue.

    - **Required fields:**

count
requested number of buffers to allocate; greater than zero.

type
a V4L2_BUF_TYPE_* enum appropriate for CAPTURE.

memory
follows standard semantics.

- **Return fields:**

  count
  actual number of buffers allocated.

Warning: The actual number of allocated buffers may differ from the count given. The client must check the updated value of count after the call returns.

Note: To allocate more than the minimum number of buffers (for pipeline depth), the client may query the V4L2_CID_MIN_BUFFERS_FOR_CAPTURE control to get the minimum number of buffers required, and pass the obtained value plus the number of additional buffers needed in the count field to VIDIOC_REQBUFS().

Alternatively, VIDIOC_CREATE_BUFS() on the CAPTURE queue can be used to have more control over buffer allocation. For example, by allocating buffers larger than the current CAPTURE format, future resolution changes can be accommodated.

- **Required fields:**

  count
  requested number of buffers to allocate; greater than zero.

  type
  a V4L2_BUF_TYPE_* enum appropriate for CAPTURE.

  memory
  follows standard semantics.

  format
  a format representing the maximum framebuffer resolution to be accommodated by newly allocated buffers.

- **Return fields:**

  count
  adjusted to the number of allocated buffers.

Warning: The actual number of allocated buffers may differ from the count given. The client must check the updated value of count after the call returns.

Note: To allocate buffers for a format different than parsed from the stream metadata, the client must proceed as follows, before the metadata parsing is initiated:

- set width and height of the OUTPUT format to desired coded resolution to let the decoder configure the CAPTURE format appropriately,
- query the CAPTURE format using VIDIOC_G_FMT() and save it until this step.

The format obtained in the query may be then used with VIDIOC_CREATE_BUFS() in this step to allocate the buffers.

11. Call VIDIOC_STREAMON() on the CAPTURE queue to start decoding frames.

Decoding

This state is reached after the *Capture Setup* sequence finishes successfully. In this state, the client queues and dequeues buffers to both queues via VIDIOC_QBUF() and VIDIOC_DQBUF(), following the standard semantics.

The content of the source OUTPUT buffers depends on the active coded pixel format and may be affected by codec-specific extended controls, as stated in the documentation of each format.

Both queues operate independently, following the standard behavior of V4L2 buffer queues and memory-to-memory devices. In addition, the order of decoded frames dequeued from the CAPTURE queue may differ from the order of queuing coded frames to the OUTPUT queue, due to properties of the selected coded format, e.g. frame reordering.

The client must not assume any direct relationship between CAPTURE and OUTPUT buffers and any specific timing of buffers becoming available to dequeue. Specifically:

- a buffer queued to OUTPUT may result in no buffers being produced on CAPTURE (e.g. if it does not contain encoded data, or if only metadata syntax structures are present in it),
- a buffer queued to OUTPUT may result in more than one buffer produced on CAPTURE (if the encoded data contained more than one frame, or if returning a decoded frame allowed the decoder to return a frame that preceded it in decode, but succeeded it in the display order),
- a buffer queued to OUTPUT may result in a buffer being produced on CAPTURE later into decode process, and/or after processing further OUTPUT buffers, or be returned out of order, e.g. if display reordering is used,
- buffers may become available on the CAPTURE queue without additional buffers queued to OUTPUT (e.g. during drain or EOS), because of the OUTPUT buffers queued in the past whose decoding results are only available at later time, due to specifics of the decoding process.

Note: To allow matching decoded `CAPTURE` buffers with `OUTPUT` buffers they originated from, the client can set the `timestamp` field of the `v4l2_buffer` struct when queuing an `OUTPUT` buffer. The `CAPTURE` buffer(s), which resulted from decoding that `OUTPUT` buffer will have their `timestamp` field set to the same value when dequeued.

In addition to the straightforward case of one `OUTPUT` buffer producing one `CAPTURE` buffer, the following cases are defined:

- one `OUTPUT` buffer generates multiple `CAPTURE` buffers: the same `OUTPUT` timestamp will be copied to multiple `CAPTURE` buffers.
- multiple `OUTPUT` buffers generate one `CAPTURE` buffer: timestamp of the `OUTPUT` buffer queued first will be copied.
- the decoding order differs from the display order (i.e. the `CAPTURE` buffers are out-of-order compared to the `OUTPUT` buffers): `CAPTURE` timestamps will not retain the order of `OUTPUT` timestamps.

During the decoding, the decoder may initiate one of the special sequences, as listed below. The sequences will result in the decoder returning all the `CAPTURE` buffers that originated from all the `OUTPUT` buffers processed before the sequence started. Last of the buffers will have the `V4L2_BUF_FLAG_LAST` flag set. To determine the sequence to follow, the client must check if there is any pending event and:

- if a `V4L2_EVENT_SOURCE_CHANGE` event with `changes` set to `V4L2_EVENT_SRC_CH_RESOLUTION` is pending, the *Dynamic Resolution Change* sequence needs to be followed,
- if a `V4L2_EVENT_EOS` event is pending, the *End of Stream* sequence needs to be followed.

Some of the sequences can be intermixed with each other and need to be handled as they happen. The exact operation is documented for each sequence.

Should a decoding error occur, it will be reported to the client with the level of details depending on the decoder capabilities. Specifically:

- the CAPTURE buffer that contains the results of the failed decode operation will be returned with the V4L2_BUF_FLAG_ERROR flag set,
- if the decoder is able to precisely report the OUTPUT buffer that triggered the error, such buffer will be returned with the V4L2_BUF_FLAG_ERROR flag set.

In case of a fatal failure that does not allow the decoding to continue, any further operations on corresponding decoder file handle will return the -EIO error code. The client may close the file handle and open a new one, or alternatively reinitialize the instance by stopping streaming on both queues, releasing all buffers and performing the Initialization sequence again.

Seek

Seek is controlled by the OUTPUT queue, as it is the source of coded data. The seek does not require any specific operation on the CAPTURE queue, but it may be affected as per normal decoder operation.

1. Stop the OUTPUT queue to begin the seek sequence via VIDIOC_STREAMOFF().
   - **Required fields:**

     type
     a V4L2_BUF_TYPE_* enum appropriate for OUTPUT.
   - The decoder will drop all the pending OUTPUT buffers and they must be treated as returned to the client (following standard semantics).
2. Restart the OUTPUT queue via VIDIOC_STREAMON().
   - **Required fields:**

     type
     a V4L2_BUF_TYPE_* enum appropriate for OUTPUT.
   - The decoder will start accepting new source bytestream buffers after the call returns.
3. Start queuing buffers containing coded data after the seek to the OUTPUT queue until a suitable resume point is found.

Note: There is no requirement to begin queuing coded data starting exactly from a resume point (e.g. SPS or a keyframe). Any queued OUTPUT buffers will be processed and returned to the client until a suitable resume point is found. While looking for a resume point, the decoder should not produce any decoded frames into CAPTURE buffers.

Some hardware is known to mishandle seeks to a non-resume point. Such an operation may result in an unspecified number of corrupted decoded frames being made available on the CAPTURE queue. Drivers must ensure that no fatal decoding errors or crashes occur, and implement any necessary handling and workarounds for hardware issues related to seek operations.

Warning: In case of the H.264/HEVC codec, the client must take care not to seek over a change of SPS/PPS. Even though the target frame could be a keyframe, the stale SPS/PPS inside decoder state would lead to undefined results when decoding. Although the decoder must handle that case without a crash or a fatal decode error, the client must not expect a sensible decode output.

If the hardware can detect such corrupted decoded frames, then corresponding buffers will be returned to the client with the V4L2_BUF_FLAG_ERROR set. See the *Decoding* section for further description of decode error reporting.

4. After a resume point is found, the decoder will start returning CAPTURE buffers containing decoded frames.

Important: A seek may result in the *Dynamic Resolution Change* sequence being initiated, due to the seek target having decoding parameters different from the part of the stream decoded before the seek. The sequence must be handled as per normal decoder operation.

Warning: It is not specified when the CAPTURE queue starts producing buffers containing decoded data from the OUTPUT buffers queued after the seek, as it operates independently from the OUTPUT queue.

The decoder may return a number of remaining CAPTURE buffers containing decoded frames originating from the OUTPUT buffers queued before the seek sequence is performed.

The VIDIOC_STREAMOFF operation discards any remaining queued OUTPUT buffers, which means that not all of the OUTPUT buffers queued before the seek sequence may have matching CAPTURE buffers produced. For example, given the sequence of operations on the OUTPUT queue:

> QBUF(A), QBUF(B), STREAMOFF(), STREAMON(), QBUF(G), QBUF(H),

any of the following results on the CAPTURE queue is allowed:

> {A' , B' , G' , H' }, {A' , G' , H' }, {G' , H' }.

To determine the CAPTURE buffer containing the first decoded frame after the seek, the client may observe the timestamps to match the CAPTURE and OUTPUT buffers or use V4L2_DEC_CMD_STOP and V4L2_DEC_CMD_START to drain the decoder.

Note: To achieve instantaneous seek, the client may restart streaming on the CAPTURE queue too to discard decoded, but not yet dequeued buffers.

Dynamic Resolution Change

Streams that include resolution metadata in the bytestream may require switching to a different resolution during the decoding.

Note: Not all decoders can detect resolution changes. Those that do set the V4L2_FMT_FLAG_DYN_RESOLUTION flag for the coded format when VIDIOC_ENUM_FMT() is called.

The sequence starts when the decoder detects a coded frame with one or more of the following parameters different from those previously established (and reflected by corresponding queries):

- coded resolution (`OUTPUT` width and height),
- visible resolution (selection rectangles),
- the minimum number of buffers needed for decoding,
- bit-depth of the bitstream has been changed.

Whenever that happens, the decoder must proceed as follows:

1. After encountering a resolution change in the stream, the decoder sends a `V4L2_EVENT_SOURCE_CHANGE` event with `changes` set to `V4L2_EVENT_SRC_CH_RESOLUTION`.

   Important: Any client query issued after the decoder queues the event will return values applying to the stream after the resolution change, including queue formats, selection rectangles and controls.

2. The decoder will then process and decode all remaining buffers from before the resolution change point.

   - The last buffer from before the change must be marked with the `V4L2_BUF_FLAG_LAST` flag, similarly to the *Drain* sequence above.

   Warning: The last buffer may be empty (with `v4l2_buffer` `bytesused` = 0) and in that case it must be ignored by the client, as it does not contain a decoded frame.

   Note: Any attempt to dequeue more `CAPTURE` buffers beyond the buffer marked with `V4L2_BUF_FLAG_LAST` will result in a -EPIPE error from `VIDIOC_DQBUF()`.

The client must continue the sequence as described below to continue the decoding process.

1. Dequeue the source change event.

   Important: A source change triggers an implicit decoder drain, similar to the explicit *Drain* sequence. The decoder is stopped after it completes. The decoding process must be resumed with either a pair of calls to `VIDIOC_STREAMOFF()` and `VIDIOC_STREAMON()` on the `CAPTURE` queue, or a call to `VIDIOC_DECODER_CMD()` with the `V4L2_DEC_CMD_START` command.

2. Continue with the *Capture Setup* sequence.

Note: During the resolution change sequence, the `OUTPUT` queue must remain streaming. Calling `VIDIOC_STREAMOFF()` on the `OUTPUT` queue would abort the sequence and initiate a seek.

In principle, the `OUTPUT` queue operates separately from the `CAPTURE` queue and this remains true for the duration of the entire resolution change sequence as well.

The client should, for best performance and simplicity, keep queuing/dequeuing buffers to/from the `OUTPUT` queue even while processing this sequence.

Drain

To ensure that all queued `OUTPUT` buffers have been processed and related `CAPTURE` buffers are given to the client, the client must follow the drain sequence described below. After the drain sequence ends, the client has received all decoded frames for all `OUTPUT` buffers queued before the sequence was started.

1. Begin drain by issuing `VIDIOC_DECODER_CMD()`.

   - **Required fields:**

     `cmd`
       set to `V4L2_DEC_CMD_STOP`.

     `flags`
       set to 0.

     `pts`
       set to 0.

   > **Warning:** The sequence can be only initiated if both `OUTPUT` and `CAPTURE` queues are streaming. For compatibility reasons, the call to `VIDIOC_DECODER_CMD()` will not fail even if any of the queues is not streaming, but at the same time it will not initiate the *Drain* sequence and so the steps described below would not be applicable.

2. Any `OUTPUT` buffers queued by the client before the `VIDIOC_DECODER_CMD()` was issued will be processed and decoded as normal. The client must continue to handle both queues independently, similarly to normal decode operation. This includes:

   - handling any operations triggered as a result of processing those buffers, such as the *Dynamic Resolution Change* sequence, before continuing with the drain sequence,

   - queuing and dequeuing `CAPTURE` buffers, until a buffer marked with the `V4L2_BUF_FLAG_LAST` flag is dequeued,

     > **Warning:** The last buffer may be empty (with `v4l2_buffer` `bytesused` = 0) and in that case it must be ignored by the client, as it does not contain a decoded frame.

     Note: Any attempt to dequeue more `CAPTURE` buffers beyond the buffer marked with `V4L2_BUF_FLAG_LAST` will result in a -EPIPE error from

VIDIOC_DQBUF().

- dequeuing processed OUTPUT buffers, until all the buffers queued before the V4L2_DEC_CMD_STOP command are dequeued,
- dequeuing the V4L2_EVENT_EOS event, if the client subscribed to it.

Note: For backwards compatibility, the decoder will signal a V4L2_EVENT_EOS event when the last frame has been decoded and all frames are ready to be dequeued. It is a deprecated behavior and the client must not rely on it. The V4L2_BUF_FLAG_LAST buffer flag should be used instead.

3. Once all the OUTPUT buffers queued before the V4L2_DEC_CMD_STOP call are dequeued and the last CAPTURE buffer is dequeued, the decoder is stopped and it will accept, but not process, any newly queued OUTPUT buffers until the client issues any of the following operations:
   - V4L2_DEC_CMD_START - the decoder will not be reset and will resume operation normally, with all the state from before the drain,
   - a pair of VIDIOC_STREAMOFF() and VIDIOC_STREAMON() on the CAPTURE queue - the decoder will resume the operation normally, however any CAPTURE buffers still in the queue will be returned to the client,
   - a pair of VIDIOC_STREAMOFF() and VIDIOC_STREAMON() on the OUTPUT queue - any pending source buffers will be returned to the client and the *Seek* sequence will be triggered.

Note: Once the drain sequence is initiated, the client needs to drive it to completion, as described by the steps above, unless it aborts the process by issuing VIDIOC_STREAMOFF() on any of the OUTPUT or CAPTURE queues. The client is not allowed to issue V4L2_DEC_CMD_START or V4L2_DEC_CMD_STOP again while the drain sequence is in progress and they will fail with -EBUSY error code if attempted.

Although not mandatory, the availability of decoder commands may be queried using VIDIOC_TRY_DECODER_CMD().

End of Stream

If the decoder encounters an end of stream marking in the stream, the decoder will initiate the *Drain* sequence, which the client must handle as described above, skipping the initial VIDIOC_DECODER_CMD().

Commit Points

Setting formats and allocating buffers trigger changes in the behavior of the decoder.

1. Setting the format on the OUTPUT queue may change the set of formats supported/advertised on the CAPTURE queue. In particular, it also means that the CAPTURE format may be reset and the client must not rely on the previously set format being preserved.
2. Enumerating formats on the CAPTURE queue always returns only formats supported for the current OUTPUT format.
3. Setting the format on the CAPTURE queue does not change the list of formats available on the OUTPUT queue. An attempt to set a CAPTURE format that is not supported for the currently selected OUTPUT format will result in the decoder adjusting the requested CAPTURE format to a supported one.
4. Enumerating formats on the OUTPUT queue always returns the full set of supported coded formats, irrespectively of the current CAPTURE format.
5. While buffers are allocated on any of the OUTPUT or CAPTURE queues, the client must not change the format on the OUTPUT queue. Drivers will return the -EBUSY error code for any such format change attempt.

To summarize, setting formats and allocation must always start with the OUTPUT queue and the OUTPUT queue is the master that governs the set of supported formats for the CAPTURE queue.

Memory-to-Memory Stateful Video Encoder Interface

A stateful video encoder takes raw video frames in display order and encodes them into a bytestream. It generates complete chunks of the bytestream, including all metadata, headers, etc. The resulting bytestream does not require any further post-processing by the client.

Performing software stream processing, header generation etc. in the driver in order to support this interface is strongly discouraged. In case such operations are needed, use of the Stateless Video Encoder Interface (in development) is strongly advised.

Conventions and Notations Used in This Document

1. The general V4L2 API rules apply if not specified in this document otherwise.
2. The meaning of words "must", "may", "should", etc. is as per RFC 2119.
3. All steps not marked "optional" are required.
4. VIDIOC_G_EXT_CTRLS() and VIDIOC_S_EXT_CTRLS() may be used interchangeably with VIDIOC_G_CTRL() and VIDIOC_S_CTRL(), unless specified otherwise.
5. Single-planar API (see *Single- and multi-planar APIs*) and applicable structures may be used interchangeably with multi-planar API, unless specified

otherwise, depending on encoder capabilities and following the general V4L2 guidelines.

6. i = [a..b]: sequence of integers from a to b, inclusive, i.e. i = [0..2]: i = 0, 1, 2.

7. Given an `OUTPUT` buffer A, then A' represents a buffer on the `CAPTURE` queue containing data that resulted from processing buffer A.

Glossary

Refer to *Glossary*.

State Machine

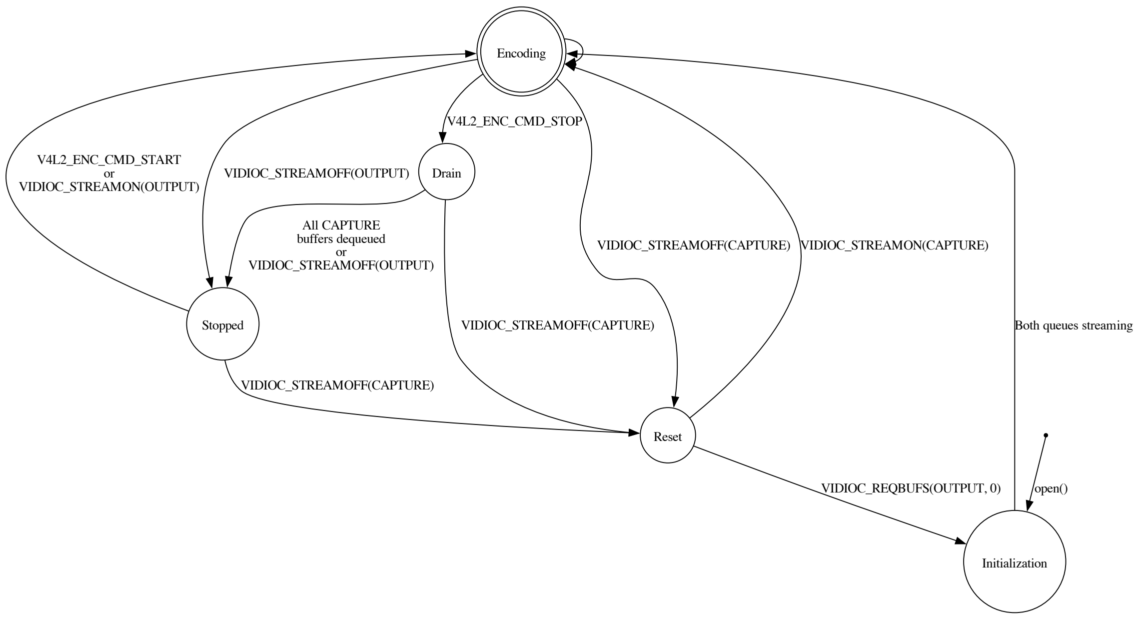

Fig. 9: Encoder State Machine

Querying Capabilities

1. To enumerate the set of coded formats supported by the encoder, the client may call `VIDIOC_ENUM_FMT()` on `CAPTURE`.

   - The full set of supported formats will be returned, regardless of the format set on `OUTPUT`.

2. To enumerate the set of supported raw formats, the client may call `VIDIOC_ENUM_FMT()` on `OUTPUT`.

   - Only the formats supported for the format currently active on `CAPTURE` will be returned.

  - In order to enumerate raw formats supported by a given coded format, the client must first set that coded format on `CAPTURE` and then enumerate the formats on `OUTPUT`.

3. The client may use `VIDIOC_ENUM_FRAMESIZES()` to detect supported resolutions for a given format, passing the desired pixel format in `v4l2_frmsizeenum` `pixel_format`.

   - Values returned by `VIDIOC_ENUM_FRAMESIZES()` for a coded pixel format will include all possible coded resolutions supported by the encoder for the given coded pixel format.
   - Values returned by `VIDIOC_ENUM_FRAMESIZES()` for a raw pixel format will include all possible frame buffer resolutions supported by the encoder for the given raw pixel format and coded format currently set on `CAPTURE`.

4. The client may use `VIDIOC_ENUM_FRAMEINTERVALS()` to detect supported frame intervals for a given format and resolution, passing the desired pixel format in `v4l2_frmsizeenum` `pixel_format` and the resolution in `v4l2_frmsizeenum` `width` and `v4l2_frmsizeenum` `height`.

   - Values returned by `VIDIOC_ENUM_FRAMEINTERVALS()` for a coded pixel format and coded resolution will include all possible frame intervals supported by the encoder for the given coded pixel format and resolution.
   - Values returned by `VIDIOC_ENUM_FRAMEINTERVALS()` for a raw pixel format and resolution will include all possible frame intervals supported by the encoder for the given raw pixel format and resolution and for the coded format, coded resolution and coded frame interval currently set on `CAPTURE`.
   - Support for `VIDIOC_ENUM_FRAMEINTERVALS()` is optional. If it is not implemented, then there are no special restrictions other than the limits of the codec itself.

5. Supported profiles and levels for the coded format currently set on `CAPTURE`, if applicable, may be queried using their respective controls via `VIDIOC_QUERYCTRL()`.

6. Any additional encoder capabilities may be discovered by querying their respective controls.

Initialization

1. Set the coded format on the `CAPTURE` queue via `VIDIOC_S_FMT()`.

   - **Required fields:**

     `type`
     a `V4L2_BUF_TYPE_*` enum appropriate for `CAPTURE`.

     `pixelformat`
     the coded format to be produced.

sizeimage
desired size of CAPTURE buffers; the encoder may adjust it to match hardware requirements.

width, height
ignored (read-only).

other fields
follow standard semantics.

- **Return fields:**

sizeimage
adjusted size of CAPTURE buffers.

width, height
the coded size selected by the encoder based on current state, e.g. OUTPUT format, selection rectangles, etc. (read-only).

Important: Changing the CAPTURE format may change the currently set OUTPUT format. How the new OUTPUT format is determined is up to the encoder and the client must ensure it matches its needs afterwards.

---

2. **Optional.** Enumerate supported OUTPUT formats (raw formats for source) for the selected coded format via `VIDIOC_ENUM_FMT()`.

   - **Required fields:**

   type
   a V4L2_BUF_TYPE_* enum appropriate for OUTPUT.

   other fields
   follow standard semantics.

   - **Return fields:**

   pixelformat
   raw format supported for the coded format currently selected on the CAPTURE queue.

   other fields
   follow standard semantics.

3. Set the raw source format on the OUTPUT queue via `VIDIOC_S_FMT()`.

   - **Required fields:**

   type
   a V4L2_BUF_TYPE_* enum appropriate for OUTPUT.

   pixelformat
   raw format of the source.

   width, height
   source resolution.

   other fields
   follow standard semantics.

- **Return fields:**

  `width`, `height`
  may be adjusted to match encoder minimums, maximums and alignment requirements, as required by the currently selected formats, as reported by `VIDIOC_ENUM_FRAMESIZES()`.

  other fields
  follow standard semantics.

- Setting the `OUTPUT` format will reset the selection rectangles to their default values, based on the new resolution, as described in the next step.

4. Set the raw frame interval on the `OUTPUT` queue via `VIDIOC_S_PARM()`. This also sets the coded frame interval on the `CAPTURE` queue to the same value.

   - ** Required fields:**

     `type`
     a `V4L2_BUF_TYPE_*` enum appropriate for `OUTPUT`.

     `parm.output`
     set all fields except `parm.output.timeperframe` to 0.

     `parm.output.timeperframe`
     the desired frame interval; the encoder may adjust it to match hardware requirements.

   - **Return fields:**

     `parm.output.timeperframe`
     the adjusted frame interval.

Important: Changing the `OUTPUT` frame interval *also* sets the framerate that the encoder uses to encode the video. So setting the frame interval to 1/24 (or 24 frames per second) will produce a coded video stream that can be played back at that speed. The frame interval for the `OUTPUT` queue is just a hint, the application may provide raw frames at a different rate. It can be used by the driver to help schedule multiple encoders running in parallel.

In the next step the `CAPTURE` frame interval can optionally be changed to a different value. This is useful for off-line encoding were the coded frame interval can be different from the rate at which raw frames are supplied.

Important: `timeperframe` deals with *frames*, not fields. So for interlaced formats this is the time per two fields, since a frame consists of a top and a bottom field.

Note: It is due to historical reasons that changing the `OUTPUT` frame interval also changes the coded frame interval on the `CAPTURE` queue. Ideally these would be independent settings, but that would break the existing API.

5. **Optional** Set the coded frame interval on the CAPTURE queue via VIDIOC_S_PARM(). This is only necessary if the coded frame interval is different from the raw frame interval, which is typically the case for off-line encoding. Support for this feature is signalled by the *V4L2_FMT_FLAG_ENC_CAP_FRAME_INTERVAL* format flag.

   - ** Required fields:**

     type
     : a V4L2_BUF_TYPE_* enum appropriate for CAPTURE.

     parm.capture
     : set all fields except parm.capture.timeperframe to 0.

     parm.capture.timeperframe
     : the desired coded frame interval; the encoder may adjust it to match hardware requirements.

   - **Return fields:**

     parm.capture.timeperframe
     : the adjusted frame interval.

   Important: Changing the CAPTURE frame interval sets the framerate for the coded video. It does *not* set the rate at which buffers arrive on the CAPTURE queue, that depends on how fast the encoder is and how fast raw frames are queued on the OUTPUT queue.

   Important: timeperframe deals with *frames*, not fields. So for interlaced formats this is the time per two fields, since a frame consists of a top and a bottom field.

   Note: Not all drivers support this functionality, in that case just set the desired coded frame interval for the OUTPUT queue.

   However, drivers that can schedule multiple encoders based on the OUTPUT frame interval must support this optional feature.

6. **Optional.** Set the visible resolution for the stream metadata via VIDIOC_S_SELECTION() on the OUTPUT queue if it is desired to be different than the full OUTPUT resolution.

   - **Required fields:**

     type
     : a V4L2_BUF_TYPE_* enum appropriate for OUTPUT.

     target
     : set to V4L2_SEL_TGT_CROP.

     r.left, r.top, r.width, r.height
     : visible rectangle; this must fit within the

V4L2_SEL_TGT_CROP_BOUNDS rectangle and may be subject to adjustment to match codec and hardware constraints.

- **Return fields:**

  r.left, r.top, r.width, r.height
  visible rectangle adjusted by the encoder.

- The following selection targets are supported on OUTPUT:

  V4L2_SEL_TGT_CROP_BOUNDS
  equal to the full source frame, matching the active OUTPUT format.

  V4L2_SEL_TGT_CROP_DEFAULT
  equal to V4L2_SEL_TGT_CROP_BOUNDS.

  V4L2_SEL_TGT_CROP
  rectangle within the source buffer to be encoded into the CAPTURE stream; defaults to V4L2_SEL_TGT_CROP_DEFAULT.

  Note: A common use case for this selection target is encoding a source video with a resolution that is not a multiple of a macroblock, e.g. the common 1920x1080 resolution may require the source buffers to be aligned to 1920x1088 for codecs with 16x16 macroblock size. To avoid encoding the padding, the client needs to explicitly configure this selection target to 1920x1080.

Warning: The encoder may adjust the crop/compose rectangles to the nearest supported ones to meet codec and hardware requirements. The client needs to check the adjusted rectangle returned by VIDIOC_S_SELECTION().

7. Allocate buffers for both OUTPUT and CAPTURE via VIDIOC_REQBUFS(). This may be performed in any order.

   - **Required fields:**

     count
     requested number of buffers to allocate; greater than zero.

     type
     a V4L2_BUF_TYPE_* enum appropriate for OUTPUT or CAPTURE.

     other fields
     follow standard semantics.

   - **Return fields:**

     count
     actual number of buffers allocated.

Warning: The actual number of allocated buffers may differ from the `count` given. The client must check the updated value of `count` after the call returns.

Note: To allocate more than the minimum number of OUTPUT buffers (for pipeline depth), the client may query the `V4L2_CID_MIN_BUFFERS_FOR_OUTPUT` control to get the minimum number of buffers required, and pass the obtained value plus the number of additional buffers needed in the `count` field to `VIDIOC_REQBUFS()`.

Alternatively, `VIDIOC_CREATE_BUFS()` can be used to have more control over buffer allocation.

- **Required fields:**

  `count`
  requested number of buffers to allocate; greater than zero.

  `type`
  a `V4L2_BUF_TYPE_*` enum appropriate for `OUTPUT`.

  other fields
  follow standard semantics.

- **Return fields:**

  `count`
  adjusted to the number of allocated buffers.

8. Begin streaming on both `OUTPUT` and `CAPTURE` queues via `VIDIOC_STREAMON()`. This may be performed in any order. The actual encoding process starts when both queues start streaming.

Note: If the client stops the `CAPTURE` queue during the encode process and then restarts it again, the encoder will begin generating a stream independent from the stream generated before the stop. The exact constraints depend on the coded format, but may include the following implications:

- encoded frames produced after the restart must not reference any frames produced before the stop, e.g. no long term references for H.264/HEVC,
- any headers that must be included in a standalone stream must be produced again, e.g. SPS and PPS for H.264/HEVC.

Encoding

This state is reached after the *Initialization* sequence finishes successfully. In this state, the client queues and dequeues buffers to both queues via `VIDIOC_QBUF()` and `VIDIOC_DQBUF()`, following the standard semantics.

The content of encoded `CAPTURE` buffers depends on the active coded pixel format and may be affected by codec-specific extended controls, as stated in the documentation of each format.

Both queues operate independently, following standard behavior of V4L2 buffer queues and memory-to-memory devices. In addition, the order of encoded frames dequeued from the `CAPTURE` queue may differ from the order of queuing raw frames to the `OUTPUT` queue, due to properties of the selected coded format, e.g. frame reordering.

The client must not assume any direct relationship between `CAPTURE` and `OUTPUT` buffers and any specific timing of buffers becoming available to dequeue. Specifically:

- a buffer queued to `OUTPUT` may result in more than one buffer produced on `CAPTURE` (for example, if returning an encoded frame allowed the encoder to return a frame that preceded it in display, but succeeded it in the decode order; however, there may be other reasons for this as well),
- a buffer queued to `OUTPUT` may result in a buffer being produced on `CAPTURE` later into encode process, and/or after processing further `OUTPUT` buffers, or be returned out of order, e.g. if display reordering is used,
- buffers may become available on the `CAPTURE` queue without additional buffers queued to `OUTPUT` (e.g. during drain or `EOS`), because of the `OUTPUT` buffers queued in the past whose encoding results are only available at later time, due to specifics of the encoding process,
- buffers queued to `OUTPUT` may not become available to dequeue instantly after being encoded into a corresponding `CAPTURE` buffer, e.g. if the encoder needs to use the frame as a reference for encoding further frames.

Note: To allow matching encoded `CAPTURE` buffers with `OUTPUT` buffers they originated from, the client can set the `timestamp` field of the `v4l2_buffer` struct when queuing an `OUTPUT` buffer. The `CAPTURE` buffer(s), which resulted from encoding that `OUTPUT` buffer will have their `timestamp` field set to the same value when dequeued.

In addition to the straightforward case of one `OUTPUT` buffer producing one `CAPTURE` buffer, the following cases are defined:

- one `OUTPUT` buffer generates multiple `CAPTURE` buffers: the same `OUTPUT` timestamp will be copied to multiple `CAPTURE` buffers,
- the encoding order differs from the presentation order (i.e. the `CAPTURE` buffers are out-of-order compared to the `OUTPUT` buffers): `CAPTURE` timestamps will not retain the order of `OUTPUT` timestamps.

Note: To let the client distinguish between frame types (keyframes, intermediate frames; the exact list of types depends on the coded format), the CAPTURE buffers will have corresponding flag bits set in their v4l2_buffer struct when dequeued. See the documentation of v4l2_buffer and each coded pixel format for exact list of flags and their meanings.

Should an encoding error occur, it will be reported to the client with the level of details depending on the encoder capabilities. Specifically:

- the CAPTURE buffer (if any) that contains the results of the failed encode operation will be returned with the V4L2_BUF_FLAG_ERROR flag set,
- if the encoder is able to precisely report the OUTPUT buffer(s) that triggered the error, such buffer(s) will be returned with the V4L2_BUF_FLAG_ERROR flag set.

Note: If a CAPTURE buffer is too small then it is just returned with the V4L2_BUF_FLAG_ERROR flag set. More work is needed to detect that this error occurred because the buffer was too small, and to provide support to free existing buffers that were too small.

In case of a fatal failure that does not allow the encoding to continue, any further operations on corresponding encoder file handle will return the -EIO error code. The client may close the file handle and open a new one, or alternatively reinitialize the instance by stopping streaming on both queues, releasing all buffers and performing the Initialization sequence again.

Encoding Parameter Changes

The client is allowed to use VIDIOC_S_CTRL() to change encoder parameters at any time. The availability of parameters is encoder-specific and the client must query the encoder to find the set of available controls.

The ability to change each parameter during encoding is encoder-specific, as per the standard semantics of the V4L2 control interface. The client may attempt to set a control during encoding and if the operation fails with the -EBUSY error code, the CAPTURE queue needs to be stopped for the configuration change to be allowed. To do this, it may follow the *Drain* sequence to avoid losing the already queued/encoded frames.

The timing of parameter updates is encoder-specific, as per the standard semantics of the V4L2 control interface. If the client needs to apply the parameters exactly at specific frame, using the Request API (*Request API*) should be considered, if supported by the encoder.

Drain

To ensure that all the queued OUTPUT buffers have been processed and the related CAPTURE buffers are given to the client, the client must follow the drain sequence described below. After the drain sequence ends, the client has received all encoded frames for all OUTPUT buffers queued before the sequence was started.

1. Begin the drain sequence by issuing VIDIOC_ENCODER_CMD().

   - **Required fields:**

     cmd
       set to V4L2_ENC_CMD_STOP.

     flags
       set to 0.

     pts
       set to 0.

   > **Warning:** The sequence can be only initiated if both OUTPUT and CAPTURE queues are streaming. For compatibility reasons, the call to VIDIOC_ENCODER_CMD() will not fail even if any of the queues is not streaming, but at the same time it will not initiate the *Drain* sequence and so the steps described below would not be applicable.

2. Any OUTPUT buffers queued by the client before the VIDIOC_ENCODER_CMD() was issued will be processed and encoded as normal. The client must continue to handle both queues independently, similarly to normal encode operation. This includes:

   - queuing and dequeuing CAPTURE buffers, until a buffer marked with the V4L2_BUF_FLAG_LAST flag is dequeued,

     > **Warning:** The last buffer may be empty (with v4l2_buffer bytesused = 0) and in that case it must be ignored by the client, as it does not contain an encoded frame.

     Note: Any attempt to dequeue more CAPTURE buffers beyond the buffer marked with V4L2_BUF_FLAG_LAST will result in a -EPIPE error from VIDIOC_DQBUF().

   - dequeuing processed OUTPUT buffers, until all the buffers queued before the V4L2_ENC_CMD_STOP command are dequeued,

   - dequeuing the V4L2_EVENT_EOS event, if the client subscribes to it.

   Note: For backwards compatibility, the encoder will signal a V4L2_EVENT_EOS event when the last frame has been encoded and all frames are ready to be dequeued. It is deprecated behavior and the client must not rely on it. The V4L2_BUF_FLAG_LAST buffer flag should be used instead.

3. Once all OUTPUT buffers queued before the V4L2_ENC_CMD_STOP call are dequeued and the last CAPTURE buffer is dequeued, the encoder is stopped and it will accept, but not process any newly queued OUTPUT buffers until the client issues any of the following operations:

   - V4L2_ENC_CMD_START - the encoder will not be reset and will resume operation normally, with all the state from before the drain,
   - a pair of VIDIOC_STREAMOFF() and VIDIOC_STREAMON() on the CAPTURE queue - the encoder will be reset (see the *Reset* sequence) and then resume encoding,
   - a pair of VIDIOC_STREAMOFF() and VIDIOC_STREAMON() on the OUTPUT queue - the encoder will resume operation normally, however any source frames queued to the OUTPUT queue between V4L2_ENC_CMD_STOP and VIDIOC_STREAMOFF() will be discarded.

Note: Once the drain sequence is initiated, the client needs to drive it to completion, as described by the steps above, unless it aborts the process by issuing VIDIOC_STREAMOFF() on any of the OUTPUT or CAPTURE queues. The client is not allowed to issue V4L2_ENC_CMD_START or V4L2_ENC_CMD_STOP again while the drain sequence is in progress and they will fail with -EBUSY error code if attempted.

For reference, handling of various corner cases is described below:

- In case of no buffer in the OUTPUT queue at the time the V4L2_ENC_CMD_STOP command was issued, the drain sequence completes immediately and the encoder returns an empty CAPTURE buffer with the V4L2_BUF_FLAG_LAST flag set.
- In case of no buffer in the CAPTURE queue at the time the drain sequence completes, the next time the client queues a CAPTURE buffer it is returned at once as an empty buffer with the V4L2_BUF_FLAG_LAST flag set.
- If VIDIOC_STREAMOFF() is called on the CAPTURE queue in the middle of the drain sequence, the drain sequence is canceled and all CAPTURE buffers are implicitly returned to the client.
- If VIDIOC_STREAMOFF() is called on the OUTPUT queue in the middle of the drain sequence, the drain sequence completes immediately and next CAPTURE buffer will be returned empty with the V4L2_BUF_FLAG_LAST flag set.

Although not mandatory, the availability of encoder commands may be queried using VIDIOC_TRY_ENCODER_CMD().

Reset

The client may want to request the encoder to reinitialize the encoding, so that the following stream data becomes independent from the stream data generated before. Depending on the coded format, that may imply that:

- encoded frames produced after the restart must not reference any frames produced before the stop, e.g. no long term references for H.264/HEVC,
- any headers that must be included in a standalone stream must be produced again, e.g. SPS and PPS for H.264/HEVC.

This can be achieved by performing the reset sequence.

1. Perform the *Drain* sequence to ensure all the in-flight encoding finishes and respective buffers are dequeued.
2. Stop streaming on the `CAPTURE` queue via `VIDIOC_STREAMOFF()`. This will return all currently queued `CAPTURE` buffers to the client, without valid frame data.
3. Start streaming on the `CAPTURE` queue via `VIDIOC_STREAMON()` and continue with regular encoding sequence. The encoded frames produced into `CAPTURE` buffers from now on will contain a standalone stream that can be decoded without the need for frames encoded before the reset sequence, starting at the first `OUTPUT` buffer queued after issuing the *V4L2_ENC_CMD_STOP* of the *Drain* sequence.

This sequence may be also used to change encoding parameters for encoders without the ability to change the parameters on the fly.

Commit Points

Setting formats and allocating buffers triggers changes in the behavior of the encoder.

1. Setting the format on the `CAPTURE` queue may change the set of formats supported/advertised on the `OUTPUT` queue. In particular, it also means that the `OUTPUT` format may be reset and the client must not rely on the previously set format being preserved.
2. Enumerating formats on the `OUTPUT` queue always returns only formats supported for the current `CAPTURE` format.
3. Setting the format on the `OUTPUT` queue does not change the list of formats available on the `CAPTURE` queue. An attempt to set the `OUTPUT` format that is not supported for the currently selected `CAPTURE` format will result in the encoder adjusting the requested `OUTPUT` format to a supported one.
4. Enumerating formats on the `CAPTURE` queue always returns the full set of supported coded formats, irrespective of the current `OUTPUT` format.
5. While buffers are allocated on any of the `OUTPUT` or `CAPTURE` queues, the client must not change the format on the `CAPTURE` queue. Drivers will return the -EBUSY error code for any such format change attempt.

To summarize, setting formats and allocation must always start with the CAPTURE queue and the CAPTURE queue is the master that governs the set of supported formats for the OUTPUT queue.

Memory-to-memory Stateless Video Decoder Interface

A stateless decoder is a decoder that works without retaining any kind of state between processed frames. This means that each frame is decoded independently of any previous and future frames, and that the client is responsible for maintaining the decoding state and providing it to the decoder with each decoding request. This is in contrast to the stateful video decoder interface, where the hardware and driver maintain the decoding state and all the client has to do is to provide the raw encoded stream and dequeue decoded frames in display order.

This section describes how user-space ("the client") is expected to communicate with stateless decoders in order to successfully decode an encoded stream. Compared to stateful codecs, the decoder/client sequence is simpler, but the cost of this simplicity is extra complexity in the client which is responsible for maintaining a consistent decoding state.

Stateless decoders make use of the *Request API*. A stateless decoder must expose the `V4L2_BUF_CAP_SUPPORTS_REQUESTS` capability on its `OUTPUT` queue when `VIDIOC_REQBUFS()` or `VIDIOC_CREATE_BUFS()` are invoked.

Depending on the encoded formats supported by the decoder, a single decoded frame may be the result of several decode requests (for instance, H.264 streams with multiple slices per frame). Decoders that support such formats must also expose the `V4L2_BUF_CAP_SUPPORTS_M2M_HOLD_CAPTURE_BUF` capability on their `OUTPUT` queue.

Querying capabilities

1. To enumerate the set of coded formats supported by the decoder, the client calls `VIDIOC_ENUM_FMT()` on the `OUTPUT` queue.
   - The driver must always return the full set of supported `OUTPUT` formats, irrespective of the format currently set on the `CAPTURE` queue.
   - Simultaneously, the driver must restrain the set of values returned by codec-specific capability controls (such as H.264 profiles) to the set actually supported by the hardware.
2. To enumerate the set of supported raw formats, the client calls `VIDIOC_ENUM_FMT()` on the `CAPTURE` queue.
   - The driver must return only the formats supported for the format currently active on the `OUTPUT` queue.
   - Depending on the currently set `OUTPUT` format, the set of supported raw formats may depend on the value of some codec-dependent controls. The client is responsible for making sure that these controls are set before querying the `CAPTURE` queue. Failure to do so will result in the default

values for these controls being used, and a returned set of formats that may not be usable for the media the client is trying to decode.

3. The client may use `VIDIOC_ENUM_FRAMESIZES()` to detect supported resolutions for a given format, passing desired pixel format in `v4l2_frmsizeenum`'s `pixel_format`.
4. Supported profiles and levels for the current `OUTPUT` format, if applicable, may be queried using their respective controls via `VIDIOC_QUERYCTRL()`.

Initialization

1. Set the coded format on the `OUTPUT` queue via `VIDIOC_S_FMT()`.
   - **Required fields:**

     `type`
     a `V4L2_BUF_TYPE_*` enum appropriate for `OUTPUT`.

     `pixelformat`
     a coded pixel format.

     `width`, `height`
     coded width and height parsed from the stream.

     other fields
     follow standard semantics.

   Note: Changing the `OUTPUT` format may change the currently set `CAPTURE` format. The driver will derive a new `CAPTURE` format from the `OUTPUT` format being set, including resolution, colorimetry parameters, etc. If the client needs a specific `CAPTURE` format, it must adjust it afterwards.

2. Call `VIDIOC_S_EXT_CTRLS()` to set all the controls (parsed headers, etc.) required by the `OUTPUT` format to enumerate the `CAPTURE` formats.
3. Call `VIDIOC_G_FMT()` for `CAPTURE` queue to get the format for the destination buffers parsed/decoded from the bytestream.
   - **Required fields:**

     `type`
     a `V4L2_BUF_TYPE_*` enum appropriate for `CAPTURE`.

   - **Returned fields:**

     `width`, `height`
     frame buffer resolution for the decoded frames.

     `pixelformat`
     pixel format for decoded frames.

     `num_planes` (for _MPLANE type only)
     number of planes for pixelformat.

     `sizeimage`, `bytesperline`
     as per standard semantics; matching frame buffer format.

Note: The value of `pixelformat` may be any pixel format supported for the `OUTPUT` format, based on the hardware capabilities. It is suggested that the driver chooses the preferred/optimal format for the current configuration. For example, a YUV format may be preferred over an RGB format, if an additional conversion step would be required for RGB.

---

4. *[optional]* Enumerate `CAPTURE` formats via `VIDIOC_ENUM_FMT()` on the `CAPTURE` queue. The client may use this ioctl to discover which alternative raw formats are supported for the current `OUTPUT` format and select one of them via `VIDIOC_S_FMT()`.

Note: The driver will return only formats supported for the currently selected `OUTPUT` format and currently set controls, even if more formats may be supported by the decoder in general.

For example, a decoder may support YUV and RGB formats for resolutions 1920x1088 and lower, but only YUV for higher resolutions (due to hardware limitations). After setting a resolution of 1920x1088 or lower as the `OUTPUT` format, `VIDIOC_ENUM_FMT()` may return a set of YUV and RGB pixel formats, but after setting a resolution higher than 1920x1088, the driver will not return RGB pixel formats, since they are unsupported for this resolution.

---

5. *[optional]* Choose a different `CAPTURE` format than suggested via `VIDIOC_S_FMT()` on `CAPTURE` queue. It is possible for the client to choose a different format than selected/suggested by the driver in `VIDIOC_G_FMT()`.

   - **Required fields:**

     `type`
     a `V4L2_BUF_TYPE_*` enum appropriate for `CAPTURE`.

     `pixelformat`
     a raw pixel format.

     `width`, `height`
     frame buffer resolution of the decoded stream; typically unchanged from what was returned with `VIDIOC_G_FMT()`, but it may be different if the hardware supports composition and/or scaling.

   After performing this step, the client must perform step 3 again in order to obtain up-to-date information about the buffers size and layout.

6. Allocate source (bytestream) buffers via `VIDIOC_REQBUFS()` on `OUTPUT` queue.

   - **Required fields:**

     `count`
     requested number of buffers to allocate; greater than zero.

     `type`
     a `V4L2_BUF_TYPE_*` enum appropriate for `OUTPUT`.

     `memory`
     follows standard semantics.

- **Return fields:**

  count
  actual number of buffers allocated.

- If required, the driver will adjust count to be equal or bigger to the minimum of required number of OUTPUT buffers for the given format and requested count. The client must check this value after the ioctl returns to get the actual number of buffers allocated.

7. Allocate destination (raw format) buffers via VIDIOC_REQBUFS() on the CAPTURE queue.

   - **Required fields:**

     count
     requested number of buffers to allocate; greater than zero. The client is responsible for deducing the minimum number of buffers required for the stream to be properly decoded (taking e.g. reference frames into account) and pass an equal or bigger number.

     type
     a V4L2_BUF_TYPE_* enum appropriate for CAPTURE.

     memory
     follows standard semantics. V4L2_MEMORY_USERPTR is not supported for CAPTURE buffers.

   - **Return fields:**

     count
     adjusted to allocated number of buffers, in case the codec requires more buffers than requested.

   - The driver must adjust count to the minimum of required number of CAPTURE buffers for the current format, stream configuration and requested count. The client must check this value after the ioctl returns to get the number of buffers allocated.

8. **Allocate requests (likely one per OUTPUT buffer) via**
   MEDIA_IOC_REQUEST_ALLOC() on the media device.

9. **Start streaming on both OUTPUT and CAPTURE queues via**
   VIDIOC_STREAMON().

Decoding

For each frame, the client is responsible for submitting at least one request to which the following is attached:

- The amount of encoded data expected by the codec for its current configuration, as a buffer submitted to the OUTPUT queue. Typically, this corresponds to one frame worth of encoded data, but some formats may allow (or require) different amounts per unit.
- All the metadata needed to decode the submitted encoded data, in the form of controls relevant to the format being decoded.

The amount of data and contents of the source OUTPUT buffer, as well as the controls that must be set on the request, depend on the active coded pixel format and might be affected by codec-specific extended controls, as stated in documentation of each format.

If there is a possibility that the decoded frame will require one or more decode requests after the current one in order to be produced, then the client must set the V4L2_BUF_FLAG_M2M_HOLD_CAPTURE_BUF flag on the OUTPUT buffer. This will result in the (potentially partially) decoded CAPTURE buffer not being made available for dequeueing, and reused for the next decode request if the timestamp of the next OUTPUT buffer has not changed.

A typical frame would thus be decoded using the following sequence:

1. Queue an OUTPUT buffer containing one unit of encoded bytestream data for the decoding request, using VIDIOC_QBUF().

   - **Required fields:**

     index
     index of the buffer being queued.

     type
     type of the buffer.

     bytesused
     number of bytes taken by the encoded data frame in the buffer.

     flags
     the V4L2_BUF_FLAG_REQUEST_FD flag must be set. Additionally, if we are not sure that the current decode request is the last one needed to produce a fully decoded frame, then V4L2_BUF_FLAG_M2M_HOLD_CAPTURE_BUF must also be set.

     request_fd
     must be set to the file descriptor of the decoding request.

     timestamp
     must be set to a unique value per frame. This value will be propagated into the decoded frame's buffer and can also be used to use this frame as the reference of another. If using multiple decode requests per frame, then the timestamps of all the OUTPUT buffers for a given frame must be identical. If the timestamp changes, then the currently held CAPTURE buffer will be made available for dequeuing and the current request will work on a new CAPTURE buffer.

2. Set the codec-specific controls for the decoding request, using VIDIOC_S_EXT_CTRLS().

   - **Required fields:**

     which
     must be V4L2_CTRL_WHICH_REQUEST_VAL.

     request_fd
     must be set to the file descriptor of the decoding request.

     other fields
     other fields are set as usual when setting controls. The controls

array must contain all the codec-specific controls required to decode a frame.

Note: It is possible to specify the controls in different invocations of VIDIOC_S_EXT_CTRLS(), or to overwrite a previously set control, as long as request_fd and which are properly set. The controls state at the moment of request submission is the one that will be considered.

Note: The order in which steps 1 and 2 take place is interchangeable.

3. Submit the request by invoking MEDIA_REQUEST_IOC_QUEUE() on the request FD.

   If the request is submitted without an OUTPUT buffer, or if some of the required controls are missing from the request, then MEDIA_REQUEST_IOC_QUEUE() will return -ENOENT. If more than one OUTPUT buffer is queued, then it will return -EINVAL. MEDIA_REQUEST_IOC_QUEUE() returning non-zero means that no CAPTURE buffer will be produced for this request.

CAPTURE buffers must not be part of the request, and are queued independently. They are returned in decode order (i.e. the same order as coded frames were submitted to the OUTPUT queue).

Runtime decoding errors are signaled by the dequeued CAPTURE buffers carrying the V4L2_BUF_FLAG_ERROR flag. If a decoded reference frame has an error, then all following decoded frames that refer to it also have the V4L2_BUF_FLAG_ERROR flag set, although the decoder will still try to produce (likely corrupted) frames.

Buffer management while decoding

Contrary to stateful decoders, a stateless decoder does not perform any kind of buffer management: it only guarantees that dequeued CAPTURE buffers can be used by the client for as long as they are not queued again. "Used" here encompasses using the buffer for compositing or display.

A dequeued capture buffer can also be used as the reference frame of another buffer.

A frame is specified as reference by converting its timestamp into nanoseconds, and storing it into the relevant member of a codec-dependent control structure. The v4l2_timeval_to_ns() function must be used to perform that conversion. The timestamp of a frame can be used to reference it as soon as all its units of encoded data are successfully submitted to the OUTPUT queue.

A decoded buffer containing a reference frame must not be reused as a decoding target until all the frames referencing it have been decoded. The safest way to achieve this is to refrain from queueing a reference buffer until all the decoded frames referencing it have been dequeued. However, if the driver can guarantee that buffers queued to the CAPTURE queue are processed in queued order, then user-space can take advantage of this guarantee and queue a reference buffer when the following conditions are met:

1. All the requests for frames affected by the reference frame have been queued, and
2. A sufficient number of CAPTURE buffers to cover all the decoded referencing frames have been queued.

When queuing a decoding request, the driver will increase the reference count of all the resources associated with reference frames. This means that the client can e.g. close the DMABUF file descriptors of reference frame buffers if it won't need them afterwards.

Seeking

In order to seek, the client just needs to submit requests using input buffers corresponding to the new stream position. It must however be aware that resolution may have changed and follow the dynamic resolution change sequence in that case. Also depending on the codec used, picture parameters (e.g. SPS/PPS for H.264) may have changed and the client is responsible for making sure that a valid state is sent to the decoder.

The client is then free to ignore any returned CAPTURE buffer that comes from the pre-seek position.

Pausing

In order to pause, the client can just cease queuing buffers onto the OUTPUT queue. Without source bytestream data, there is no data to process and the codec will remain idle.

Dynamic resolution change

If the client detects a resolution change in the stream, it will need to perform the initialization sequence again with the new resolution:

1. If the last submitted request resulted in a CAPTURE buffer being held by the use of the V4L2_BUF_FLAG_M2M_HOLD_CAPTURE_BUF flag, then the last frame is not available on the CAPTURE queue. In this case, a V4L2_DEC_CMD_FLUSH command shall be sent. This will make the driver dequeue the held CAPTURE buffer.
2. Wait until all submitted requests have completed and dequeue the corresponding output buffers.
3. Call VIDIOC_STREAMOFF() on both the OUTPUT and CAPTURE queues.
4. Free all CAPTURE buffers by calling VIDIOC_REQBUFS() on the CAPTURE queue with a buffer count of zero.
5. Perform the initialization sequence again (minus the allocation of OUTPUT buffers), with the new resolution set on the OUTPUT queue. Note that due

to resolution constraints, a different format may need to be picked on the CAPTURE queue.

Drain

If the last submitted request resulted in a CAPTURE buffer being held by the use of the V4L2_BUF_FLAG_M2M_HOLD_CAPTURE_BUF flag, then the last frame is not available on the CAPTURE queue. In this case, a V4L2_DEC_CMD_FLUSH command shall be sent. This will make the driver dequeue the held CAPTURE buffer.

After that, in order to drain the stream on a stateless decoder, the client just needs to wait until all the submitted requests are completed.

Raw VBI Data Interface

VBI is an abbreviation of Vertical Blanking Interval, a gap in the sequence of lines of an analog video signal. During VBI no picture information is transmitted, allowing some time while the electron beam of a cathode ray tube TV returns to the top of the screen. Using an oscilloscope you will find here the vertical synchronization pulses and short data packages ASK modulated[1] onto the video signal. These are transmissions of services such as Teletext or Closed Caption.

Subject of this interface type is raw VBI data, as sampled off a video signal, or to be added to a signal for output. The data format is similar to uncompressed video images, a number of lines times a number of samples per line, we call this a VBI image.

Conventionally V4L2 VBI devices are accessed through character device special files named /dev/vbi and /dev/vbi0 to /dev/vbi31 with major number 81 and minor numbers 224 to 255. /dev/vbi is typically a symbolic link to the preferred VBI device. This convention applies to both input and output devices.

To address the problems of finding related video and VBI devices VBI capturing and output is also available as device function under /dev/video. To capture or output raw VBI data with these devices applications must call the *VIDIOC_S_FMT* ioctl. Accessed as /dev/vbi, raw VBI capturing or output is the default device function.

Querying Capabilities

Devices supporting the raw VBI capturing or output API set the V4L2_CAP_VBI_CAPTURE or V4L2_CAP_VBI_OUTPUT flags, respectively, in the capabilities field of struct *v4l2_capability* returned by the *ioctl VIDIOC_QUERYCAP* ioctl. At least one of the read/write, streaming or asynchronous I/O methods must be supported. VBI devices may or may not have a tuner or modulator.

[1] ASK: Amplitude-Shift Keying. A high signal level represents a ‘1’ bit, a low level a ‘0’ bit.

Supplemental Functions

VBI devices shall support *video input or output*, *tuner or modulator*, and *controls* ioctls as needed. The *video standard* ioctls provide information vital to program a VBI device, therefore must be supported.

Raw VBI Format Negotiation

Raw VBI sampling abilities can vary, in particular the sampling frequency. To properly interpret the data V4L2 specifies an ioctl to query the sampling parameters. Moreover, to allow for some flexibility applications can also suggest different parameters.

As usual these parameters are *not* reset at `open()` time to permit Unix tool chains, programming a device and then reading from it as if it was a plain file. Well written V4L2 applications should always ensure they really get what they want, requesting reasonable parameters and then checking if the actual parameters are suitable.

To query the current raw VBI capture parameters applications set the `type` field of a struct `v4l2_format` to `V4L2_BUF_TYPE_VBI_CAPTURE` or `V4L2_BUF_TYPE_VBI_OUTPUT`, and call the *VIDIOC_G_FMT* ioctl with a pointer to this structure. Drivers fill the struct `v4l2_vbi_format` `vbi` member of the `fmt` union.

To request different parameters applications set the `type` field of a struct `v4l2_format` as above and initialize all fields of the struct `v4l2_vbi_format` `vbi` member of the `fmt` union, or better just modify the results of *VIDIOC_G_FMT*, and call the *VIDIOC_S_FMT* ioctl with a pointer to this structure. Drivers return an `EINVAL` error code only when the given parameters are ambiguous, otherwise they modify the parameters according to the hardware capabilities and return the actual parameters. When the driver allocates resources at this point, it may return an `EBUSY` error code to indicate the returned parameters are valid but the required resources are currently not available. That may happen for instance when the video and VBI areas to capture would overlap, or when the driver supports multiple opens and another process already requested VBI capturing or output. Anyway, applications must expect other resource allocation points which may return `EBUSY`, at the *ioctl VIDIOC_STREAMON, VIDIOC_STREAMOFF* ioctl and the first `read()` , `write()` and `select()` calls.

VBI devices must implement both the *VIDIOC_G_FMT* and *VIDIOC_S_FMT* ioctl, even if *VIDIOC_S_FMT* ignores all requests and always returns default parameters as *VIDIOC_G_FMT* does. *VIDIOC_TRY_FMT* is optional.

type **`v4l2_vbi_format`**

Table 82: struct v4l2_vbi_format

__u32	`sampling_rate`	Samples per second, i. e. unit 1 Hz.

continues on next page

Table 82 – continued from previous page

__u32	offset	Horizontal offset of the VBI image, relative to the leading edge of the line synchronization pulse and counted in samples: The first sample in the VBI image will be located offset / sampling_rate seconds following the leading edge. See also *Figure 4.1. Line synchronization*.
__u32	samples_per_line	
__u32	sample_format	Defines the sample format as in *Image Formats*, a four-character-code.[2] Usually this is V4L2_PIX_FMT_GREY, i. e. each sample consists of 8 bits with lower values oriented towards the black level. Do not assume any other correlation of values with the signal level. For example, the MSB does not necessarily indicate if the signal is 'high' or 'low' because 128 may not be the mean value of the signal. Drivers shall not convert the sample format by software.
__u32	start[Page 463, 2]	This is the scanning system line number associated with the first line of the VBI image, of the first and the second field respectively. See *Figure 4.2. ITU-R 525 line numbering (M/NTSC and M/PAL)* and *Figure 4.3. ITU-R 625 line numbering* for valid values. The V4L2_VBI_ITU_525_F1_START, V4L2_VBI_ITU_525_F2_START, V4L2_VBI_ITU_625_F1_STAR and V4L2_VBI_ITU_625_F2_START defines give the start line numbers for each field for each 525 or 625 line format as a convenience. Don't forget that ITU line numbering starts at 1, not 0. VBI input drivers can return start values 0 i the hardware cannot reliable identify scanning lines, VB acquisition may not require this information.
__u32	count[2]	The number of lines in the first and second field image, respectively.

Drivers should be as flexibility as possible. For example, it may be possible to extend or move the VBI capture window down to the picture area, implementing a 'full field mode' to capture data service transmissions embedded in the picture.
An application can set the first or second count value to zero if no data is required from the respective field; count[1] if the scanning system is progressive, i. e. not interlaced. The corresponding start value shall be ignored by the application and driver. Anyway, drivers may not support single field capturing and return both count values non-zero.
Both count values set to zero, or line numbers are outside the bounds depicted[4], or a field image covering lines of two fields, are invalid and shall not be returned by the driver.
To initialize the start and count fields, applications must first determine the current video standard selection. The *v4l2_std_id* or the framelines field of struct *v4l2_standard* can be evaluated for this purpose.

__u32	flags	See *Raw VBI Format Flags* below. Currently only drivers set flags, applications must set this field to zero.
__u32	reserved[2]	This array is reserved for future extensions. Drivers and applications must set it to zero.

[2] A few devices may be unable to sample VBI data at all but can extend the video capture window to the VBI region.

[4] The valid values ar shown at *Figure 4.2. ITU-R 525 line numbering (M/NTSC and M/PAL)* and *Figure 4.3. ITU-R 625 line numbering*.

Table 83: Raw VBI Format Flags

V4L2_VBI_UNSYNC	0x0001	This flag indicates hardware which does not properly dis tinguish between fields. Normally the VBI image stores the first field (lower scanning line numbers) first in memory This may be a top or bottom field depending on the videc standard. When this flag is set the first or second field may be stored first, however the fields are still in correct tempo ral order with the older field first in memory.[3]
V4L2_VBI_INTERLACED	0x0002	By default the two field images will be passed sequen tially; all lines of the first field followed by all lines o the second field (compare *Field Order* V4L2_FIELD_SEQ_TE and V4L2_FIELD_SEQ_BT, whether the top or bottom fielc is first in memory depends on the video standard) When this flag is set, the two fields are interlaced (cf V4L2_FIELD_INTERLACED). The first line of the first field fol lowed by the first line of the second field, then the two sec ond lines, and so on. Such a layout may be necessary wher the hardware has been programmed to capture or outpu interlaced video images and is unable to separate the fields for VBI capturing at the same time. For simplicity setting this flag implies that both count values are equal and non zero.

Remember the VBI image format depends on the selected video standard, therefore the application must choose a new standard or query the current standard first. Attempts to read or write data ahead of format negotiation, or after switching the video standard which may invalidate the negotiated VBI parameters, should be refused by the driver. A format change during active I/O is not permitted.

Reading and writing VBI images

To assure synchronization with the field number and easier implementation, the smallest unit of data passed at a time is one frame, consisting of two fields of VBI images immediately following in memory.

The total size of a frame computes as follows:

```
(count[0] + count[1]) * samples_per_line * sample size in bytes
```

The sample size is most likely always one byte, applications must check the `sample_format` field though, to function properly with other drivers.

A VBI device may support *read/write* and/or streaming (*memory mapping* or *user pointer*) I/O. The latter bears the possibility of synchronizing video and VBI data by using buffer timestamps.

Remember the *VIDIOC_STREAMON* ioctl and the first *read()*, *write()* and *select()* call can be resource allocation points returning an EBUSY error code

[3] Most VBI services transmit on both fields, but some have different semantics depending on the field number. These cannot be reliable decoded or encoded when V4L2_VBI_UNSYNC is set.

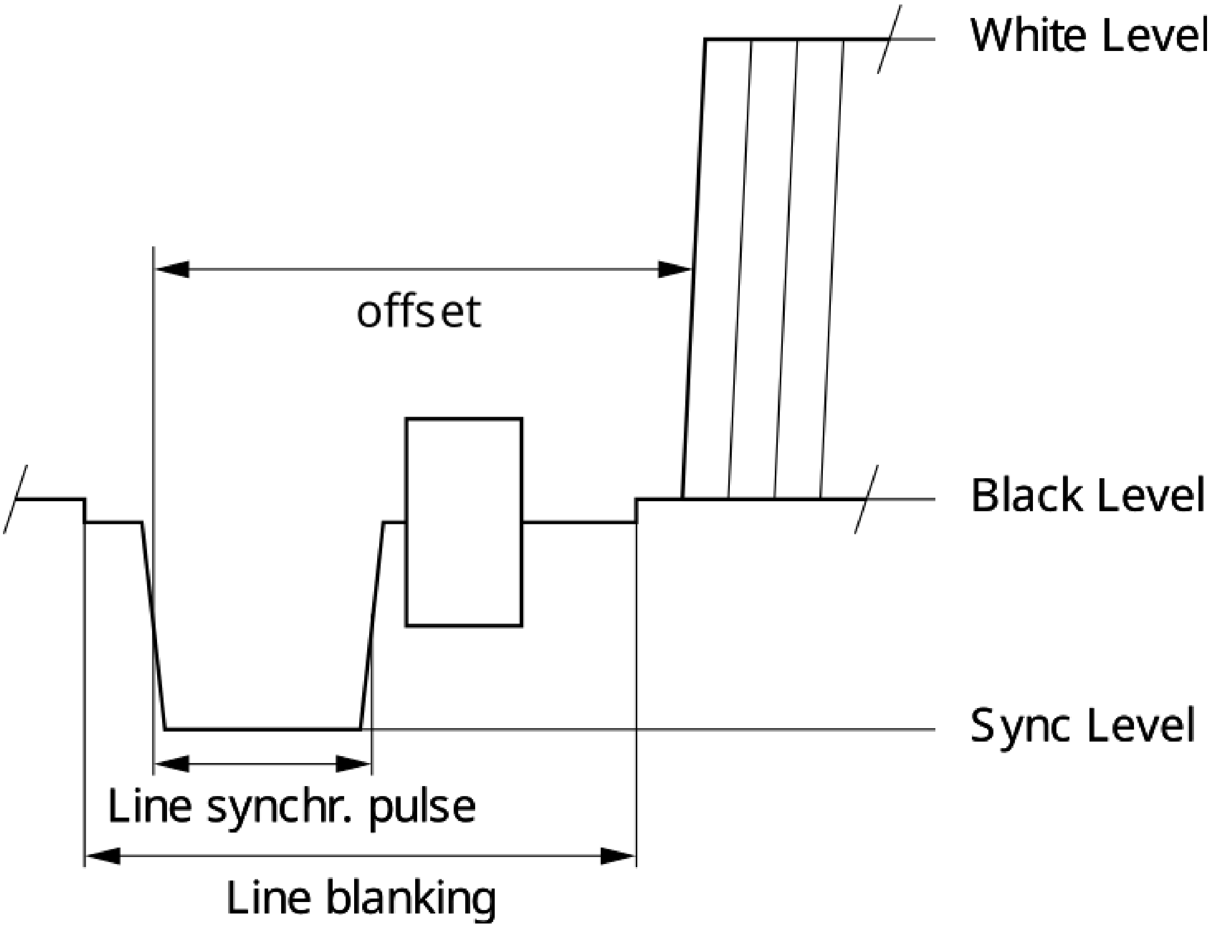

Fig. 10: **Figure 4.1. Line synchronization**

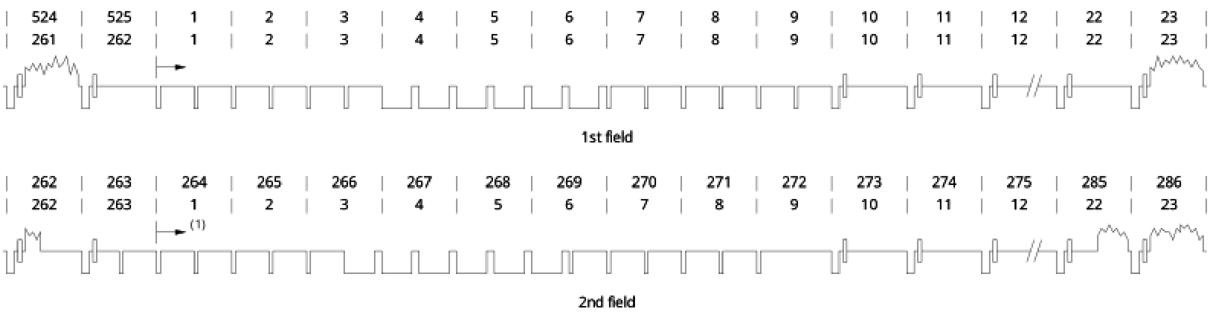

Fig. 11: **Figure 4.2. ITU-R 525 line numbering (M/NTSC and M/PAL)**

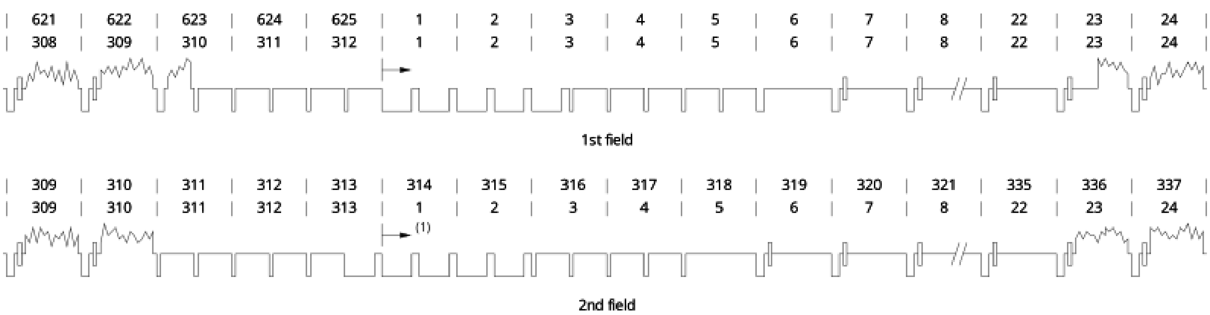

Fig. 12: **Figure 4.3. ITU-R 625 line numbering**

if the required hardware resources are temporarily unavailable, for example the device is already in use by another process.

Sliced VBI Data Interface

VBI stands for Vertical Blanking Interval, a gap in the sequence of lines of an analog video signal. During VBI no picture information is transmitted, allowing some time while the electron beam of a cathode ray tube TV returns to the top of the screen.

Sliced VBI devices use hardware to demodulate data transmitted in the VBI. V4L2 drivers shall *not* do this by software, see also the *raw VBI interface*. The data is passed as short packets of fixed size, covering one scan line each. The number of packets per video frame is variable.

Sliced VBI capture and output devices are accessed through the same character special files as raw VBI devices. When a driver supports both interfaces, the default function of a `/dev/vbi` device is *raw* VBI capturing or output, and the sliced VBI function is only available after calling the *VIDIOC_S_FMT* ioctl as defined below. Likewise a `/dev/video` device may support the sliced VBI API, however the default function here is video capturing or output. Different file descriptors must be used to pass raw and sliced VBI data simultaneously, if this is supported by the driver.

Querying Capabilities

Devices supporting the sliced VBI capturing or output API set the `V4L2_CAP_SLICED_VBI_CAPTURE` or `V4L2_CAP_SLICED_VBI_OUTPUT` flag respectively, in the `capabilities` field of struct *`v4l2_capability`* returned by the *ioctl VIDIOC_QUERYCAP* ioctl. At least one of the read/write, streaming or asynchronous *I/O methods* must be supported. Sliced VBI devices may have a tuner or modulator.

Supplemental Functions

Sliced VBI devices shall support *video input or output* and *tuner or modulator* ioctls if they have these capabilities, and they may support *User Controls* ioctls. The *video standard* ioctls provide information vital to program a sliced VBI device, therefore must be supported.

Sliced VBI Format Negotiation

To find out which data services are supported by the hardware applications can call the *VIDIOC_G_SLICED_VBI_CAP* ioctl. All drivers implementing the sliced VBI interface must support this ioctl. The results may differ from those of the *VIDIOC_S_FMT* ioctl when the number of VBI lines the hardware can capture or output per frame, or the number of services it can identify on a given line are limited. For example on PAL line 16 the hardware may be able to look for a VPS or Teletext signal, but not both at the same time.

To determine the currently selected services applications set the `type` field of struct *`v4l2_format`* to `V4L2_BUF_TYPE_SLICED_VBI_CAPTURE` or `V4L2_BUF_TYPE_SLICED_VBI_OUTPUT`, and the *VIDIOC_G_FMT* ioctl fills the `fmt.sliced` member, a struct *`v4l2_sliced_vbi_format`*.

Applications can request different parameters by initializing or modifying the `fmt.sliced` member and calling the *VIDIOC_S_FMT* ioctl with a pointer to the struct *`v4l2_format`* structure.

The sliced VBI API is more complicated than the raw VBI API because the hardware must be told which VBI service to expect on each scan line. Not all services may be supported by the hardware on all lines (this is especially true for VBI output where Teletext is often unsupported and other services can only be inserted in one specific line). In many cases, however, it is sufficient to just set the `service_set` field to the required services and let the driver fill the `service_lines` array according to hardware capabilities. Only if more precise control is needed should the programmer set the `service_lines` array explicitly.

The *VIDIOC_S_FMT* ioctl modifies the parameters according to hardware capabilities. When the driver allocates resources at this point, it may return an `EBUSY` error code if the required resources are temporarily unavailable. Other resource allocation points which may return `EBUSY` can be the *ioctl VIDIOC_STREAMON, VIDIOC_STREAMOFF* ioctl and the first *`read()`*, *`write()`* and *`select()`* call.

type **`v4l2_sliced_vbi_format`**

struct v4l2_sliced_vbi_format

__u16	service_set	If service_set is non-zero when passed with *VIDIOC_S_FMT* or *VIDIOC_TRY_FMT*, the service_lines array will be filled by the driver according to the services specified in this field. For example, if service_set is initialized with V4L2_SLICED_TELETEXT_B | V4L2_SLICED_WSS_625, a driver for the cx25840 video decoder sets lines 7-22 of both fields[1] to V4L2_SLICED_TELETEXT_B and line 23 of the first field to V4L2_SLICED_WSS_625. If service_set is set to zero, then the values of service_lines will be used instead. On return the driver sets this field to the union of all elements of the returned service_lines array. It may contain less services than requested, perhaps just one, if the hardware cannot handle more services simultaneously. It may be empty (zero) if none of the requested services are supported by the hardware.		
__u16	service_lines[2][24]	Applications initialize this array with sets of data services the driver shall look for or insert on the respective scan line. Subject to hardware capabilities drivers return the requested set, a subset, which may be just a single service, or an empty set. When the hardware cannot handle multiple services on the same line the driver shall choose one. No assumptions can be made on which service the driver chooses. Data services are defined in *Sliced VBI services*. Array indices map to ITU-R line numbers[2] as follows:		
		Element	525 line systems	625 line systems
		service_lines[0][1]	1	1
		service_lines[0][23]	23	23
		service_lines[1][1]	264	314
		service_lines[1][23]	286	336
		Drivers must set service_lines [0][0] and service_lines[1][0] to zero. The V4L2_VBI_ITU_525_F1_START, V4L2_VBI_ITU_525_F2_START, V4L2_VBI_ITU_625_F1_START and V4L2_VBI_ITU_625_F2_START defines give the start line numbers for each field for each 525 or 625 line format as a convenience. Don' t forget that ITU line numbering starts at 1, not 0.		
__u32	io_size	Maximum number of bytes passed by one *read()* or *write()* call, and the buffer size in bytes for the *ioctl VIDIOC_QBUF, VIDIOC_DQBUF* and *VIDIOC_DQBUF* ioctl. Drivers set this field to the size of struct *v4l2_sliced_vbi_data* times the number of non-zero elements in the returned service_lines array (that is the number of lines potentially carrying data).		
__u32	reserved[2]	This array is reserved for future extensions. Applications and drivers must set it to zero.		

Sliced VBI services

Symbol	Value	Reference	Lines, usually	Payload
V4L2_SLICED_TELETEXT_B (Teletext System B)	0x0001	*ETS 300 706*, *ITU BT.653*	PAL/SECAM line 7-22, 320-335 (second field 7-22)	Last 42 of the 45 byte Teletext packet, that is wi out clock run-in and framing code, lsb first tra mitted.
V4L2_SLICED_VPS	0x0400	*ETS 300 231*	PAL line 16	Byte number 3 to 15 according to Figure 9 of E 300 231, lsb first transmitted.
V4L2_SLICED_CAPTION_525	0x1000	*CEA 608-E*	NTSC line 21, 284 (second field 21)	Two bytes in transmission order, including par bit, lsb first transmitted.
V4L2_SLICED_WSS_625	0x4000	*ITU BT.1119*, *EN 300 294*	PAL/SECAM line 23	Byte 0 1 msb lsb msb lsb Bit 7 6 5 4 3 2 1 0 x x 13 12 11 10 9
V4L2_SLICED_VBI_525	0x1000	Set of services applicable to 525 line systems.		
V4L2_SLICED_VBI_625	0x4401	Set of services applicable to 625 line systems.		

[1] According to *ETS 300 706* lines 6-22 of the first field and lines 5-22 of the second field may carry Teletext data.

[2] See also *Figure 4.2. ITU-R 525 line numbering (M/NTSC and M/PAL)* and *Figure 4.3. ITU-R 625 line numbering*.

Drivers may return an EINVAL error code when applications attempt to read or write data without prior format negotiation, after switching the video standard (which may invalidate the negotiated VBI parameters) and after switching the video input (which may change the video standard as a side effect). The *VIDIOC_S_FMT* ioctl may return an EBUSY error code when applications attempt to change the format while i/o is in progress (between a *ioctl VIDIOC_STREAMON, VIDIOC_STREAMOFF* and *VIDIOC_STREAMOFF* call, and after the first *read()* or *write()* call).

Reading and writing sliced VBI data

A single *read()* or *write()* call must pass all data belonging to one video frame. That is an array of struct *v4l2_sliced_vbi_data* structures with one or more elements and a total size not exceeding io_size bytes. Likewise in streaming I/O mode one buffer of io_size bytes must contain data of one video frame. The id of unused struct *v4l2_sliced_vbi_data* elements must be zero.

type **v4l2_sliced_vbi_data**

struct v4l2_sliced_vbi_data

__u32	`id`	A flag from *Sliced VBI services* identifying the type of data in this packet. Only a single bit must be set. When the `id` of a captured packet is zero, the packet is empty and the contents of other fields are undefined. Applications shall ignore empty packets. When the `id` of a packet for output is zero the contents of the `data` field are undefined and the driver must no longer insert data on the requested `field` and `line`.
__u32	`field`	The video field number this data has been captured from, or shall be inserted at. `0` for the first field, `1` for the second field.
__u32	`line`	The field (as opposed to frame) line number this data has been captured from, or shall be inserted at. See *Figure 4.2. ITU-R 525 line numbering (M/NTSC and M/PAL)* and *Figure 4.3. ITU-R 625 line numbering* for valid values. Sliced VBI capture devices can set the line number of all packets to `0` if the hardware cannot reliably identify scan lines. The field number must always be valid.
__u32	`reserved`	This field is reserved for future extensions. Applications and drivers must set it to zero.
__u8	`data[48]`	The packet payload. See *Sliced VBI services* for the contents and number of bytes passed for each data type. The contents of padding bytes at the end of this array are undefined, drivers and applications shall ignore them.

Packets are always passed in ascending line number order, without duplicate line numbers. The *`write()`* function and the *ioctl VIDIOC_QBUF, VIDIOC_DQBUF* ioctl must return an `EINVAL` error code when applications violate this rule. They must also return an EINVAL error code when applications pass an incorrect field or line number, or a combination of `field`, `line` and `id` which has not been negotiated with the *VIDIOC_G_FMT* or *VIDIOC_S_FMT* ioctl. When the line numbers are unknown the driver must pass the packets in transmitted order. The driver can insert empty packets with `id` set to zero anywhere in the packet array.

To assure synchronization and to distinguish from frame dropping, when a captured frame does not carry any of the requested data services drivers must pass one or more empty packets. When an application fails to pass VBI data in time for output, the driver must output the last VPS and WSS packet again, and disable the output of Closed Caption and Teletext data, or output data which is ignored by Closed Caption and Teletext decoders.

A sliced VBI device may support *read/write* and/or streaming (*memory mapping* and/or *user pointer*) I/O. The latter bears the possibility of synchronizing video and VBI data by using buffer timestamps.

Sliced VBI Data in MPEG Streams

If a device can produce an MPEG output stream, it may be capable of providing *negotiated sliced VBI services* as data embedded in the MPEG stream. Users or applications control this sliced VBI data insertion with the *V4L2_CID_MPEG_STREAM_VBI_FMT* control.

If the driver does not provide the *V4L2_CID_MPEG_STREAM_VBI_FMT* control, or only allows that control to be set to *V4L2_MPEG_STREAM_VBI_FMT_NONE*, then the device cannot embed sliced VBI data in the MPEG stream.

The *V4L2_CID_MPEG_STREAM_VBI_FMT* control does not implicitly set the device driver to capture nor cease capturing sliced VBI data. The control only indicates to embed sliced VBI data in the MPEG stream, if an application has negotiated sliced VBI service be captured.

It may also be the case that a device can embed sliced VBI data in only certain types of MPEG streams: for example in an MPEG-2 PS but not an MPEG-2 TS. In this situation, if sliced VBI data insertion is requested, the sliced VBI data will be embedded in MPEG stream types when supported, and silently omitted from MPEG stream types where sliced VBI data insertion is not supported by the device.

The following subsections specify the format of the embedded sliced VBI data.

MPEG Stream Embedded, Sliced VBI Data Format: NONE

The *V4L2_MPEG_STREAM_VBI_FMT_NONE* embedded sliced VBI format shall be interpreted by drivers as a control to cease embedding sliced VBI data in MPEG streams. Neither the device nor driver shall insert "empty" embedded sliced VBI data packets in the MPEG stream when this format is set. No MPEG stream data structures are specified for this format.

MPEG Stream Embedded, Sliced VBI Data Format: IVTV

The *V4L2_MPEG_STREAM_VBI_FMT_IVTV* embedded sliced VBI format, when supported, indicates to the driver to embed up to 36 lines of sliced VBI data per frame in an MPEG-2 *Private Stream 1 PES* packet encapsulated in an MPEG-2 *Program Pack* in the MPEG stream.

Historical context: This format specification originates from a custom, embedded, sliced VBI data format used by the `ivtv` driver. This format has already been informally specified in the kernel sources in the file `Documentation/userspace-api/media/drivers/cx2341x-uapi.rst` . The maximum size of the payload and other aspects of this format are driven by the CX23415 MPEG decoder's capabilities and limitations with respect to extracting, decoding, and displaying sliced VBI data embedded within an MPEG stream.

This format's use is *not* exclusive to the `ivtv` driver *nor* exclusive to CX2341x devices, as the sliced VBI data packet insertion into the MPEG stream is implemented in driver software. At least the `cx18` driver provides sliced VBI data insertion into an MPEG-2 PS in this format as well.

The following definitions specify the payload of the MPEG-2 *Private Stream 1 PES* packets that contain sliced VBI data when *V4L2_MPEG_STREAM_VBI_FMT_IVTV* is set. (The MPEG-2 *Private Stream 1 PES* packet header and encapsulating MPEG-2 *Program Pack* header are not detailed here. Please refer to the MPEG-2 specifications for details on those packet headers.)

The payload of the MPEG-2 *Private Stream 1 PES* packets that contain sliced VBI data is specified by struct *v4l2_mpeg_vbi_fmt_ivtv*. The payload is variable length, depending on the actual number of lines of sliced VBI data present in a video frame. The payload may be padded at the end with unspecified fill bytes to align the end of the payload to a 4-byte boundary. The payload shall never exceed 1552 bytes (2 fields with 18 lines/field with 43 bytes of data/line and a 4 byte magic number).

type **v4l2_mpeg_vbi_fmt_ivtv**

struct v4l2_mpeg_vbi_fmt_ivtv

__u8	magic[4]	A "magic" constant from *Magic Constants for struct v4l2_mpeg_vbi_fmt_ivtv magic field* that indicates this is a valid sliced VBI data payload and also indicates which member of the anonymous union, itv0 or ITV0, to use for the payload data.
union {	(anonymous)	
struct *v4l2_r*	itv0	The primary form of the

Magic Constants for struct v4l2_mpeg_vbi_fmt_ivtv magic field

Defined Symbol	Value	Description
V4L2_MPEG_VBI_IVTV_MAGIC0	"itv0"	Indicates the itv0 member of the union in struct *v4l2_mpeg_vbi_fmt_ivtv* is valid.
V4L2_MPEG_VBI_IVTV_MAGIC1	"ITV0"	Indicates the ITV0 member of the union in struct *v4l2_mpeg_vbi_fmt_ivtv* is valid and that 36 lines of sliced VBI data are present.

type **v4l2_mpeg_vbi_itv0**

type **v4l2_mpeg_vbi_ITV0**

structs v4l2_mpeg_vbi_itv0 and v4l2_mpeg_vbi_ITV0

__le32	linemask[2]	Bitmasks indicating the VBI service lines present. These linemask values are stored in little endian byte order in the MPEG stream. Some reference linemask bit positions with their corresponding VBI line number and video field are given below. b_0 indicates the least significant bit of a linemask value:
		`linemask[0] b0:     line  6  first field`
		`linemask[0] b17:    line 23  first field`
		`linemask[0] b18:    line  6  second field`
		`linemask[0] b31:    line 19  second field`
		`linemask[1] b0:     line 20  second field`
		`linemask[1] b3:     line 23  second field`
		`linemask[1] b4-b31: unused and set to 0`

struct *v4l2_mpeg_vbi_itv0_line*	line[35]	This is a variable length array that holds from 1 to 35 lines of sliced VBI data. The sliced VBI data lines present correspond to the bits set in the linemask array, starting from b_0 of linemask[0] up through b_{31} of linemask[0], and from b_0 of linemask[1] up through b_3 of linemask[1]. line[0] corresponds to the first bit found set in the linemask array, line[1] corresponds to the second bit found set in the linemask array, etc. If no linemask array bits are set, then line[0] may contain one line of unspecified data that should be ignored by applications.

struct v4l2_mpeg_vbi_ITV0

struct *v4l2_mpeg_vbi_itv0_line*	line[36]	A fixed length array of 36 lines of sliced VBI data line[0] through line[17] correspond to lines 6 through 23 of the first field. line[18] through line[35] corresponds to lines 6 through 23 of the second field.

type **v4l2_mpeg_vbi_itv0_line**

struct v4l2_mpeg_vbi_itv0_line

__u8	id	A line identifier value from *Line Identifiers for struct v4l2_mpeg_vbi_itv0_line id field* that indicates the type of sliced VBI data stored on this line.
__u8	data[42]	The sliced VBI data for the line.

Line Identifiers for struct v4l2_mpeg_vbi_itv0_line id field

Defined Symbol	Value	Description
V4L2_MPEG_VBI_IVTV_TELETEXT_B	1	Refer to *Sliced VBI services* for a description of the line payload.
V4L2_MPEG_VBI_IVTV_CAPTION_525	4	Refer to *Sliced VBI services* for a description of the line payload.
V4L2_MPEG_VBI_IVTV_WSS_625	5	Refer to *Sliced VBI services* for a description of the line payload.
V4L2_MPEG_VBI_IVTV_VPS	7	Refer to *Sliced VBI services* for a description of the line payload.

Radio Interface

This interface is intended for AM and FM (analog) radio receivers and transmitters.

Conventionally V4L2 radio devices are accessed through character device special files named /dev/radio and /dev/radio0 to /dev/radio63 with major number 81 and minor numbers 64 to 127.

Querying Capabilities

Devices supporting the radio interface set the `V4L2_CAP_RADIO` and `V4L2_CAP_TUNER` or `V4L2_CAP_MODULATOR` flag in the `capabilities` field of struct `v4l2_capability` returned by the *ioctl VIDIOC_QUERYCAP* ioctl. Other combinations of capability flags are reserved for future extensions.

Supplemental Functions

Radio devices can support *controls*, and must support the *tuner or modulator* ioctls.

They do not support the video input or output, audio input or output, video standard, cropping and scaling, compression and streaming parameter, or overlay ioctls. All other ioctls and I/O methods are reserved for future extensions.

Programming

Radio devices may have a couple audio controls (as discussed in *User Controls*) such as a volume control, possibly custom controls. Further all radio devices have one tuner or modulator (these are discussed in *Tuners and Modulators*) with index number zero to select the radio frequency and to determine if a monaural or FM stereo program is received/emitted. Drivers switch automatically between AM and FM depending on the selected frequency. The *VIDIOC_G_TUNER* or *VIDIOC_G_MODULATOR* ioctl reports the supported frequency range.

RDS Interface

The Radio Data System transmits supplementary information in binary format, for example the station name or travel information, on an inaudible audio subcarrier of a radio program. This interface is aimed at devices capable of receiving and/or transmitting RDS information.

For more information see the core RDS standard *IEC 62106* and the RBDS standard *NRSC-4-B*.

Note: Note that the RBDS standard as is used in the USA is almost identical to the RDS standard. Any RDS decoder/encoder can also handle RBDS. Only some of the fields have slightly different meanings. See the RBDS standard for more information.

The RBDS standard also specifies support for MMBS (Modified Mobile Search). This is a proprietary format which seems to be discontinued. The RDS interface does not support this format. Should support for MMBS (or the so-called 'E blocks' in general) be needed, then please contact the linux-media mailing list: https://linuxtv.org/lists.php.

Querying Capabilities

Devices supporting the RDS capturing API set the V4L2_CAP_RDS_CAPTURE flag in the capabilities field of struct *v4l2_capability* returned by the *ioctl VIDIOC_QUERYCAP* ioctl. Any tuner that supports RDS will set the V4L2_TUNER_CAP_RDS flag in the capability field of struct *v4l2_tuner*. If the driver only passes RDS blocks without interpreting the data the V4L2_TUNER_CAP_RDS_BLOCK_IO flag has to be set, see *Reading RDS data*. For future use the flag V4L2_TUNER_CAP_RDS_CONTROLS has also been defined. However, a driver for a radio tuner with this capability does not yet exist, so if you are planning to write such a driver you should discuss this on the linux-media mailing list: https://linuxtv.org/lists.php.

Whether an RDS signal is present can be detected by looking at the rxsubchans field of struct *v4l2_tuner*: the V4L2_TUNER_SUB_RDS will be set if RDS data was detected.

Devices supporting the RDS output API set the V4L2_CAP_RDS_OUTPUT flag in the capabilities field of struct *v4l2_capability* returned by the *ioctl VIDIOC_QUERYCAP* ioctl. Any modulator that supports RDS will set the V4L2_TUNER_CAP_RDS flag in the capability field of struct *v4l2_modulator*. In order to enable the RDS transmission one must set the V4L2_TUNER_SUB_RDS bit in the txsubchans field of struct *v4l2_modulator*. If the driver only passes RDS blocks without interpreting the data the V4L2_TUNER_CAP_RDS_BLOCK_IO flag has to be set. If the tuner is capable of handling RDS entities like program identification codes and radio text, the flag V4L2_TUNER_CAP_RDS_CONTROLS should be set, see *Writing RDS data* and *FM Transmitter Control Reference*.

Reading RDS data

RDS data can be read from the radio device with the *read()* function. The data is packed in groups of three bytes.

Writing RDS data

RDS data can be written to the radio device with the *write()* function. The data is packed in groups of three bytes, as follows:

RDS datastructures

type **v4l2_rds_data**

Table 85: struct v4l2_rds_data

__u8	lsb	Least Significant Byte of RDS Block
__u8	msb	Most Significant Byte of RDS Block
__u8	block	Block description

Table 86: Block description

Bits 0-2	Block (aka offset) of the received data.
Bits 3-5	Deprecated. Currently identical to bits 0-2. Do not use these bits.
Bit 6	Corrected bit. Indicates that an error was corrected for this data block.
Bit 7	Error bit. Indicates that an uncorrectable error occurred during reception of this block.

Table 87: Block defines

V4L2_RDS_BLOCK_MSK		7	Mask for bits 0-2 to get the block ID.
V4L2_RDS_BLOCK_A		0	Block A.
V4L2_RDS_BLOCK_B		1	Block B.
V4L2_RDS_BLOCK_C		2	Block C.
V4L2_RDS_BLOCK_D		3	Block D.
V4L2_RDS_BLOCK_C_ALT		4	Block C' .
V4L2_RDS_BLOCK_INVALID	read-only	7	An invalid block.
V4L2_RDS_BLOCK_CORRECTED	read-only	0x40	A bit error was detected but correcte
V4L2_RDS_BLOCK_ERROR	read-only	0x80	An uncorrectable error occurred.

Software Defined Radio Interface (SDR)

SDR is an abbreviation of Software Defined Radio, the radio device which uses application software for modulation or demodulation. This interface is intended for controlling and data streaming of such devices.

SDR devices are accessed through character device special files named `/dev/swradio0` to `/dev/swradio255` with major number 81 and dynamically allocated minor numbers 0 to 255.

Querying Capabilities

Devices supporting the SDR receiver interface set the `V4L2_CAP_SDR_CAPTURE` and `V4L2_CAP_TUNER` flag in the `capabilities` field of struct `v4l2_capability` returned by the *ioctl VIDIOC_QUERYCAP* ioctl. That flag means the device has an Analog to Digital Converter (ADC), which is a mandatory element for the SDR receiver.

Devices supporting the SDR transmitter interface set the `V4L2_CAP_SDR_OUTPUT` and `V4L2_CAP_MODULATOR` flag in the `capabilities` field of struct `v4l2_capability` returned by the *ioctl VIDIOC_QUERYCAP* ioctl. That flag means the device has an Digital to Analog Converter (DAC), which is a mandatory element for the SDR transmitter.

At least one of the read/write, streaming or asynchronous I/O methods must be supported.

Supplemental Functions

SDR devices can support *controls*, and must support the *Tuners and Modulators* ioctls. Tuner ioctls are used for setting the ADC/DAC sampling rate (sampling frequency) and the possible radio frequency (RF).

The `V4L2_TUNER_SDR` tuner type is used for setting SDR device ADC/DAC frequency, and the `V4L2_TUNER_RF` tuner type is used for setting radio frequency. The tuner index of the RF tuner (if any) must always follow the SDR tuner index. Normally the SDR tuner is #0 and the RF tuner is #1.

The *ioctl VIDIOC_S_HW_FREQ_SEEK* ioctl is not supported.

Data Format Negotiation

The SDR device uses the *Data Formats* ioctls to select the capture and output format. Both the sampling resolution and the data streaming format are bound to that selectable format. In addition to the basic *Data Formats* ioctls, the *ioctl VIDIOC_ENUM_FMT* ioctl must be supported as well.

To use the *Data Formats* ioctls applications set the `type` field of a struct `v4l2_format` to `V4L2_BUF_TYPE_SDR_CAPTURE` or `V4L2_BUF_TYPE_SDR_OUTPUT` and use the struct *`v4l2_sdr_format`* `sdr` member of the `fmt` union as needed per the desired operation. Currently there are two fields, `pixelformat` and `buffersize`, of struct *`v4l2_sdr_format`* which are used. Content of the `pixelformat` is V4L2 FourCC code of the data format. The `buffersize` field is maximum buffer size in bytes required for data transfer, set by the driver in order to inform application.

type **`v4l2_sdr_format`**

Table 88: struct v4l2_sdr_format

__u32	`pixelformat`	The data format or type of compression, se by the application. This is a little endian *fou character code*. V4L2 defines SDR format in *SDR Formats*.
__u32	`buffersize`	Maximum size in bytes required for data Value is set by the driver.
__u8	`reserved[24]`	This array is reserved for future extension Drivers and applications must set it to zero.

An SDR device may support *read/write* and/or streaming (*memory mapping* or *user pointer*) I/O.

Touch Devices

Touch devices are accessed through character device special files named `/dev/v4l-touch0` to `/dev/v4l-touch255` with major number 81 and dynamically allocated minor numbers 0 to 255.

Overview

Sensors may be Optical, or Projected Capacitive touch (PCT).

Processing is required to analyse the raw data and produce input events. In some systems, this may be performed on the ASIC and the raw data is purely a side-channel for diagnostics or tuning. In other systems, the ASIC is a simple analogue front end device which delivers touch data at high rate, and any touch processing must be done on the host.

For capacitive touch sensing, the touchscreen is composed of an array of horizontal and vertical conductors (alternatively called rows/columns, X/Y lines, or tx/rx). Mutual Capacitance measured is at the nodes where the conductors cross. Alternatively, Self Capacitance measures the signal from each column and row independently.

A touch input may be determined by comparing the raw capacitance measurement to a no-touch reference (or "baseline") measurement:

Delta = Raw - Reference

The reference measurement takes account of variations in the capacitance across the touch sensor matrix, for example manufacturing irregularities, environmental or edge effects.

Querying Capabilities

Devices supporting the touch interface set the `V4L2_CAP_VIDEO_CAPTURE` flag and the `V4L2_CAP_TOUCH` flag in the `capabilities` field of `v4l2_capability` returned by the *ioctl VIDIOC_QUERYCAP* ioctl.

At least one of the read/write or streaming I/O methods must be supported.

The formats supported by touch devices are documented in *Touch Formats*.

Data Format Negotiation

A touch device may support any I/O method.

Event Interface

The V4L2 event interface provides a means for a user to get immediately notified on certain conditions taking place on a device. This might include start of frame or loss of signal events, for example. Changes in the value or state of a V4L2 control can also be reported through events.

To receive events, the events the user is interested in first must be subscribed using the *ioctl VIDIOC_SUBSCRIBE_EVENT, VIDIOC_UNSUBSCRIBE_EVENT* ioctl. Once an event is subscribed, the events of subscribed types are dequeueable using the *ioctl VIDIOC_DQEVENT* ioctl. Events may be unsubscribed using VIDIOC_UNSUBSCRIBE_EVENT ioctl. The special event type V4L2_EVENT_ALL may be used to unsubscribe all the events the driver supports.

The event subscriptions and event queues are specific to file handles. Subscribing an event on one file handle does not affect other file handles.

The information on dequeueable events is obtained by using select or poll system calls on video devices. The V4L2 events use POLLPRI events on poll system call and exceptions on select system call.

Starting with kernel 3.1 certain guarantees can be given with regards to events:

1. Each subscribed event has its own internal dedicated event queue. This means that flooding of one event type will not interfere with other event types.
2. If the internal event queue for a particular subscribed event becomes full, then the oldest event in that queue will be dropped.
3. Where applicable, certain event types can ensure that the payload of the oldest event that is about to be dropped will be merged with the payload of the next oldest event. Thus ensuring that no information is lost, but only an intermediate step leading up to that information. See the documentation for the event you want to subscribe to whether this is applicable for that event or not.

Sub-device Interface

The complex nature of V4L2 devices, where hardware is often made of several integrated circuits that need to interact with each other in a controlled way, leads to complex V4L2 drivers. The drivers usually reflect the hardware model in software, and model the different hardware components as software blocks called sub-devices.

V4L2 sub-devices are usually kernel-only objects. If the V4L2 driver implements the media device API, they will automatically inherit from media entities. Applications will be able to enumerate the sub-devices and discover the hardware topology using the media entities, pads and links enumeration API.

In addition to make sub-devices discoverable, drivers can also choose to make them directly configurable by applications. When both the sub-device driver and the V4L2 device driver support this, sub-devices will feature a character device node on which ioctls can be called to

- query, read and write sub-devices controls

- subscribe and unsubscribe to events and retrieve them
- negotiate image formats on individual pads

Sub-device character device nodes, conventionally named `/dev/v4l-subdev*`, use major number 81.

Drivers may opt to limit the sub-device character devices to only expose operations that do not modify the device state. In such a case the sub-devices are referred to as `read-only` in the rest of this documentation, and the related restrictions are documented in individual ioctls.

Controls

Most V4L2 controls are implemented by sub-device hardware. Drivers usually merge all controls and expose them through video device nodes. Applications can control all sub-devices through a single interface.

Complex devices sometimes implement the same control in different pieces of hardware. This situation is common in embedded platforms, where both sensors and image processing hardware implement identical functions, such as contrast adjustment, white balance or faulty pixels correction. As the V4L2 controls API doesn't support several identical controls in a single device, all but one of the identical controls are hidden.

Applications can access those hidden controls through the sub-device node with the V4L2 control API described in *User Controls*. The ioctls behave identically as when issued on V4L2 device nodes, with the exception that they deal only with controls implemented in the sub-device.

Depending on the driver, those controls might also be exposed through one (or several) V4L2 device nodes.

Events

V4L2 sub-devices can notify applications of events as described in *Event Interface*. The API behaves identically as when used on V4L2 device nodes, with the exception that it only deals with events generated by the sub-device. Depending on the driver, those events might also be reported on one (or several) V4L2 device nodes.

Pad-level Formats

Warning: Pad-level formats are only applicable to very complex devices that need to expose low-level format configuration to user space. Generic V4L2 applications do *not* need to use the API described in this section.

Note: For the purpose of this section, the term *format* means the combination of media bus data format, frame width and frame height.

Image formats are typically negotiated on video capture and output devices using the format and *selection* ioctls. The driver is responsible for configuring every block in the video pipeline according to the requested format at the pipeline input and/or output.

For complex devices, such as often found in embedded systems, identical image sizes at the output of a pipeline can be achieved using different hardware configurations. One such example is shown on *Image Format Negotiation on Pipelines*, where image scaling can be performed on both the video sensor and the host image processing hardware.

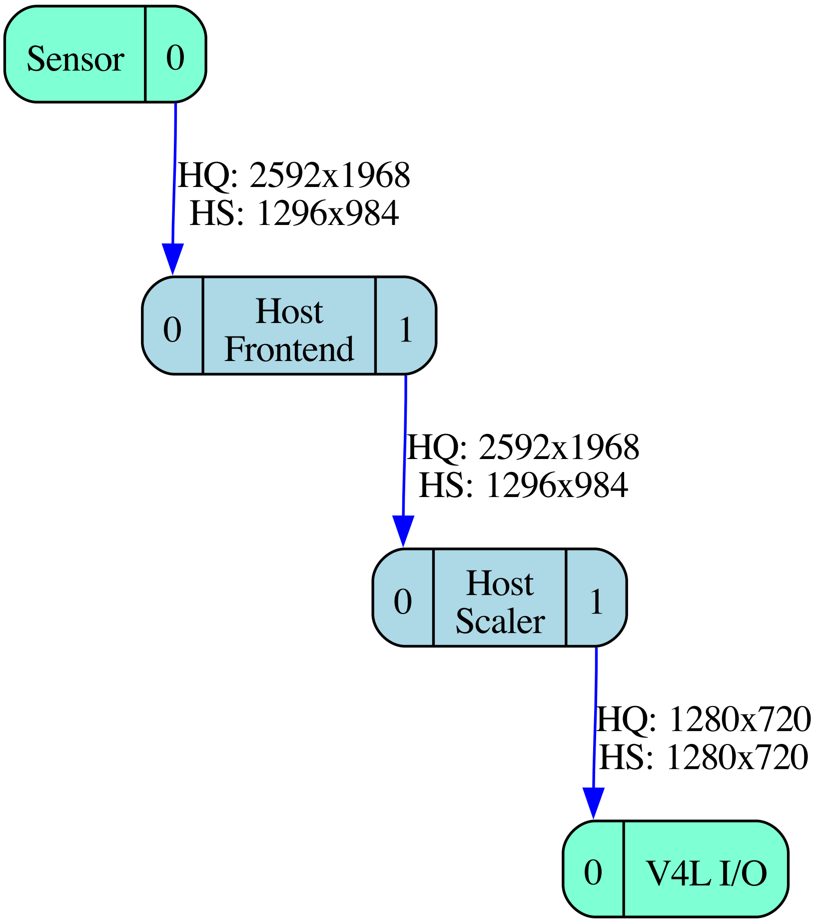

Fig. 13: Image Format Negotiation on Pipelines
High quality and high speed pipeline configuration

The sensor scaler is usually of less quality than the host scaler, but scaling on the sensor is required to achieve higher frame rates. Depending on the use case (quality vs. speed), the pipeline must be configured differently. Applications need to configure the formats at every point in the pipeline explicitly.

Drivers that implement the *media API* can expose pad-level image format configuration to applications. When they do, applications can use the *VID-*

IOC_SUBDEV_G_FMT and *VIDIOC_SUBDEV_S_FMT* ioctls. to negotiate formats on a per-pad basis.

Applications are responsible for configuring coherent parameters on the whole pipeline and making sure that connected pads have compatible formats. The pipeline is checked for formats mismatch at *VIDIOC_STREAMON* time, and an EPIPE error code is then returned if the configuration is invalid.

Pad-level image format configuration support can be tested by calling the *ioctl VIDIOC_SUBDEV_G_FMT, VIDIOC_SUBDEV_S_FMT* ioctl on pad 0. If the driver returns an EINVAL error code pad-level format configuration is not supported by the sub-device.

Format Negotiation

Acceptable formats on pads can (and usually do) depend on a number of external parameters, such as formats on other pads, active links, or even controls. Finding a combination of formats on all pads in a video pipeline, acceptable to both application and driver, can't rely on formats enumeration only. A format negotiation mechanism is required.

Central to the format negotiation mechanism are the get/set format operations. When called with the `which` argument set to *V4L2_SUBDEV_FORMAT_TRY*, the *VIDIOC_SUBDEV_G_FMT* and *VIDIOC_SUBDEV_S_FMT* ioctls operate on a set of formats parameters that are not connected to the hardware configuration. Modifying those 'try' formats leaves the device state untouched (this applies to both the software state stored in the driver and the hardware state stored in the device itself).

While not kept as part of the device state, try formats are stored in the sub-device file handles. A *VIDIOC_SUBDEV_G_FMT* call will return the last try format set *on the same sub-device file handle*. Several applications querying the same sub-device at the same time will thus not interact with each other.

To find out whether a particular format is supported by the device, applications use the *VIDIOC_SUBDEV_S_FMT* ioctl. Drivers verify and, if needed, change the requested `format` based on device requirements and return the possibly modified value. Applications can then choose to try a different format or accept the returned value and continue.

Formats returned by the driver during a negotiation iteration are guaranteed to be supported by the device. In particular, drivers guarantee that a returned format will not be further changed if passed to an *VIDIOC_SUBDEV_S_FMT* call as-is (as long as external parameters, such as formats on other pads or links' configuration are not changed).

Drivers automatically propagate formats inside sub-devices. When a try or active format is set on a pad, corresponding formats on other pads of the same sub-device can be modified by the driver. Drivers are free to modify formats as required by the device. However, they should comply with the following rules when possible:

- Formats should be propagated from sink pads to source pads. Modifying a format on a source pad should not modify the format on any sink pad.

- Sub-devices that scale frames using variable scaling factors should reset the scale factors to default values when sink pads formats are modified. If the 1:1 scaling ratio is supported, this means that source pads formats should be reset to the sink pads formats.

Formats are not propagated across links, as that would involve propagating them from one sub-device file handle to another. Applications must then take care to configure both ends of every link explicitly with compatible formats. Identical formats on the two ends of a link are guaranteed to be compatible. Drivers are free to accept different formats matching device requirements as being compatible.

Sample Pipeline Configuration shows a sample configuration sequence for the pipeline described in *Image Format Negotiation on Pipelines* (table columns list entity names and pad numbers).

Table 89: Sample Pipeline Configuration

	Sensor/0 format	Frontend/0 format	Frontend/1 format	Scaler/0 format	Scaler/0 compose selection rectangle	Scaler/1 format
Initial state	2048x1536 SGRBG8_1X8	(default)	(default)	(default)	(default)	(default)
Configure frontend sink format	2048x1536 SGRBG8_1X8	*2048x1536 SGRBG8_1X8*	*2046x1534 SGRBG8_1X8*	(default)	(default)	(default)
Configure scaler sink format	2048x1536 SGRBG8_1X8	2048x1536 SGRBG8_1X8	2046x1534 SGRBG8_1X8	*2046x1534 SGRBG8_1X8*	*0,0/2046x1534*	*2046x1534 SGRBG8_1X8*
Configure scaler sink compose selection	2048x1536 SGRBG8_1X8	2048x1536 SGRBG8_1X8	2046x1534 SGRBG8_1X8	2046x1534 SGRBG8_1X8	*0,0/1280x960*	*1280x960 SGRBG8_1X8*

1. Initial state. The sensor source pad format is set to its native 3MP size and V4L2_MBUS_FMT_SGRBG8_1X8 media bus code. Formats on the host frontend and scaler sink and source pads have the default values, as well as the compose rectangle on the scaler' s sink pad.
2. The application configures the frontend sink pad format' s size to 2048x1536 and its media bus code to V4L2_MBUS_FMT_SGRBG_1X8. The driver propagates the format to the frontend source pad.
3. The application configures the scaler sink pad format' s size to 2046x1534 and the media bus code to V4L2_MBUS_FMT_SGRBG_1X8 to match the frontend source size and media bus code. The media bus code on the sink pad is set to V4L2_MBUS_FMT_SGRBG_1X8. The driver propagates the size to the compose selection rectangle on the scaler' s sink pad, and the format to the scaler source pad.
4. The application configures the size of the compose selection rectangle of the scaler' s sink pad 1280x960. The driver propagates the size to the scaler' s source pad format.

When satisfied with the try results, applications can set the active formats by setting the `which` argument to `V4L2_SUBDEV_FORMAT_ACTIVE`. Active formats are changed exactly as try formats by drivers. To avoid modifying the hardware state during format negotiation, applications should negotiate try formats first and then modify the active settings using the try formats returned during the last negotiation iteration. This guarantees that the active format will be applied as-is by the driver without being modified.

Selections: cropping, scaling and composition

Many sub-devices support cropping frames on their input or output pads (or possible even on both). Cropping is used to select the area of interest in an image, typically on an image sensor or a video decoder. It can also be used as part of digital zoom implementations to select the area of the image that will be scaled up.

Crop settings are defined by a crop rectangle and represented in a struct *v4l2_rect* by the coordinates of the top left corner and the rectangle size. Both the coordinates and sizes are expressed in pixels.

As for pad formats, drivers store try and active rectangles for the selection targets *Common selection definitions*.

On sink pads, cropping is applied relative to the current pad format. The pad format represents the image size as received by the sub-device from the previous block in the pipeline, and the crop rectangle represents the sub-image that will be transmitted further inside the sub-device for processing.

The scaling operation changes the size of the image by scaling it to new dimensions. The scaling ratio isn't specified explicitly, but is implied from the original and scaled image sizes. Both sizes are represented by struct *v4l2_rect*.

Scaling support is optional. When supported by a subdev, the crop rectangle on the subdev's sink pad is scaled to the size configured using the *VIDIOC_SUBDEV_S_SELECTION* IOCTL using `V4L2_SEL_TGT_COMPOSE` selection target on the same pad. If the subdev supports scaling but not composing, the top and left values are not used and must always be set to zero.

On source pads, cropping is similar to sink pads, with the exception that the source size from which the cropping is performed, is the COMPOSE rectangle on the sink pad. In both sink and source pads, the crop rectangle must be entirely contained inside the source image size for the crop operation.

The drivers should always use the closest possible rectangle the user requests on all selection targets, unless specifically told otherwise. `V4L2_SEL_FLAG_GE` and `V4L2_SEL_FLAG_LE` flags may be used to round the image size either up or down. *Selection flags*

Types of selection targets

Actual targets

Actual targets (without a postfix) reflect the actual hardware configuration at any point of time. There is a BOUNDS target corresponding to every actual target.

BOUNDS targets

BOUNDS targets is the smallest rectangle that contains all valid actual rectangles. It may not be possible to set the actual rectangle as large as the BOUNDS rectangle, however. This may be because e.g. a sensor's pixel array is not rectangular but cross-shaped or round. The maximum size may also be smaller than the BOUNDS rectangle.

Order of configuration and format propagation

Inside subdevs, the order of image processing steps will always be from the sink pad towards the source pad. This is also reflected in the order in which the configuration must be performed by the user: the changes made will be propagated to any subsequent stages. If this behaviour is not desired, the user must set `V4L2_SEL_FLAG_KEEP_CONFIG` flag. This flag causes no propagation of the changes are allowed in any circumstances. This may also cause the accessed rectangle to be adjusted by the driver, depending on the properties of the underlying hardware.

The coordinates to a step always refer to the actual size of the previous step. The exception to this rule is the sink compose rectangle, which refers to the sink compose bounds rectangle —if it is supported by the hardware.

1. Sink pad format. The user configures the sink pad format. This format defines the parameters of the image the entity receives through the pad for further processing.
2. Sink pad actual crop selection. The sink pad crop defines the crop performed to the sink pad format.
3. Sink pad actual compose selection. The size of the sink pad compose rectangle defines the scaling ratio compared to the size of the sink pad crop rectangle. The location of the compose rectangle specifies the location of the actual sink compose rectangle in the sink compose bounds rectangle.
4. Source pad actual crop selection. Crop on the source pad defines crop performed to the image in the sink compose bounds rectangle.
5. Source pad format. The source pad format defines the output pixel format of the subdev, as well as the other parameters with the exception of the image width and height. Width and height are defined by the size of the source pad actual crop selection.

Accessing any of the above rectangles not supported by the subdev will return `EINVAL`. Any rectangle referring to a previous unsupported rectangle coordinates will instead refer to the previous supported rectangle. For example, if sink crop is not supported, the compose selection will refer to the sink pad format dimensions instead.

In the above example, the subdev supports cropping on its sink pad. To configure it, the user sets the media bus format on the subdev's sink pad. Now the actual crop rectangle can be set on the sink pad —the location and size of this rectangle reflect the location and size of a rectangle to be cropped from the sink format. The size of the sink crop rectangle will also be the size of the format of the subdev's source pad.

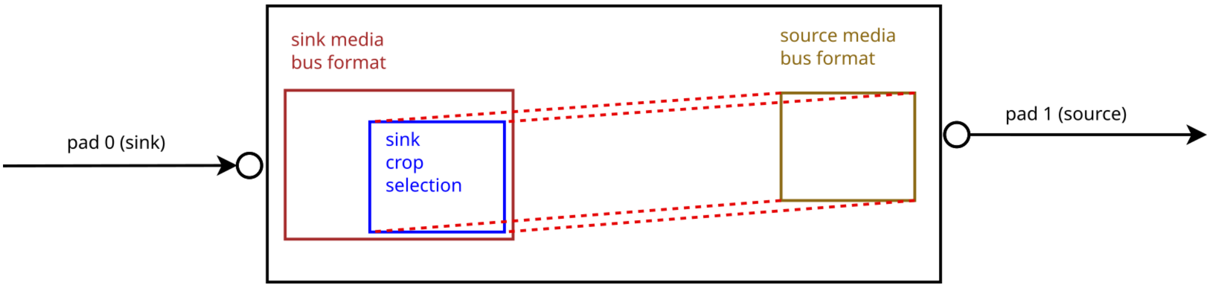

Fig. 14: **Figure 4.5. Image processing in subdevs: simple crop example**

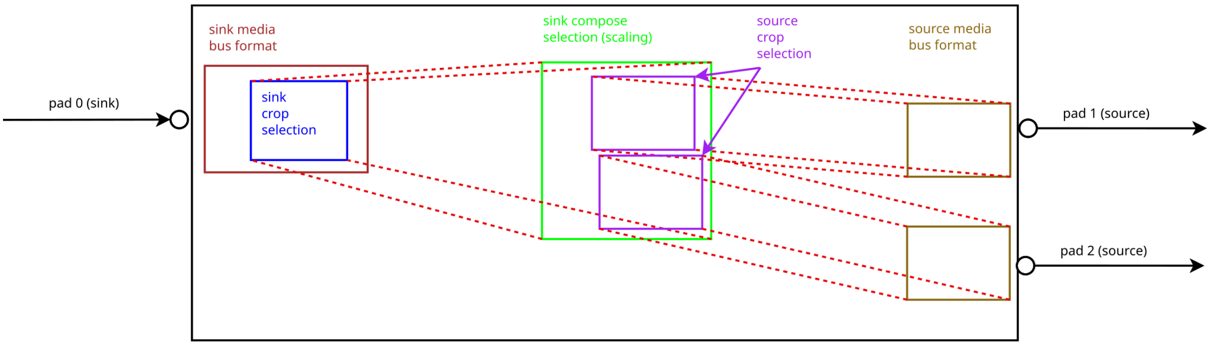

Fig. 15: **Figure 4.6. Image processing in subdevs: scaling with multiple sources**

In this example, the subdev is capable of first cropping, then scaling and finally cropping for two source pads individually from the resulting scaled image. The location of the scaled image in the cropped image is ignored in sink compose target. Both of the locations of the source crop rectangles refer to the sink scaling rectangle, independently cropping an area at location specified by the source crop rectangle from it.

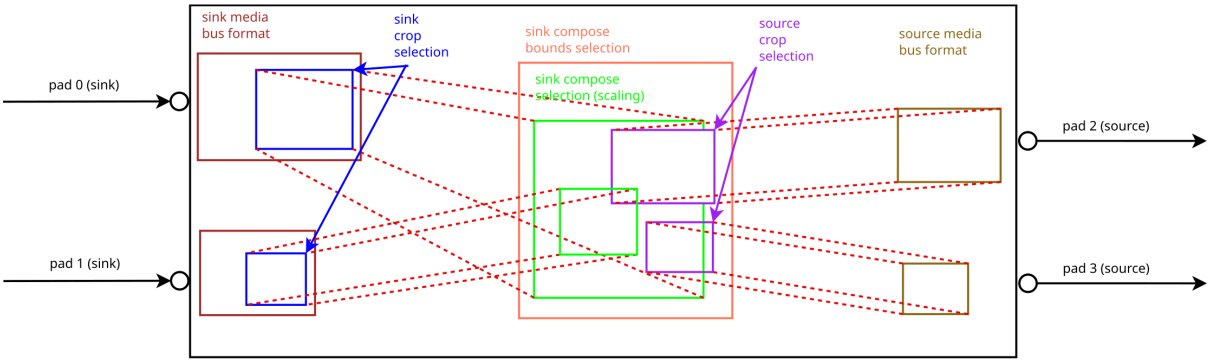

Fig. 16: **Figure 4.7. Image processing in subdevs: scaling and composition with multiple sinks and sources**

The subdev driver supports two sink pads and two source pads. The images from both of the sink pads are individually cropped, then scaled and further composed on the composition bounds rectangle. From that, two independent streams are cropped and sent out of the subdev from the source pads.

Media Bus Formats

type **v4l2_mbus_framefmt**

Table 90: struct v4l2_mbus_framefmt

__u32	`width`	Image width in pixels.
__u32	`height`	Image height in pixels. If `field` is one of `V4L2_FIELD_TOP`, `V4L2_FIELD_BOTTOM` or `V4L2_FIELD_ALTERNATE` then height refers to the number of lines in the field, otherwise it refers to the number of lines in the frame (which is twice the field height for interlaced formats).
__u32	`code`	Format code, from enum *v4l2_mbus_pixelcode*.
__u32	`field`	Field order, from enum *`v4l2_field`*. See *Field Order* for details.
__u32	`colorspace`	Image colorspace, from enum *`v4l2_colorspace`*. Must be set by the driver for subdevices. If the application sets the flag `V4L2_MBUS_FRAMEFMT_SET_CSC` then the application can set this field on the source pad to request a specific colorspace for the media bus data. If the driver cannot handle the requested conversion, it will return another supported colorspace. The driver indicates that colorspace conversion is supported by setting the flag V4L2_SUBDEV_MBUS_CODE_CSC_COLORS in the corresponding struct `v4l2_subdev_mbus_code_enum` during enumeration. See *Subdev Media Bus Code Enumerate Flags*.
union {	(anonymous)	
__u16	`ycbcr_enc`	Y' CbCr encoding, from enum *`v4l2_ycbcr_encoding`*. This information supplements the `colorspace` and must be set by the driver for subdevices, see *Colorspaces*. If the application sets the flag `V4L2_MBUS_FRAMEFMT_SET_CSC` then the application can set this field on a source pad to request a specific Y' CbCr encoding for the media bus data. If the driver cannot handle the requested conversion, it will return another supported encoding. This field is ignored for HSV media bus formats. The driver indicates that ycbcr_enc conversion is supported by setting the flag V4L2_SUBDEV_MBUS_CODE_CSC_YCBCR_ in the corresponding struct `v4l2_subdev_mbus_code_enum` during enumeration. See *Subdev Media Bus Code Enumerate Flags*.
__u16	`hsv_enc`	HSV encoding, from enum *`v4l2_hsv_encoding`*. This information supplements the `colorspace` and must be set by the driver for subdevices, see *Colorspaces*. If the application sets the flag `V4L2_MBUS_FRAMEFMT_SET_CSC` then the application can set this field on a source

Table 91: v4l2_mbus_framefmt Flags

V4L2	0x0(	Set by the application. It is only used for source pads and is ignored for sink pads. If set, then request the subdevice to do colorspace conversion from the received colorspace to the requested colorspace values. If the colorimetry field (`colorspace`, `xfer_func`, `ycbcr_enc`, `hsv_enc` or `quantization`) is set to `*_DEFAULT`, then that colorimetry setting will remain unchanged from what was received. So in order to change the quantization, only the `quantization` field shall be set to non default value (`V4L2_QUANTIZATION_FULL_RANGE` or `V4L2_QUANTIZATION_LIM_RANGE`) and all other colorimetry fields shall be set to `*_DEFAULT`. To check which conversions are supported by the hardware for the current media bus frame format, see *Subdev Media Bus Code Enumerate Flags*.

Media Bus Pixel Codes

The media bus pixel codes describe image formats as flowing over physical buses (both between separate physical components and inside SoC devices). This should not be confused with the V4L2 pixel formats that describe, using four character codes, image formats as stored in memory.

While there is a relationship between image formats on buses and image formats in memory (a raw Bayer image won't be magically converted to JPEG just by storing it to memory), there is no one-to-one correspondence between them.

The media bus pixel codes document parallel formats. Should the pixel data be transported over a serial bus, the media bus pixel code that describes a parallel format that transfers a sample on a single clock cycle is used. For instance, both MEDIA_BUS_FMT_BGR888_1X24 and MEDIA_BUS_FMT_BGR888_3X8 are used on parallel busses for transferring an 8 bits per sample BGR data, whereas on serial busses the data in this format is only referred to using MEDIA_BUS_FMT_BGR888_1X24. This is because there is effectively only a single way to transport that format on the serial busses.

Packed RGB Formats

Those formats transfer pixel data as red, green and blue components. The format code is made of the following information.

- The red, green and blue components order code, as encoded in a pixel sample. Possible values are RGB and BGR.
- The number of bits per component, for each component. The values can be different for all components. Common values are 555 and 565.
- The number of bus samples per pixel. Pixels that are wider than the bus width must be transferred in multiple samples. Common values are 1 and 2.
- The bus width.

- For formats where the total number of bits per pixel is smaller than the number of bus samples per pixel times the bus width, a padding value stating if the bytes are padded in their most high order bits (PADHI) or low order bits (PADLO). A “C” prefix is used for component-wise padding in the most high order bits (CPADHI) or low order bits (CPADLO) of each separate component.
- For formats where the number of bus samples per pixel is larger than 1, an endianness value stating if the pixel is transferred MSB first (BE) or LSB first (LE).

For instance, a format where pixels are encoded as 5-bits red, 5-bits green and 5-bit blue values padded on the high bit, transferred as 2 8-bit samples per pixel with the most significant bits (padding, red and half of the green value) transferred first will be named `MEDIA_BUS_FMT_RGB555_2X8_PADHI_BE`.

The following tables list existing packed RGB formats.

Table 92: RGB formats

Identifier	Code	Data organization																																
		Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
MEDIA_BUS_FMT_RGB444_1X12	0x1016																						r_3	r_2	r_1	r_0	g_3	g_2	g_1	g_0	b_3	b_2	b_1	b_0
MEDIA_BUS_FMT_RGB444_2X8_PADI	0x1001																										0	0	0	0	r_3	r_2	r_1	r_0
																											g_3	g_2	g_1	g_0	b_3	b_2	b_1	b_0
MEDIA_BUS_FMT_RGB444_2X8_PADI	0x1002																										g_3	g_2	g_1	g_0	b_3	b_2	b_1	b_0
																											0	0	0	0	r_3	r_2	r_1	r_0
MEDIA_BUS_FMT_RGB555_2X8_PADI	0x1003																										0	r_4	r_3	r_2	r_1	r_0	g_4	g_3
																											g_2	g_1	g_0	b_4	b_3	b_2	b_1	b_0
MEDIA_BUS_FMT_RGB555_2X8_PADI	0x1004																										g_2	g_1	g_0	b_4	b_3	b_2	b_1	b_0
																											0	r_4	r_3	r_2	r_1	r_0	g_4	g_3
MEDIA_BUS_FMT_RGB565_1X16	0x1017																		r_4	r_3	r_2	r_1	r_0	g_5	g_4	g_3	g_2	g_1	g_0	b_4	b_3	b_2	b_1	b_0
MEDIA_BUS_FMT_BGR565_2X8_BE	0x1005																										b_4	b_3	b_2	b_1	b_0	g_5	g_4	g_3
																											g_2	g_1	g_0	r_4	r_3	r_2	r_1	r_0
MEDIA_BUS_FMT_BGR565_2X8_LE	0x1006																										g_2	g_1	g_0	r_4	r_3	r_2	r_1	r_0
																											b_4	b_3	b_2	b_1	b_0	g_5	g_4	g_3
MEDIA_BUS_FMT_RGB565_2X8_BE	0x1007																										r_4	r_3	r_2	r_1	r_0	g_5	g_4	g_3
																											g_2	g_1	g_0	b_4	b_3	b_2	b_1	b_0
MEDIA_BUS_FMT_RGB565_2X8_LE	0x1008																										g_2	g_1	g_0	b_4	b_3	b_2	b_1	b_0
																											r_4	r_3	r_2	r_1	r_0	g_5	g_4	g_3
MEDIA_BUS_FMT_RGB666_1X18	0x1009																r_5	r_4	r_3	r_2	r_1	r_0	g_5	g_4	g_3	g_2	g_1	g_0	b_5	b_4	b_3	b_2	b_1	b_0
MEDIA_BUS_FMT_RBG888_1X24	0x100e										r_7	r_6	r_5	r_4	r_3	r_2	r_1	r_0	b_7	b_6	b_5	b_4	b_3	b_2	b_1	b_0	g_7	g_6	g_5	g_4	g_3	g_2	g_1	g_0
MEDIA_BUS_FMT_RGB666_1X24_CPA	0x1015										0	0	r_5	r_4	r_3	r_2	r_1	r_0	0	0	g_5	g_4	g_3	g_2	g_1	g_0	0	0	b_5	b_4	b_3	b_2	b_1	b_0
MEDIA_BUS_FMT_BGR888_1X24	0x1013										b_7	b_6	b_5	b_4	b_3	b_2	b_1	b_0	g_7	g_6	g_5	g_4	g_3	g_2	g_1	g_0	r_7	r_6	r_5	r_4	r_3	r_2	r_1	r_0
MEDIA_BUS_FMT_BGR888_3X8	0x101b																										b_7	b_6	b_5	b_4	b_3	b_2	b_1	b_0
																											g_7	g_6	g_5	g_4	g_3	g_2	g_1	g_0
																											r_7	r_6	r_5	r_4	r_3	r_2	r_1	r_0
MEDIA_BUS_FMT_GBR888_1X24	0x1014										g_7	g_6	g_5	g_4	g_3	g_2	g_1	g_0	b_7	b_6	b_5	b_4	b_3	b_2	b_1	b_0	r_7	r_6	r_5	r_4	r_3	r_2	r_1	r_0
MEDIA_BUS_FMT_RGB888_1X24	0x100a										r_7	r_6	r_5	r_4	r_3	r_2	r_1	r_0	g_7	g_6	g_5	g_4	g_3	g_2	g_1	g_0	b_7	b_6	b_5	b_4	b_3	b_2	b_1	b_0
MEDIA_BUS_FMT_RGB888_2X12_BE	0x100b																						r_7	r_6	r_5	r_4	r_3	r_2	r_1	r_0	g_7	g_6	g_5	g_4
																							g_3	g_2	g_1	g_0	b_7	b_6	b_5	b_4	b_3	b_2	b_1	b_0
MEDIA_BUS_FMT_RGB888_2X12_LE	0x100c																						g_3	g_2	g_1	g_0	b_7	b_6	b_5	b_4	b_3	b_2	b_1	b_0
																							r_7	r_6	r_5	r_4	r_3	r_2	r_1	r_0	g_7	g_6	g_5	g_4
MEDIA_BUS_FMT_RGB888_3X8	0x101c																										r_7	r_6	r_5	r_4	r_3	r_2	r_1	r_0
																											g_7	g_6	g_5	g_4	g_3	g_2	g_1	g_0
																											b_7	b_6	b_5	b_4	b_3	b_2	b_1	b_0
MEDIA_BUS_FMT_ARGB888_1X32	0x100d		a_7	a_6	a_5	a_4	a_3	a_2	a_1	a_0	r_7	r_6	r_5	r_4	r_3	r_2	r_1	r_0	g_7	g_6	g_5	g_4	g_3	g_2	g_1	g_0	b_7	b_6	b_5	b_4	b_3	b_2	b_1	b_0
MEDIA_BUS_FMT_RGB888_1X32_PAI	0x100f		0	0	0	0	0	0	0	0	r_7	r_6	r_5	r_4	r_3	r_2	r_1	r_0	g_7	g_6	g_5	g_4	g_3	g_2	g_1	g_0	b_7	b_6	b_5	b_4	b_3	b_2	b_1	b_0
MEDIA_BUS_FMT_RGB101010_1X30	0x1018				r_9	r_8	r_7	r_6	r_5	r_4	r_3	r_2	r_1	r_0	g_9	g_8	g_7	g_6	g_5	g_4	g_3	g_2	g_1	g_0	b_9	b_8	b_7	b_6	b_5	b_4	b_3	b_2	b_1	b_0

The following table list existing packed 36bit wide RGB formats.

Table 93: 36bit RGB formats

Identifier	Code	Data organization																																			
		Bit	35	34	33	32	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1
MEDIA_BUS_FMT_RGB121212_1X36	0x1019		r_{11}	r_{10}	r_9	r_8	r_7	r_6	r_5	r_4	r_3	r_2	r_1	r_0	g_{11}	g_{10}	g_9	g_8	g_7	g_6	g_5	g_4	g_3	g_2	g_1	g_0	b_{11}	b_{10}	b_9	b_8	b_7	b_6	b_5	b_4	b_3	b_2	b_1

The following table list existing packed 48bit wide RGB formats.

Table 94: 48bit RGB formats

Identifier	Code	Data organization																																
		Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	47 15	46 14	45 13	44 12	43 11	42 10	41 9	40 8	39 7	38 6	37 5	36 4	35 3	34 2	33 1	32 0
MEDIA_BUS_FMT_RGB161616_1X48	0x101a																		r_{15}	r_{14}	r_{13}	r_{12}	r_{11}	r_{10}	r_9	r_8	r_7	r_6	r_5	r_4	r_3	r_2	r_1	r_0
			g_{15}	g_{14}	g_{13}	g_{12}	g_{11}	g_{10}	g_9	g_8	g_7	g_6	g_5	g_4	g_3	g_2	g_1	g_0	b_{15}	b_{14}	b_{13}	b_{12}	b_{11}	b_{10}	b_9	b_8	b_7	b_6	b_5	b_4	b_3	b_2	b_1	b_0

On LVDS buses, usually each sample is transferred serialized in seven time slots per pixel clock, on three (18-bit) or four (24-bit) differential data pairs at the same time. The remaining bits are used for control signals as defined by SPWG/PSWG/VESA or JEIDA standards. The 24-bit RGB format serialized in seven time slots on four lanes using JEIDA defined bit mapping will be named `MEDIA_BUS_FMT_RGB888_1X7X4_JEIDA`, for example.

Table 95: LVDS RGB formats

Identifier	Code			Data organization			
		Timeslot	Lane	3	2	1	0
MEDIA_BUS_FMT_RGB666_1X7X3_SPWG	0x1010	0			d	b_1	g_0
		1			d	b_0	r_5
		2			d	g_5	r_4
		3			b_5	g_4	r_3
		4			b_4	g_3	r_2
		5			b_3	g_2	r_1
		6			b_2	g_1	r_0
MEDIA_BUS_FMT_RGB888_1X7X4_SPWG	0x1011	0		d	d	b_1	g_0
		1		b_7	d	b_0	r_5
		2		b_6	d	g_5	r_4
		3		g_7	b_5	g_4	r_3
		4		g_6	b_4	g_3	r_2
		5		r_7	b_3	g_2	r_1
		6		r_6	b_2	g_1	r_0
MEDIA_BUS_FMT_RGB888_1X7X4_JEIDA	0x1012	0		d	d	b_3	g_2
		1		b_1	d	b_2	r_7
		2		b_0	d	g_7	r_6
		3		g_1	b_7	g_6	r_5
		4		g_0	b_6	g_5	r_4
		5		r_1	b_5	g_4	r_3
		6		r_0	b_4	g_3	r_2

Bayer Formats

Those formats transfer pixel data as red, green and blue components. The format code is made of the following information.

- The red, green and blue components order code, as encoded in a pixel sample. The possible values are shown in *Figure 4.8 Bayer Patterns*.
- The number of bits per pixel component. All components are transferred on the same number of bits. Common values are 8, 10 and 12.
- The compression (optional). If the pixel components are ALAW- or DPCM-compressed, a mention of the compression scheme and the number of bits per compressed pixel component.
- The number of bus samples per pixel. Pixels that are wider than the bus width must be transferred in multiple samples. Common values are 1 and 2.
- The bus width.
- For formats where the total number of bits per pixel is smaller than the number of bus samples per pixel times the bus width, a padding value stating if

the bytes are padded in their most high order bits (PADHI) or low order bits (PADLO).

- For formats where the number of bus samples per pixel is larger than 1, an endianness value stating if the pixel is transferred MSB first (BE) or LSB first (LE).

For instance, a format with uncompressed 10-bit Bayer components arranged in a red, green, green, blue pattern transferred as 2 8-bit samples per pixel with the least significant bits transferred first will be named `MEDIA_BUS_FMT_SRGGB10_2X8_PADHI_LE`.

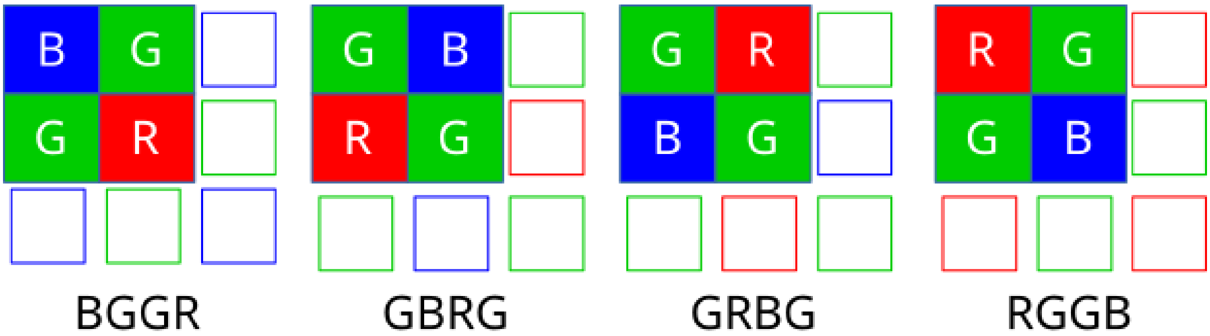

Fig. 17: **Figure 4.8 Bayer Patterns**

The following table lists existing packed Bayer formats. The data organization is given as an example for the first pixel only.

Table 96: Bayer Formats

Identifier	Code	Data organization																
		Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
MEDIA_BUS_FMT_SBGGR8_1X8	0x3001										b_7	b_6	b_5	b_4	b_3	b_2	b_1	b_0
MEDIA_BUS_FMT_SGBRG8_1X8	0x3013										g_7	g_6	g_5	g_4	g_3	g_2	g_1	g_0
MEDIA_BUS_FMT_SGRBG8_1X8	0x3002										g_7	g_6	g_5	g_4	g_3	g_2	g_1	g_0
MEDIA_BUS_FMT_SRGGB8_1X8	0x3014										r_7	r_6	r_5	r_4	r_3	r_2	r_1	r_0
MEDIA_BUS_FMT_SBGGR10_ALAW8	0x3015										b_7	b_6	b_5	b_4	b_3	b_2	b_1	b_0
MEDIA_BUS_FMT_SGBRG10_ALAW8	0x3016										g_7	g_6	g_5	g_4	g_3	g_2	g_1	g_0
MEDIA_BUS_FMT_SGRBG10_ALAW8	0x3017										g_7	g_6	g_5	g_4	g_3	g_2	g_1	g_0
MEDIA_BUS_FMT_SRGGB10_ALAW8	0x3018										r_7	r_6	r_5	r_4	r_3	r_2	r_1	r_0
MEDIA_BUS_FMT_SBGGR10_DPCM8	0x300b										b_7	b_6	b_5	b_4	b_3	b_2	b_1	b_0
MEDIA_BUS_FMT_SGBRG10_DPCM8	0x300c										g_7	g_6	g_5	g_4	g_3	g_2	g_1	g_0
MEDIA_BUS_FMT_SGRBG10_DPCM8	0x3009										g_7	g_6	g_5	g_4	g_3	g_2	g_1	g_0
MEDIA_BUS_FMT_SRGGB10_DPCM8	0x300d										r_7	r_6	r_5	r_4	r_3	r_2	r_1	r_0
MEDIA_BUS_FMT_SBGGR10_2X8_PA	0x3003										0 b_7	0 b_6	0 b_5	0 b_4	0 b_3	0 b_2	b_9 b_1	b_8 b_0
MEDIA_BUS_FMT_SBGGR10_2X8_PA	0x3004										b_7 0	b_6 0	b_5 0	b_4 0	b_3 0	b_2 0	b_1 b_9	b_0 b_8
MEDIA_BUS_FMT_SBGGR10_2X8_PA	0x3005										b_9 b_1	b_8 b_0	b_7 0	b_6 0	b_5 0	b_4 0	b_3 0	b_2 0
MEDIA_BUS_FMT_SBGGR10_2X8_PA	0x3006										b_1 b_9	b_0 b_8	0 b_7	0 b_6	0 b_5	0 b_4	0 b_3	0 b_2
MEDIA_BUS_FMT_SBGGR10_1X10	0x3007								b_9	b_8	b_7	b_6	b_5	b_4	b_3	b_2	b_1	b_0
MEDIA_BUS_FMT_SGBRG10_1X10	0x300e								g_9	g_8	g_7	g_6	g_5	g_4	g_3	g_2	g_1	g_0
MEDIA_BUS_FMT_SGRBG10_1X10	0x300a								g_9	g_8	g_7	g_6	g_5	g_4	g_3	g_2	g_1	g_0
MEDIA_BUS_FMT_SRGGB10_1X10	0x300f								r_9	r_8	r_7	r_6	r_5	r_4	r_3	r_2	r_1	r_0
MEDIA_BUS_FMT_SBGGR12_1X12	0x3008						b_{11}	b_{10}	b_9	b_8	b_7	b_6	b_5	b_4	b_3	b_2	b_1	b_0
MEDIA_BUS_FMT_SGBRG12_1X12	0x3010						g_{11}	g_{10}	g_9	g_8	g_7	g_6	g_5	g_4	g_3	g_2	g_1	g_0
MEDIA_BUS_FMT_SGRBG12_1X12	0x3011						g_{11}	g_{10}	g_9	g_8	g_7	g_6	g_5	g_4	g_3	g_2	g_1	g_0

continues on next page

Table 96 – continued from previous page

Identifier	Code	Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
			Data organization															
MEDIA_BUS_FMT_SRGGB12_1X12	0x3012						r_{11}	r_{10}	r_9	r_8	r_7	r_6	r_5	r_4	r_3	r_2	r_1	r_0
MEDIA_BUS_FMT_SBGGR14_1X14	0x3019				b_{13}	b_{12}	b_{11}	b_{10}	b_9	b_8	b_7	b_6	b_5	b_4	b_3	b_2	b_1	b_0
MEDIA_BUS_FMT_SGBRG14_1X14	0x301a				g_{13}	g_{12}	g_{11}	g_{10}	g_9	g_8	g_7	g_6	g_5	g_4	g_3	g_2	g_1	g_0
MEDIA_BUS_FMT_SGRBG14_1X14	0x301b				g_{13}	g_{12}	g_{11}	g_{10}	g_9	g_8	g_7	g_6	g_5	g_4	g_3	g_2	g_1	g_0
MEDIA_BUS_FMT_SRGGB14_1X14	0x301c				r_{13}	r_{12}	r_{11}	r_{10}	r_9	r_8	r_7	r_6	r_5	r_4	r_3	r_2	r_1	r_0
MEDIA_BUS_FMT_SBGGR16_1X16	0x301d		b_{15}	b_{14}	b_{13}	b_{12}	b_{11}	b_{10}	b_9	b_8	b_7	b_6	b_5	b_4	b_3	b_2	b_1	b_0
MEDIA_BUS_FMT_SGBRG16_1X16	0x301e		g_{15}	g_{14}	g_{13}	g_{12}	g_{11}	g_{10}	g_9	g_8	g_7	g_6	g_5	g_4	g_3	g_2	g_1	g_0
MEDIA_BUS_FMT_SGRBG16_1X16	0x301f		g_{15}	g_{14}	g_{13}	g_{12}	g_{11}	g_{10}	g_9	g_8	g_7	g_6	g_5	g_4	g_3	g_2	g_1	g_0
MEDIA_BUS_FMT_SRGGB16_1X16	0x3020		r_{15}	r_{14}	r_{13}	r_{12}	r_{11}	r_{10}	r_9	r_8	r_7	r_6	r_5	r_4	r_3	r_2	r_1	r_0

Packed YUV Formats

Those data formats transfer pixel data as (possibly downsampled) Y, U and V components. Some formats include dummy bits in some of their samples and are collectively referred to as "YDYC" (Y-Dummy-Y-Chroma) formats. One cannot rely on the values of these dummy bits as those are undefined.

The format code is made of the following information.

- The Y, U and V components order code, as transferred on the bus. Possible values are YUYV, UYVY, YVYU and VYUY for formats with no dummy bit, and YDYUYDYV, YDYVYDYU, YUYDYVYD and YVYDYUYD for YDYC formats.
- The number of bits per pixel component. All components are transferred on the same number of bits. Common values are 8, 10 and 12.
- The number of bus samples per pixel. Pixels that are wider than the bus width must be transferred in multiple samples. Common values are 0.5 (encoded as 0_5; in this case two pixels are transferred per bus sample), 1, 1.5 (encoded as 1_5) and 2.
- The bus width. When the bus width is larger than the number of bits per pixel component, several components are packed in a single bus sample. The components are ordered as specified by the order code, with components on the left of the code transferred in the high order bits. Common values are 8 and 16.

For instance, a format where pixels are encoded as 8-bit YUV values downsampled to 4:2:2 and transferred as 2 8-bit bus samples per pixel in the U, Y, V, Y order will be named `MEDIA_BUS_FMT_UYVY8_2X8`.

YUV Formats lists existing packed YUV formats and describes the organization of each pixel data in each sample. When a format pattern is split across multiple samples each of the samples in the pattern is described.

The role of each bit transferred over the bus is identified by one of the following codes.

- y_x for luma component bit number x
- u_x for blue chroma component bit number x
- v_x for red chroma component bit number x
- a_x for alpha component bit number x

- for non-available bits (for positions higher than the bus width)
- d for dummy bits

Table 97: YUV Formats

Identifier	Code	Data organization																																
		Bit	31	30	29	28	27	26	25	24	23	22	21	10	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
MEDIA_BUS_FMT_Y8_1X8	0x2001																										y_7	y_6	y_5	y_4	y_3	y_2	y_1	y_0
MEDIA_BUS_FMT_UV8_1X8	0x2015																										u_7	u_6	u_5	u_4	u_3	u_2	u_1	u_0
																											v_7	v_6	v_5	v_4	v_3	v_2	v_1	v_0
MEDIA_BUS_FMT_UYVY8_1_5X8	0x2002																										u_7	u_6	u_5	u_4	u_3	u_2	u_1	u_0
																											y_7	y_6	y_5	y_4	y_3	y_2	y_1	y_0
																											y_7	y_6	y_5	y_4	y_3	y_2	y_1	y_0
																											v_7	v_6	v_5	v_4	v_3	v_2	v_1	v_0
																											y_7	y_6	y_5	y_4	y_3	y_2	y_1	y_0
																											y_7	y_6	y_5	y_4	y_3	y_2	y_1	y_0
MEDIA_BUS_FMT_VYUY8_1_5X8	0x2003																										v_7	v_6	v_5	v_4	v_3	v_2	v_1	v_0
																											y_7	y_6	y_5	y_4	y_3	y_2	y_1	y_0
																											y_7	y_6	y_5	y_4	y_3	y_2	y_1	y_0
																											u_7	u_6	u_5	u_4	u_3	u_2	u_1	u_0
																											y_7	y_6	y_5	y_4	y_3	y_2	y_1	y_0
																											y_7	y_6	y_5	y_4	y_3	y_2	y_1	y_0
MEDIA_BUS_FMT_YUYV8_1_5X8	0x2004																										y_7	y_6	y_5	y_4	y_3	y_2	y_1	y_0
																											y_7	y_6	y_5	y_4	y_3	y_2	y_1	y_0
																											u_7	u_6	u_5	u_4	u_3	u_2	u_1	u_0
																											y_7	y_6	y_5	y_4	y_3	y_2	y_1	y_0
																											y_7	y_6	y_5	y_4	y_3	y_2	y_1	y_0
																											v_7	v_6	v_5	v_4	v_3	v_2	v_1	v_0
MEDIA_BUS_FMT_YVYU8_1_5X8	0x2005																										y_7	y_6	y_5	y_4	y_3	y_2	y_1	y_0
																											y_7	y_6	y_5	y_4	y_3	y_2	y_1	y_0
																											v_7	v_6	v_5	v_4	v_3	v_2	v_1	v_0
																											y_7	y_6	y_5	y_4	y_3	y_2	y_1	y_0
																											y_7	y_6	y_5	y_4	y_3	y_2	y_1	y_0
																											u_7	u_6	u_5	u_4	u_3	u_2	u_1	u_0
MEDIA_BUS_FMT_UYVY8_2X8	0x2006																										u_7	u_6	u_5	u_4	u_3	u_2	u_1	u_0
																											y_7	y_6	y_5	y_4	y_3	y_2	y_1	y_0
																											v_7	v_6	v_5	v_4	v_3	v_2	v_1	v_0
																											y_7	y_6	y_5	y_4	y_3	y_2	y_1	y_0
MEDIA_BUS_FMT_VYUY8_2X8	0x2007																										v_7	v_6	v_5	v_4	v_3	v_2	v_1	v_0
																											y_7	y_6	y_5	y_4	y_3	y_2	y_1	y_0
																											u_7	u_6	u_5	u_4	u_3	u_2	u_1	u_0
																											y_7	y_6	y_5	y_4	y_3	y_2	y_1	y_0
MEDIA_BUS_FMT_YUYV8_2X8	0x2008																										y_7	y_6	y_5	y_4	y_3	y_2	y_1	y_0
																											u_7	u_6	u_5	u_4	u_3	u_2	u_1	u_0
																											y_7	y_6	y_5	y_4	y_3	y_2	y_1	y_0
																											v_7	v_6	v_5	v_4	v_3	v_2	v_1	v_0
MEDIA_BUS_FMT_YVYU8_2X8	0x2009																										y_7	y_6	y_5	y_4	y_3	y_2	y_1	y_0
																											v_7	v_6	v_5	v_4	v_3	v_2	v_1	v_0
																											y_7	y_6	y_5	y_4	y_3	y_2	y_1	y_0
																											u_7	u_6	u_5	u_4	u_3	u_2	u_1	u_0
MEDIA_BUS_FMT_Y10_1X10	0x200a																								y_9	y_8	y_7	y_6	y_5	y_4	y_3	y_2	y_1	y_0
MEDIA_BUS_FMT_Y10_2X8_PADHI_L	0x202c																										y_7	y_6	y_5	y_4	y_3	y_2	y_1	y_0
																											0	0	0	0	0	0	y_9	y_8
MEDIA_BUS_FMT_UYVY10_2X10	0x2018																								u_9	u_8	u_7	u_6	u_5	u_4	u_3	u_2	u_1	u_0
																									y_9	y_8	y_7	y_6	y_5	y_4	y_3	y_2	y_1	y_0
																									v_9	v_8	v_7	v_6	v_5	v_4	v_3	v_2	v_1	v_0
																									y_9	y_8	y_7	y_6	y_5	y_4	y_3	y_2	y_1	y_0
MEDIA_BUS_FMT_VYUY10_2X10	0x2019																								v_9	v_8	v_7	v_6	v_5	v_4	v_3	v_2	v_1	v_0
																									y_9	y_8	y_7	y_6	y_5	y_4	y_3	y_2	y_1	y_0
																									u_9	u_8	u_7	u_6	u_5	u_4	u_3	u_2	u_1	u_0
																									y_9	y_8	y_7	y_6	y_5	y_4	y_3	y_2	y_1	y_0
MEDIA_BUS_FMT_YUYV10_2X10	0x200b																								y_9	y_8	y_7	y_6	y_5	y_4	y_3	y_2	y_1	y_0
																									u_9	u_8	u_7	u_6	u_5	u_4	u_3	u_2	u_1	u_0
																									y_9	y_8	y_7	y_6	y_5	y_4	y_3	y_2	y_1	y_0
																									v_9	v_8	v_7	v_6	v_5	v_4	v_3	v_2	v_1	v_0
MEDIA_BUS_FMT_YVYU10_2X10	0x200c																								y_9	y_8	y_7	y_6	y_5	y_4	y_3	y_2	y_1	y_0
																									v_9	v_8	v_7	v_6	v_5	v_4	v_3	v_2	v_1	v_0
																									y_9	y_8	y_7	y_6	y_5	y_4	y_3	y_2	y_1	y_0
																									u_9	u_8	u_7	u_6	u_5	u_4	u_3	u_2	u_1	u_0
MEDIA_BUS_FMT_Y12_1X12	0x2013																						y_{11}	y_{10}	y_9	y_8	y_7	y_6	y_5	y_4	y_3	y_2	y_1	y_0
MEDIA_BUS_FMT_UYVY12_2X12	0x201c																						u_{11}	u_{10}	u_9	u_8	u_7	u_6	u_5	u_4	u_3	u_2	u_1	u_0
																							y_{11}	y_{10}	y_9	y_8	y_7	y_6	y_5	y_4	y_3	y_2	y_1	y_0
																							v_{11}	v_{10}	v_9	v_8	v_7	v_6	v_5	v_4	v_3	v_2	v_1	v_0
																							y_{11}	y_{10}	y_9	y_8	y_7	y_6	y_5	y_4	y_3	y_2	y_1	y_0
MEDIA_BUS_FMT_VYUY12_2X12	0x201d																						v_{11}	v_{10}	v_9	v_8	v_7	v_6	v_5	v_4	v_3	v_2	v_1	v_0
																							y_{11}	y_{10}	y_9	y_8	y_7	y_6	y_5	y_4	y_3	y_2	y_1	y_0
																							u_{11}	u_{10}	u_9	u_8	u_7	u_6	u_5	u_4	u_3	u_2	u_1	u_0
																							y_{11}	y_{10}	y_9	y_8	y_7	y_6	y_5	y_4	y_3	y_2	y_1	y_0
MEDIA_BUS_FMT_YUYV12_2X12	0x201e																						y_{11}	y_{10}	y_9	y_8	y_7	y_6	y_5	y_4	y_3	y_2	y_1	y_0
																							u_{11}	u_{10}	u_9	u_8	u_7	u_6	u_5	u_4	u_3	u_2	u_1	u_0
																							y_{11}	y_{10}	y_9	y_8	y_7	y_6	y_5	y_4	y_3	y_2	y_1	y_0

continues on next page

Table 97 – continued from previous page

Identifier	Code	Bit	31	30	29	28	27	26	25	24	23	22	21	10	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
																							v_{11}	v_{10}	v_9	v_8	v_7	v_6	v_5	v_4	v_3	v_2	v_1	v_0
MEDIA_BUS_FMT_YVYU12_2X12	0x201f																						y_{11}	y_{10}	y_9	y_8	y_7	y_6	y_5	y_4	y_3	y_2	y_1	y_0
																							v_{11}	v_{10}	v_9	v_8	v_7	v_6	v_5	v_4	v_3	v_2	v_1	v_0
																							y_{11}	y_{10}	y_9	y_8	y_7	y_6	y_5	y_4	y_3	y_2	y_1	y_0
																							u_{11}	u_{10}	u_9	u_8	u_7	u_6	u_5	u_4	u_3	u_2	u_1	u_0
MEDIA_BUS_FMT_Y14_1X14	0x202d																				y_{13}	y_{12}	y_{11}	y_{10}	y_9	y_8	y_7	y_6	y_5	y_4	y_3	y_2	y_1	y_0
MEDIA_BUS_FMT_UYVY8_1X16	0x200f																		u_7	u_6	u_5	u_4	u_3	u_2	u_1	u_0	y_7	y_6	y_5	y_4	y_3	y_2	y_1	y_0
																			v_7	v_6	v_5	v_4	v_3	v_2	v_1	v_0	y_7	y_6	y_5	y_4	y_3	y_2	y_1	y_0
MEDIA_BUS_FMT_VYUY8_1X16	0x2010																		v_7	v_6	v_5	v_4	v_3	v_2	v_1	v_0	y_7	y_6	y_5	y_4	y_3	y_2	y_1	y_0
																			u_7	u_6	u_5	u_4	u_3	u_2	u_1	u_0	y_7	y_6	y_5	y_4	y_3	y_2	y_1	y_0
MEDIA_BUS_FMT_YUYV8_1X16	0x2011																		y_7	y_6	y_5	y_4	y_3	y_2	y_1	y_0	u_7	u_6	u_5	u_4	u_3	u_2	u_1	u_0
																			y_7	y_6	y_5	y_4	y_3	y_2	y_1	y_0	v_7	v_6	v_5	v_4	v_3	v_2	v_1	v_0
MEDIA_BUS_FMT_YVYU8_1X16	0x2012																		y_7	y_6	y_5	y_4	y_3	y_2	y_1	y_0	v_7	v_6	v_5	v_4	v_3	v_2	v_1	v_0
																			y_7	y_6	y_5	y_4	y_3	y_2	y_1	y_0	u_7	u_6	u_5	u_4	u_3	u_2	u_1	u_0
MEDIA_BUS_FMT_YDYUYDYV8_1X16	0x2014																		y_7	y_6	y_5	y_4	y_3	y_2	y_1	y_0	d	d	d	d	d	d	d	d
																			y_7	y_6	y_5	y_4	y_3	y_2	y_1	y_0	u_7	u_6	u_5	u_4	u_3	u_2	u_1	u_0
																			y_7	y_6	y_5	y_4	y_3	y_2	y_1	y_0	d	d	d	d	d	d	d	d
																			y_7	y_6	y_5	y_4	y_3	y_2	y_1	y_0	v_7	v_6	v_5	v_4	v_3	v_2	v_1	v_0
MEDIA_BUS_FMT_UYVY10_1X20	0x201a														u_9	u_8	u_7	u_6	u_5	u_4	u_3	u_2	u_1	u_0	y_9	y_8	y_7	y_6	y_5	y_4	y_3	y_2	y_1	y_0
															v_9	v_8	v_7	v_6	v_5	v_4	v_3	v_2	v_1	v_0	y_9	y_8	y_7	y_6	y_5	y_4	y_3	y_2	y_1	y_0
MEDIA_BUS_FMT_VYUY10_1X20	0x201b														v_9	v_8	v_7	v_6	v_5	v_4	v_3	v_2	v_1	v_0	y_9	y_8	y_7	y_6	y_5	y_4	y_3	y_2	y_1	y_0
															u_9	u_8	u_7	u_6	u_5	u_4	u_3	u_2	u_1	u_0	y_9	y_8	y_7	y_6	y_5	y_4	y_3	y_2	y_1	y_0
MEDIA_BUS_FMT_YUYV10_1X20	0x200d														y_9	y_8	y_7	y_6	y_5	y_4	y_3	y_2	y_1	y_0	u_9	u_8	u_7	u_6	u_5	u_4	u_3	u_2	u_1	u_0
															y_9	y_8	y_7	y_6	y_5	y_4	y_3	y_2	y_1	y_0	v_9	v_8	v_7	v_6	v_5	v_4	v_3	v_2	v_1	v_0
MEDIA_BUS_FMT_YVYU10_1X20	0x200e														y_9	y_8	y_7	y_6	y_5	y_4	y_3	y_2	y_1	y_0	v_9	v_8	v_7	v_6	v_5	v_4	v_3	v_2	v_1	v_0
															y_9	y_8	y_7	y_6	y_5	y_4	y_3	y_2	y_1	y_0	u_9	u_8	u_7	u_6	u_5	u_4	u_3	u_2	u_1	u_0
MEDIA_BUS_FMT_VUY8_1X24	0x201a										v_7	v_6	v_5	v_4	v_3	v_2	v_1	v_0	u_7	u_6	u_5	u_4	u_3	u_2	u_1	u_0	y_7	y_6	y_5	y_4	y_3	y_2	y_1	y_0
MEDIA_BUS_FMT_YUV8_1X24	0x2025										y_7	y_6	y_5	y_4	y_3	y_2	y_1	y_0	u_7	u_6	u_5	u_4	u_3	u_2	u_1	u_0	v_7	v_6	v_5	v_4	v_3	v_2	v_1	v_0
MEDIA_BUS_FMT_UYYVYY8_0_5X24	0x2026										u_7	u_6	u_5	u_4	u_3	u_2	u_1	u_0	y_7	y_6	y_5	y_4	y_3	y_2	y_1	y_0	y_7	y_6	y_5	y_4	y_3	y_2	y_1	y_0
											v_7	v_6	v_5	v_4	v_3	v_2	v_1	v_0	y_7	y_6	y_5	y_4	y_3	y_2	y_1	y_0	y_7	y_6	y_5	y_4	y_3	y_2	y_1	y_0
MEDIA_BUS_FMT_UYVY12_1X24	0x2020										u_{11}	u_{10}	u_9	u_8	u_7	u_6	u_5	u_4	u_3	u_2	u_1	u_0	y_{11}	y_{10}	y_9	y_8	y_7	y_6	y_5	y_4	y_3	y_2	y_1	y_0
											v_{11}	v_{10}	v_9	v_8	v_7	v_6	v_5	v_4	v_3	v_2	v_1	v_0	y_{11}	y_{10}	y_9	y_8	y_7	y_6	y_5	y_4	y_3	y_2	y_1	y_0
MEDIA_BUS_FMT_VYUY12_1X24	0x2021										v_{11}	v_{10}	v_9	v_8	v_7	v_6	v_5	v_4	v_3	v_2	v_1	v_0	y_{11}	y_{10}	y_9	y_8	y_7	y_6	y_5	y_4	y_3	y_2	y_1	y_0
											u_{11}	u_{10}	u_9	u_8	u_7	u_6	u_5	u_4	u_3	u_2	u_1	u_0	y_{11}	y_{10}	y_9	y_8	y_7	y_6	y_5	y_4	y_3	y_2	y_1	y_0
MEDIA_BUS_FMT_YUYV12_1X24	0x2022										y_{11}	y_{10}	y_9	y_8	y_7	y_6	y_5	y_4	y_3	y_2	y_1	y_0	u_{11}	u_{10}	u_9	u_8	u_7	u_6	u_5	u_4	u_3	u_2	u_1	u_0
											y_{11}	y_{10}	y_9	y_8	y_7	y_6	y_5	y_4	y_3	y_2	y_1	y_0	v_{11}	v_{10}	v_9	v_8	v_7	v_6	v_5	v_4	v_3	v_2	v_1	v_0
MEDIA_BUS_FMT_YVYU12_1X24	0x2023										y_{11}	y_{10}	y_9	y_8	y_7	y_6	y_5	y_4	y_3	y_2	y_1	y_0	v_{11}	v_{10}	v_9	v_8	v_7	v_6	v_5	v_4	v_3	v_2	v_1	v_0
											y_{11}	y_{10}	y_9	y_8	y_7	y_6	y_5	y_4	y_3	y_2	y_1	y_0	u_{11}	u_{10}	u_9	u_8	u_7	u_6	u_5	u_4	u_3	u_2	u_1	u_0
MEDIA_BUS_FMT_YUV10_1X30	0x2016				y_9	y_8	y_7	y_6	y_5	y_4	y_3	y_2	y_1	y_0	u_9	u_8	u_7	u_6	u_5	u_4	u_3	u_2	u_1	u_0	v_9	v_8	v_7	v_6	v_5	v_4	v_3	v_2	v_1	v_0
MEDIA_BUS_FMT_UYYVYY10_0_5X30	0x2027				u_9	u_8	u_7	u_6	u_5	u_4	u_3	u_2	u_1	u_0	y_9	y_8	y_7	y_6	y_5	y_4	y_3	y_2	y_1	y_0	y_9	y_8	y_7	y_6	y_5	y_4	y_3	y_2	y_1	y_0
					v_9	v_8	v_7	v_6	v_5	v_4	v_3	v_2	v_1	v_0	y_9	y_8	y_7	y_6	y_5	y_4	y_3	y_2	y_1	y_0	y_9	y_8	y_7	y_6	y_5	y_4	y_3	y_2	y_1	y_0
MEDIA_BUS_FMT_AYUV8_1X32	0x2017		a_7	a_6	a_5	a_4	a_3	a_2	a_1	a_0	y_7	y_6	y_5	y_4	y_3	y_2	y_1	y_0	u_7	u_6	u_5	u_4	u_3	u_2	u_1	u_0	v_7	v_6	v_5	v_4	v_3	v_2	v_1	v_0

The following table list existing packed 36bit wide YUV formats.

Table 98: 36bit YUV Formats

Identifier	Code	Bit	35	34	33	32	31	30	29	28	27	26	25	24	23	22	21	10	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1
MEDIA_BUS_FMT_UYYVYY12_0_5X36	0x2028		u_{11}	u_{10}	u_9	u_8	u_7	u_6	u_5	u_4	u_3	u_2	u_1	u_0	y_{11}	y_{10}	y_9	y_8	y_7	y_6	y_5	y_4	y_3	y_2	y_1	y_0	y_{11}	y_{10}	y_9	y_8	y_7	y_6	y_5	y_4	y_3	y_2	y_1
			v_{11}	v_{10}	v_9	v_8	v_7	v_6	v_5	v_4	v_3	v_2	v_1	v_0	y_{11}	y_{10}	y_9	y_8	y_7	y_6	y_5	y_4	y_3	y_2	y_1	y_0	y_{11}	y_{10}	y_9	y_8	y_7	y_6	y_5	y_4	y_3	y_2	y_1
MEDIA_BUS_FMT_YUV12_1X36	0x2029		y_{11}	y_{10}	y_9	y_8	y_7	y_6	y_5	y_4	y_3	y_2	y_1	y_0	u_{11}	u_{10}	u_9	u_8	u_7	u_6	u_5	u_4	u_3	u_2	u_1	u_0	v_{11}	v_{10}	v_9	v_8	v_7	v_6	v_5	v_4	v_3	v_2	v_1

The following table list existing packed 48bit wide YUV formats.

Table 99: 48bit YUV Formats

Identifier	Code	Bit	31	30	29	28	27	26	25	24	23	22	21	10	19	18	17	16	47 15	46 14	45 13	44 12	43 11	42 10	41 9	40 8	39 7	38 6	37 5	36 4	35 3	34 2	33 1	32 0
MEDIA_BUS_FMT_YUV16_1X48	0x202a																		y_{15}	y_{14}	y_{13}	y_{12}	y_{11}	y_{10}	y_8	y_8	y_7	y_6	y_5	y_4	y_3	y_2	y_1	y_0
			u_{15}	u_{14}	u_{13}	u_{12}	u_{11}	u_{10}	u_9	u_8	u_7	u_6	u_5	u_4	u_3	u_2	u_1	u_0	v_{15}	v_{14}	v_{13}	v_{12}	v_{11}	v_{10}	v_9	v_8	v_7	v_6	v_5	v_4	v_3	v_2	v_1	v_0
MEDIA_BUS_FMT_UYYVYY16_0_5X48	0x202b																		u_{15}	u_{14}	u_{13}	u_{12}	u_{11}	u_{10}	u_9	u_8	u_7	u_6	u_5	u_4	u_3	u_2	u_1	u_0
			y_{15}	y_{14}	y_{13}	y_{12}	y_{11}	y_{10}	y_9	y_8	y_7	y_6	y_5	y_4	y_3	y_2	y_1	y_0	y_{15}	y_{14}	y_{13}	y_{12}	y_{11}	y_{10}	y_8	y_8	y_7	y_6	y_5	y_4	y_3	y_2	y_1	y_0
																			v_{15}	v_{14}	v_{13}	v_{12}	v_{11}	v_{10}	v_9	v_8	v_7	v_6	v_5	v_4	v_3	v_2	v_1	v_0
			y_{15}	y_{14}	y_{13}	y_{12}	y_{11}	y_{10}	y_9	y_8	y_7	y_6	y_5	y_4	y_3	y_2	y_1	y_0	y_{15}	y_{14}	y_{13}	y_{12}	y_{11}	y_{10}	y_8	y_8	y_7	y_6	y_5	y_4	y_3	y_2	y_1	y_0

HSV/HSL Formats

Those formats transfer pixel data as RGB values in a cylindrical-coordinate system using Hue-Saturation-Value or Hue-Saturation-Lightness components. The format code is made of the following information.

- The hue, saturation, value or lightness and optional alpha components order code, as encoded in a pixel sample. The only currently supported value is AHSV.
- The number of bits per component, for each component. The values can be different for all components. The only currently supported value is 8888.
- The number of bus samples per pixel. Pixels that are wider than the bus width must be transferred in multiple samples. The only currently supported value is 1.
- The bus width.
- For formats where the total number of bits per pixel is smaller than the number of bus samples per pixel times the bus width, a padding value stating if the bytes are padded in their most high order bits (PADHI) or low order bits (PADLO).
- For formats where the number of bus samples per pixel is larger than 1, an endianness value stating if the pixel is transferred MSB first (BE) or LSB first (LE).

The following table lists existing HSV/HSL formats.

Table 100: HSV/HSL formats

Identifier	Code		Data organization																															
		Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
MEDIA_BUS_FMT_AHSV8888_1X32	0x6001		a_7	a_6	a_5	a_4	a_3	a_2	a_1	a_0	h_7	h_6	h_5	h_4	h_3	h_2	h_1	h_0	s_7	s_6	s_5	s_4	s_3	s_2	s_1	s_0	v_7	v_6	v_5	v_4	v_3	v_2	v_1	v_0

JPEG Compressed Formats

Those data formats consist of an ordered sequence of 8-bit bytes obtained from JPEG compression process. Additionally to the `_JPEG` postfix the format code is made of the following information.

- The number of bus samples per entropy encoded byte.
- The bus width.

For instance, for a JPEG baseline process and an 8-bit bus width the format will be named `MEDIA_BUS_FMT_JPEG_1X8`.

The following table lists existing JPEG compressed formats.

Table 101: JPEG Formats

Identifier	Code	Remarks
MEDIA_BUS_FMT_JPEG_1X8	0x4001	Besides of its usage for the parallel bus this format is recommended for transmission of JPEG data over MIPI CSI bus using the User Defined 8-bit Data types.

Vendor and Device Specific Formats

This section lists complex data formats that are either vendor or device specific.

The following table lists the existing vendor and device specific formats.

Table 102: Vendor and device specific formats

Identifier	Code	Comments
MEDIA_BUS_FMT_S5C_UYVY_JPEG_1X8	0x5001	Interleaved raw UYVY and JPEG image format with embedded meta-data used by Samsung S3C73MX camera sensors.

Metadata Interface

Metadata refers to any non-image data that supplements video frames with additional information. This may include statistics computed over the image, frame capture parameters supplied by the image source or device specific parameters for specifying how the device processes images. This interface is intended for transfer of metadata between the userspace and the hardware and control of that operation.

The metadata interface is implemented on video device nodes. The device can be dedicated to metadata or can support both video and metadata as specified in its reported capabilities.

Querying Capabilities

Device nodes supporting the metadata capture interface set the `V4L2_CAP_META_CAPTURE` flag in the `device_caps` field of the `v4l2_capability` structure returned by the `VIDIOC_QUERYCAP()` ioctl. That flag means the device can capture metadata to memory. Similarly, device nodes supporting metadata output interface set the `V4L2_CAP_META_OUTPUT` flag in the `device_caps` field of `v4l2_capability` structure. That flag means the device can read metadata from memory.

At least one of the read/write or streaming I/O methods must be supported.

Data Format Negotiation

The metadata device uses the *Data Formats* ioctls to select the capture format. The metadata buffer content format is bound to that selected format. In addition to the basic *Data Formats* ioctls, the `VIDIOC_ENUM_FMT()` ioctl must be supported as well.

To use the *Data Formats* ioctls applications set the `type` field of the `v4l2_format` structure to `V4L2_BUF_TYPE_META_CAPTURE` or to `V4L2_BUF_TYPE_META_OUTPUT` and use the *v4l2_meta_format* `meta` member of the `fmt` union as needed per the desired operation. Both drivers and applications must set the remainder of the `v4l2_format` structure to 0.

type **v4l2_meta_format**

Table 103: struct v4l2_meta_format

__u32	`dataformat`	The data format, set by the application. This is a little endian *four character code*. V4L2 defines metadata formats in *Metadata Formats*.
__u32	`buffersize`	Maximum buffer size in bytes required for data. The value is set by the driver.

8.2.5 Libv4l Userspace Library

Introduction

libv4l is a collection of libraries which adds a thin abstraction layer on top of video4linux2 devices. The purpose of this (thin) layer is to make it easy for application writers to support a wide variety of devices without having to write separate code for different devices in the same class.

An example of using libv4l is provided by *v4l2grab*.

libv4l consists of 3 different libraries:

libv4lconvert

libv4lconvert is a library that converts several different pixelformats found in V4L2 drivers into a few common RGB and YUY formats.

It currently accepts the following V4L2 driver formats: *V4L2_PIX_FMT_BGR24*, *V4L2_PIX_FMT_HM12*, *V4L2_PIX_FMT_JPEG*, *V4L2_PIX_FMT_MJPEG*, *V4L2_PIX_FMT_MR97310A*, *V4L2_PIX_FMT_OV511*, *V4L2_PIX_FMT_OV518*, *V4L2_PIX_FMT_PAC207*, *V4L2_PIX_FMT_PJPG*, *V4L2_PIX_FMT_RGB24*, *V4L2_PIX_FMT_SBGGR8*, *V4L2_PIX_FMT_SGBRG8*, *V4L2_PIX_FMT_SGRBG8*, *V4L2_PIX_FMT_SN9C10X*, *V4L2_PIX_FMT_SN9C20X_I420*, *V4L2_PIX_FMT_SPCA501*, *V4L2_PIX_FMT_SPCA505*, *V4L2_PIX_FMT_SPCA508*, *V4L2_PIX_FMT_SPCA561*, *V4L2_PIX_FMT_SQ905C*, *V4L2_PIX_FMT_SRGGB8*, *V4L2_PIX_FMT_UYVY*, *V4L2_PIX_FMT_YUV420*, *V4L2_PIX_FMT_YUYV*, *V4L2_PIX_FMT_YVU420*, and *V4L2_PIX_FMT_YVYU*.

Later on libv4lconvert was expanded to also be able to do various video processing functions to improve webcam video quality. The video processing is split in to 2 parts: libv4lconvert/control and libv4lconvert/processing.

The control part is used to offer video controls which can be used to control the video processing functions made available by libv4lconvert/processing. These controls are stored application wide (until reboot) by using a persistent shared memory object.

libv4lconvert/processing offers the actual video processing functionality.

libv4l1

This library offers functions that can be used to quickly make v4l1 applications work with v4l2 devices. These functions work exactly like the normal open/close/etc, except that libv4l1 does full emulation of the v4l1 api on top of v4l2 drivers, in case of v4l1 drivers it will just pass calls through.

Since those functions are emulations of the old V4L1 API, it shouldn't be used for new applications.

libv4l2

This library should be used for all modern V4L2 applications.

It provides handles to call V4L2 open/ioctl/close/poll methods. Instead of just providing the raw output of the device, it enhances the calls in the sense that it will use libv4lconvert to provide more video formats and to enhance the image quality.

In most cases, libv4l2 just passes the calls directly through to the v4l2 driver, intercepting the calls to *VIDIOC_TRY_FMT*, *VIDIOC_G_FMT*, *VIDIOC_S_FMT*, *VIDIOC_ENUM_FRAMESIZES* and *VIDIOC_ENUM_FRAMEINTERVALS* in order to emulate the formats *V4L2_PIX_FMT_BGR24*, *V4L2_PIX_FMT_RGB24*, *V4L2_PIX_FMT_YUV420*, and *V4L2_PIX_FMT_YVU420*, if they aren't available in the driver. *VIDIOC_ENUM_FMT* keeps enumerating the hardware supported formats, plus the emulated formats offered by libv4l at the end.

Libv4l device control functions

The common file operation methods are provided by libv4l.

Those functions operate just like the gcc function `dup()` and V4L2 functions *`open()`*, *`close()`*, `ioctl()`, *`read()`*, *`mmap()`* and *`munmap()`*:

int **`v4l2_open`**(const char *file, int oflag, ...)

  operates like the *`open()`* function.

int **`v4l2_close`**(int fd)

  operates like the *`close()`* function.

int **`v4l2_dup`**(int fd)

  operates like the libc `dup()` function, duplicating a file handler.

int **v4l2_ioctl**(int fd, unsigned long int request, ...)
: operates like the `ioctl()` function.

int **v4l2_read**(int fd, void *buffer, size_t n)
: operates like the *read()* function.

void **v4l2_mmap**(void *start, size_t length, int prot, int flags, int fd, int64_t offset);
: operates like the *munmap()* function.

int **v4l2_munmap**(void *_start, size_t length);
: operates like the *munmap()* function.

Those functions provide additional control:

int **v4l2_fd_open**(int fd, int v4l2_flags)
: opens an already opened fd for further use through v4l2lib and possibly modify libv4l2's default behavior through the `v4l2_flags` argument. Currently, `v4l2_flags` can be `V4L2_DISABLE_CONVERSION`, to disable format conversion.

int **v4l2_set_control**(int fd, int cid, int value)
: This function takes a value of 0 - 65535, and then scales that range to the actual range of the given v4l control id, and then if the cid exists and is not locked sets the cid to the scaled value.

int **v4l2_get_control**(int fd, int cid)
: This function returns a value of 0 - 65535, scaled to from the actual range of the given v4l control id. when the cid does not exist, could not be accessed for some reason, or some error occurred 0 is returned.

v4l1compat.so wrapper library

This library intercepts calls to *open()*, *close()*, `ioctl()`, *mmap()* and *munmap()* operations and redirects them to the libv4l counterparts, by using `LD_PRELOAD=/usr/lib/v4l1compat.so`. It also emulates V4L1 calls via V4L2 API.

It allows usage of binary legacy applications that still don't use libv4l.

8.2.6 Changes

The following chapters document the evolution of the V4L2 API, errata or extensions. They are also intended to help application and driver writers to port or update their code.

Differences between V4L and V4L2

The Video For Linux API was first introduced in Linux 2.1 to unify and replace various TV and radio device related interfaces, developed independently by driver writers in prior years. Starting with Linux 2.5 the much improved V4L2 API replaces the V4L API. The support for the old V4L calls were removed from Kernel, but the library *Libv4l Userspace Library* supports the conversion of a V4L API system call into a V4L2 one.

Opening and Closing Devices

For compatibility reasons the character device file names recommended for V4L2 video capture, overlay, radio and raw vbi capture devices did not change from those used by V4L. They are listed in *Interfaces* and below in *V4L Device Types, Names and Numbers*.

The teletext devices (minor range 192-223) have been removed in V4L2 and no longer exist. There is no hardware available anymore for handling pure teletext. Instead raw or sliced VBI is used.

The V4L `videodev` module automatically assigns minor numbers to drivers in load order, depending on the registered device type. We recommend that V4L2 drivers by default register devices with the same numbers, but the system administrator can assign arbitrary minor numbers using driver module options. The major device number remains 81.

Table 104: V4L Device Types, Names and Numbers

Device Type	File Name	Minor Numbers
Video capture and overlay	`/dev/video` and `/dev/bttv0`[1], `/dev/video0` to `/dev/video63`	0-63
Radio receiver	`/dev/radio`[2], `/dev/radio0` to `/dev/radio63`	64-127
Raw VBI capture	`/dev/vbi`, `/dev/vbi0` to `/dev/vbi31`	224-255

V4L prohibits (or used to prohibit) multiple opens of a device file. V4L2 drivers *may* support multiple opens, see *Opening and Closing Devices* for details and consequences.

V4L drivers respond to V4L2 ioctls with an `EINVAL` error code.

[1] According to Documentation/admin-guide/devices.rst these should be symbolic links to `/dev/video0`. Note the original bttv interface is not compatible with V4L or V4L2.

[2] According to `Documentation/admin-guide/devices.rst` a symbolic link to `/dev/radio0`.

Querying Capabilities

The V4L VIDIOCGCAP ioctl is equivalent to V4L2's *ioctl VIDIOC_QUERYCAP*.

The name field in struct video_capability became card in struct *v4l2_capability*, type was replaced by capabilities. Note V4L2 does not distinguish between device types like this, better think of basic video input, video output and radio devices supporting a set of related functions like video capturing, video overlay and VBI capturing. See *Opening and Closing Devices* for an introduction.

struct video_capability type	struct *v4l2_capability* capabilities flags	Purpose
VID_TYPE_CAPTURE	V4L2_CAP_VIDEO_CAPTURE	The *video capture* interface is supported.
VID_TYPE_TUNER	V4L2_CAP_TUNER	The device has a *tuner or modulator*.
VID_TYPE_TELETEXT	V4L2_CAP_VBI_CAPTURE	The *raw VBI capture* interface is supported.
VID_TYPE_OVERLAY	V4L2_CAP_VIDEO_OVERLAY	The *video overlay* interface is supported.
VID_TYPE_CHROMAKEY	V4L2_FBUF_CAP_CHROMAKEY in field capability of struct *v4l2_framebuffer*	Whether chromakey overlay is supported. For more information on overlay see *Video Overlay Interface*.
VID_TYPE_CLIPPING	V4L2_FBUF_CAP_LIST_CLIPPING and V4L2_FBUF_CAP_BITMAP_CLIP in field capability of struct *v4l2_framebuffer*	Whether clipping the overlaid image is supported, see *Video Overlay Interface*.
VID_TYPE_FRAMERAM	V4L2_FBUF_CAP_EXTERNOVERLAY *not set* in field capability of struct *v4l2_framebuffer*	Whether overlay overwrites frame buffer memory, see *Video Overlay Interface*.
VID_TYPE_SCALES	-	This flag indicates if the hardware can scale images. The V4L2 API implies the scale factor by setting the cropping dimensions and image size with the *VIDIOC_S_CROP* and *VIDIOC_S_FMT* ioctl, respectively. The driver returns the closest sizes possible. For more information on cropping and scaling see *Image Cropping, Insertion and Scaling – the CROP API*.

continues on next page

Table 105 – continued from previous page

struct video_capability type	struct *v4l2_capability* capabilities flags	Purpose
VID_TYPE_MONOCHROME	-	Applications can enumerate the supported image formats with the *ioctl VIDIOC_ENUM_FMT* ioctl to determine if the device supports grey scale capturing only. For more information on image formats see *Image Formats*.
VID_TYPE_SUBCAPTURE	-	Applications can call the *VIDIOC_G_CROP* ioctl to determine if the device supports capturing a subsection of the full picture ("cropping" in V4L2). If not, the ioctl returns the EINVAL error code. For more information on cropping and scaling see *Image Cropping, Insertion and Scaling – the CROP API*.
VID_TYPE_MPEG_DECODER	-	Applications can enumerate the supported image formats with the *ioctl VIDIOC_ENUM_FMT* ioctl to determine if the device supports MPEG streams.
VID_TYPE_MPEG_ENCODER	-	See above.
VID_TYPE_MJPEG_DECODER	-	See above.
VID_TYPE_MJPEG_ENCODER	-	See above.

The `audios` field was replaced by `capabilities` flag `V4L2_CAP_AUDIO`, indicating *if* the device has any audio inputs or outputs. To determine their number applications can enumerate audio inputs with the *VIDIOC_G_AUDIO* ioctl. The audio ioctls are described in *Audio Inputs and Outputs*.

The `maxwidth`, `maxheight`, `minwidth` and `minheight` fields were removed. Calling the *VIDIOC_S_FMT* or *VIDIOC_TRY_FMT* ioctl with the desired dimensions returns the closest size possible, taking into account the current video standard, cropping and scaling limitations.

Video Sources

V4L provides the VIDIOCGCHAN and VIDIOCSCHAN ioctl using struct video_channel to enumerate the video inputs of a V4L device. The equivalent V4L2 ioctls are *ioctl VIDIOC_ENUMINPUT*, *VIDIOC_G_INPUT* and *VIDIOC_S_INPUT* using struct *v4l2_input* as discussed in *Video Inputs and Outputs*.

The channel field counting inputs was renamed to index, the video input types were renamed as follows:

struct video_channel type	struct *v4l2_input* type
VIDEO_TYPE_TV	V4L2_INPUT_TYPE_TUNER
VIDEO_TYPE_CAMERA	V4L2_INPUT_TYPE_CAMERA

Unlike the tuners field expressing the number of tuners of this input, V4L2 assumes each video input is connected to at most one tuner. However a tuner can have more than one input, i. e. RF connectors, and a device can have multiple tuners. The index number of the tuner associated with the input, if any, is stored in field tuner of struct *v4l2_input*. Enumeration of tuners is discussed in *Tuners and Modulators*.

The redundant VIDEO_VC_TUNER flag was dropped. Video inputs associated with a tuner are of type V4L2_INPUT_TYPE_TUNER. The VIDEO_VC_AUDIO flag was replaced by the audioset field. V4L2 considers devices with up to 32 audio inputs. Each set bit in the audioset field represents one audio input this video input combines with. For information about audio inputs and how to switch between them see *Audio Inputs and Outputs*.

The norm field describing the supported video standards was replaced by std. The V4L specification mentions a flag VIDEO_VC_NORM indicating whether the standard can be changed. This flag was a later addition together with the norm field and has been removed in the meantime. V4L2 has a similar, albeit more comprehensive approach to video standards, see *Video Standards* for more information.

Tuning

The V4L VIDIOCGTUNER and VIDIOCSTUNER ioctl and struct video_tuner can be used to enumerate the tuners of a V4L TV or radio device. The equivalent V4L2 ioctls are *VIDIOC_G_TUNER* and *VIDIOC_S_TUNER* using struct *v4l2_tuner*. Tuners are covered in *Tuners and Modulators*.

The tuner field counting tuners was renamed to index. The fields name, rangelow and rangehigh remained unchanged.

The VIDEO_TUNER_PAL, VIDEO_TUNER_NTSC and VIDEO_TUNER_SECAM flags indicating the supported video standards were dropped. This information is now contained in the associated struct *v4l2_input*. No replacement exists for the VIDEO_TUNER_NORM flag indicating whether the video standard can be switched. The mode field to select a different video standard was replaced by a whole new set of ioctls and structures described in *Video Standards*. Due to its ubiquity it should be mentioned the BTTV driver supports several standards in addition to the regular

VIDEO_MODE_PAL (0), VIDEO_MODE_NTSC, VIDEO_MODE_SECAM and VIDEO_MODE_AUTO (3). Namely N/PAL Argentina, M/PAL, N/PAL, and NTSC Japan with numbers 3-6 (sic).

The VIDEO_TUNER_STEREO_ON flag indicating stereo reception became V4L2_TUNER_SUB_STEREO in field rxsubchans. This field also permits the detection of monaural and bilingual audio, see the definition of struct *v4l2_tuner* for details. Presently no replacement exists for the VIDEO_TUNER_RDS_ON and VIDEO_TUNER_MBS_ON flags.

The VIDEO_TUNER_LOW flag was renamed to V4L2_TUNER_CAP_LOW in the struct *v4l2_tuner* capability field.

The VIDIOCGFREQ and VIDIOCSFREQ ioctl to change the tuner frequency where renamed to *VIDIOC_G_FREQUENCY* and *VIDIOC_S_FREQUENCY*. They take a pointer to a struct *v4l2_frequency* instead of an unsigned long integer.

Image Properties

V4L2 has no equivalent of the VIDIOCGPICT and VIDIOCSPICT ioctl and struct video_picture. The following fields where replaced by V4L2 controls accessible with the *ioctls VIDIOC_QUERYCTRL, VIDIOC_QUERY_EXT_CTRL and VIDIOC_QUERYMENU*, *VIDIOC_G_CTRL* and *VIDIOC_S_CTRL* ioctls:

struct video_picture	V4L2 Control ID
brightness	V4L2_CID_BRIGHTNESS
hue	V4L2_CID_HUE
colour	V4L2_CID_SATURATION
contrast	V4L2_CID_CONTRAST
whiteness	V4L2_CID_WHITENESS

The V4L picture controls are assumed to range from 0 to 65535 with no particular reset value. The V4L2 API permits arbitrary limits and defaults which can be queried with the *ioctls VIDIOC_QUERYCTRL, VIDIOC_QUERY_EXT_CTRL and VIDIOC_QUERYMENU* ioctl. For general information about controls see *User Controls*.

The depth (average number of bits per pixel) of a video image is implied by the selected image format. V4L2 does not explicitly provide such information assuming applications recognizing the format are aware of the image depth and others need not know. The palette field moved into the struct *v4l2_pix_format*:

struct video_picture palette	struct *v4l2_pix_format* pixfmt
VIDEO_PALETTE_GREY	*V4L2_PIX_FMT_GREY*
VIDEO_PALETTE_HI240	*V4L2_PIX_FMT_HI240*[3]
VIDEO_PALETTE_RGB565	*V4L2_PIX_FMT_RGB565*
VIDEO_PALETTE_RGB555	*V4L2_PIX_FMT_RGB555*
VIDEO_PALETTE_RGB24	*V4L2_PIX_FMT_BGR24*
VIDEO_PALETTE_RGB32	*V4L2_PIX_FMT_BGR32*[4]
VIDEO_PALETTE_YUV422	*V4L2_PIX_FMT_YUYV*
VIDEO_PALETTE_YUYV[5]	*V4L2_PIX_FMT_YUYV*
VIDEO_PALETTE_UYVY	*V4L2_PIX_FMT_UYVY*
VIDEO_PALETTE_YUV420	None
VIDEO_PALETTE_YUV411	*V4L2_PIX_FMT_Y41P*[6]
VIDEO_PALETTE_RAW	None[7]
VIDEO_PALETTE_YUV422P	*V4L2_PIX_FMT_YUV422P*
VIDEO_PALETTE_YUV411P	*V4L2_PIX_FMT_YUV411P*[8]
VIDEO_PALETTE_YUV420P	*V4L2_PIX_FMT_YVU420*
VIDEO_PALETTE_YUV410P	*V4L2_PIX_FMT_YVU410*

V4L2 image formats are defined in *Image Formats*. The image format can be selected with the *VIDIOC_S_FMT* ioctl.

Audio

The VIDIOCGAUDIO and VIDIOCSAUDIO ioctl and struct video_audio are used to enumerate the audio inputs of a V4L device. The equivalent V4L2 ioctls are *VIDIOC_G_AUDIO* and *VIDIOC_S_AUDIO* using struct *v4l2_audio* as discussed in *Audio Inputs and Outputs*.

The audio "channel number" field counting audio inputs was renamed to index.

On VIDIOCSAUDIO the mode field selects *one* of the VIDEO_SOUND_MONO, VIDEO_SOUND_STEREO, VIDEO_SOUND_LANG1 or VIDEO_SOUND_LANG2 audio demodulation modes. When the current audio standard is BTSC VIDEO_SOUND_LANG2 refers to SAP and VIDEO_SOUND_LANG1 is meaningless. Also undocumented in the V4L specification, there is no way to query the selected mode. On VIDIOCGAUDIO the driver returns the *actually received* audio programmes in this field. In the V4L2 API this information is stored in the struct *v4l2_tuner* rxsubchans and audmode fields, respectively. See *Tuners and Modulators* for more information on tuners. Related to audio modes struct *v4l2_audio* also reports if this is a mono or stereo input, regardless if the source is a tuner.

[3] This is a custom format used by the BTTV driver, not one of the V4L2 standard formats.

[4] Presumably all V4L RGB formats are little-endian, although some drivers might interpret them according to machine endianness. V4L2 defines little-endian, big-endian and red/blue swapped variants. For details see *RGB Formats*.

[5] VIDEO_PALETTE_YUV422 and VIDEO_PALETTE_YUYV are the same formats. Some V4L drivers respond to one, some to the other.

[6] Not to be confused with V4L2_PIX_FMT_YUV411P, which is a planar format.

[7] V4L explains this as: "RAW capture (BT848)"

[8] Not to be confused with V4L2_PIX_FMT_Y41P, which is a packed format.

The following fields where replaced by V4L2 controls accessible with the *ioctls VIDIOC_QUERYCTRL, VIDIOC_QUERY_EXT_CTRL and VIDIOC_QUERYMENU*, *VIDIOC_G_CTRL* and *VIDIOC_S_CTRL* ioctls:

struct video_audio	V4L2 Control ID
volume	V4L2_CID_AUDIO_VOLUME
bass	V4L2_CID_AUDIO_BASS
treble	V4L2_CID_AUDIO_TREBLE
balance	V4L2_CID_AUDIO_BALANCE

To determine which of these controls are supported by a driver V4L provides the flags VIDEO_AUDIO_VOLUME, VIDEO_AUDIO_BASS, VIDEO_AUDIO_TREBLE and VIDEO_AUDIO_BALANCE. In the V4L2 API the *ioctls VIDIOC_QUERYCTRL, VIDIOC_QUERY_EXT_CTRL and VIDIOC_QUERYMENU* ioctl reports if the respective control is supported. Accordingly the VIDEO_AUDIO_MUTABLE and VIDEO_AUDIO_MUTE flags where replaced by the boolean V4L2_CID_AUDIO_MUTE control.

All V4L2 controls have a step attribute replacing the struct video_audio step field. The V4L audio controls are assumed to range from 0 to 65535 with no particular reset value. The V4L2 API permits arbitrary limits and defaults which can be queried with the *ioctls VIDIOC_QUERYCTRL, VIDIOC_QUERY_EXT_CTRL and VIDIOC_QUERYMENU* ioctl. For general information about controls see *User Controls*.

Frame Buffer Overlay

The V4L2 ioctls equivalent to VIDIOCGFBUF and VIDIOCSFBUF are *VIDIOC_G_FBUF* and *VIDIOC_S_FBUF*. The base field of struct video_buffer remained unchanged, except V4L2 defines a flag to indicate non-destructive overlays instead of a NULL pointer. All other fields moved into the struct *v4l2_pix_format* fmt substructure of struct *v4l2_framebuffer*. The depth field was replaced by pixelformat. See *RGB Formats* for a list of RGB formats and their respective color depths.

Instead of the special ioctls VIDIOCGWIN and VIDIOCSWIN V4L2 uses the general-purpose data format negotiation ioctls *VIDIOC_G_FMT* and *VIDIOC_S_FMT*. They take a pointer to a struct *v4l2_format* as argument. Here the win member of the fmt union is used, a struct *v4l2_window*.

The x, y, width and height fields of struct video_window moved into struct *v4l2_rect* substructure w of struct *v4l2_window*. The chromakey, clips, and clipcount fields remained unchanged. Struct video_clip was renamed to struct *v4l2_clip*, also containing a struct *v4l2_rect*, but the semantics are still the same.

The VIDEO_WINDOW_INTERLACE flag was dropped. Instead applications must set the field field to V4L2_FIELD_ANY or V4L2_FIELD_INTERLACED. The VIDEO_WINDOW_CHROMAKEY flag moved into struct *v4l2_framebuffer*, under the new name V4L2_FBUF_FLAG_CHROMAKEY.

In V4L, storing a bitmap pointer in clips and setting clipcount to VIDEO_CLIP_BITMAP (-1) requests bitmap clipping, using a fixed size bitmap of

1024 × 625 bits. Struct *v4l2_window* has a separate bitmap pointer field for this purpose and the bitmap size is determined by w.width and w.height.

The VIDIOCCAPTURE ioctl to enable or disable overlay was renamed to *ioctl VIDIOC_OVERLAY*.

Cropping

To capture only a subsection of the full picture V4L defines the VIDIOCGCAPTURE and VIDIOCSCAPTURE ioctls using struct video_capture. The equivalent V4L2 ioctls are *VIDIOC_G_CROP* and *VIDIOC_S_CROP* using struct *v4l2_crop*, and the related *ioctl VIDIOC_CROPCAP* ioctl. This is a rather complex matter, see *Image Cropping, Insertion and Scaling – the CROP API* for details.

The x, y, width and height fields moved into struct *v4l2_rect* substructure c of struct *v4l2_crop*. The decimation field was dropped. In the V4L2 API the scaling factor is implied by the size of the cropping rectangle and the size of the captured or overlaid image.

The VIDEO_CAPTURE_ODD and VIDEO_CAPTURE_EVEN flags to capture only the odd or even field, respectively, were replaced by V4L2_FIELD_TOP and V4L2_FIELD_BOTTOM in the field named field of struct *v4l2_pix_format* and struct *v4l2_window*. These structures are used to select a capture or overlay format with the *VIDIOC_S_FMT* ioctl.

Reading Images, Memory Mapping

Capturing using the read method

There is no essential difference between reading images from a V4L or V4L2 device using the *read()* function, however V4L2 drivers are not required to support this I/O method. Applications can determine if the function is available with the *ioctl VIDIOC_QUERYCAP* ioctl. All V4L2 devices exchanging data with applications must support the *select()* and *poll()* functions.

To select an image format and size, V4L provides the VIDIOCSPICT and VIDIOCSWIN ioctls. V4L2 uses the general-purpose data format negotiation ioctls *VIDIOC_G_FMT* and *VIDIOC_S_FMT*. They take a pointer to a struct *v4l2_format* as argument, here the struct *v4l2_pix_format* named pix of its fmt union is used.

For more information about the V4L2 read interface see *Read/Write*.

Capturing using memory mapping

Applications can read from V4L devices by mapping buffers in device memory, or more often just buffers allocated in DMA-able system memory, into their address space. This avoids the data copying overhead of the read method. V4L2 supports memory mapping as well, with a few differences.

V4L	V4L2
	The image format must be selected before buffers are allocated, with the *VIDIOC_S_FMT* ioctl. When no format is selected the driver may use the last, possibly by another application requested format.
Applications cannot change the number of buffers. The it is built into the driver, unless it has a module option to change the number when the driver module is loaded.	The *ioctl VIDIOC_REQBUFS* ioctl allocates the desired number of buffers, this is a required step in the initialization sequence.
Drivers map all buffers as one contiguous range of memory. The `VIDIOCGMBUF` ioctl is available to query the number of buffers, the offset of each buffer from the start of the virtual file, and the overall amount of memory used, which can be used as arguments for the `mmap()` function.	Buffers are individually mapped. The offset and size of each buffer can be determined with the *ioctl VIDIOC_QUERYBUF* ioctl.
The `VIDIOCMCAPTURE` ioctl prepares a buffer for capturing. It also determines the image format for this buffer. The ioctl returns immediately, eventually with an `EAGAIN` error code if no video signal had been detected. When the driver supports more than one buffer applications can call the ioctl multiple times and thus have multiple outstanding capture requests. The `VIDIOCSYNC` ioctl suspends execution until a particular buffer has been filled.	Drivers maintain an incoming and outgoing queue. *ioctl VIDIOC_QBUF, VIDIOC_DQBUF* enqueues any empty buffer into the incoming queue. Filled buffers are dequeued from the outgoing queue with the *VIDIOC_DQBUF* ioctl. To wait until filled buffers become available this function, `select()` or `poll()` can be used. The *ioctl VIDIOC_STREAMON, VIDIOC_STREAMOFF* ioctl must be called once after enqueueing one or more buffers to start capturing. Its counterpart *VIDIOC_STREAMOFF* stops capturing and dequeues all buffers from both queues. Applications can query the signal status, if known, with the *ioctl VIDIOC_ENUMINPUT* ioctl.

For a more in-depth discussion of memory mapping and examples, see *Streaming I/O (Memory Mapping)*.

Reading Raw VBI Data

Originally the V4L API did not specify a raw VBI capture interface, only the device file /dev/vbi was reserved for this purpose. The only driver supporting this interface was the BTTV driver, de-facto defining the V4L VBI interface. Reading from the device yields a raw VBI image with the following parameters:

struct *v4l2_vbi_form*	V4L, BTTV driver
sampling_rate	28636363 Hz NTSC (or any other 525-line standard); 35468950 Hz PAL and SECAM (625-line standards)
offset	?
samples_per_line	2048
sample_format	V4L2_PIX_FMT_GREY. The last four bytes (a machine endianness integer) contain a frame counter.
start[]	10, 273 NTSC; 22, 335 PAL and SECAM
count[]	16, 16[9]
flags	0

Undocumented in the V4L specification, in Linux 2.3 the VIDIOCGVBIFMT and VIDIOCSVBIFMT ioctls using struct vbi_format were added to determine the VBI image parameters. These ioctls are only partially compatible with the V4L2 VBI interface specified in *Raw VBI Data Interface*.

An offset field does not exist, sample_format is supposed to be VIDEO_PALETTE_RAW, equivalent to V4L2_PIX_FMT_GREY. The remaining fields are probably equivalent to struct *v4l2_vbi_format*.

Apparently only the Zoran (ZR 36120) driver implements these ioctls. The semantics differ from those specified for V4L2 in two ways. The parameters are reset on *open()* and VIDIOCSVBIFMT always returns an EINVAL error code if the parameters are invalid.

Miscellaneous

V4L2 has no equivalent of the VIDIOCGUNIT ioctl. Applications can find the VBI device associated with a video capture device (or vice versa) by reopening the device and requesting VBI data. For details see *Opening and Closing Devices*.

No replacement exists for VIDIOCKEY, and the V4L functions for microcode programming. A new interface for MPEG compression and playback devices is documented in *Extended Controls API*.

[9] Old driver versions used different values, eventually the custom BTTV_VBISIZE ioctl was added to query the correct values.

Changes of the V4L2 API

Soon after the V4L API was added to the kernel it was criticised as too inflexible. In August 1998 Bill Dirks proposed a number of improvements and began to work on documentation, example drivers and applications. With the help of other volunteers this eventually became the V4L2 API, not just an extension but a replacement for the V4L API. However it took another four years and two stable kernel releases until the new API was finally accepted for inclusion into the kernel in its present form.

Early Versions

1998-08-20: First version.

1998-08-27: The *select()* function was introduced.

1998-09-10: New video standard interface.

1998-09-18: The `VIDIOC_NONCAP` ioctl was replaced by the otherwise meaningless `O_TRUNC` *open()* flag, and the aliases `O_NONCAP` and `O_NOIO` were defined. Applications can set this flag if they intend to access controls only, as opposed to capture applications which need exclusive access. The `VIDEO_STD_XXX` identifiers are now ordinals instead of flags, and the `video_std_construct()` helper function takes id and transmission arguments.

1998-09-28: Revamped video standard. Made video controls individually enumerable.

1998-10-02: The `id` field was removed from struct `video_standard` and the color subcarrier fields were renamed. The *ioctl VIDIOC_QUERYSTD, VIDIOC_SUBDEV_QUERYSTD* ioctl was renamed to *ioctl VIDIOC_ENUMSTD, VIDIOC_SUBDEV_ENUMSTD*, *VIDIOC_G_INPUT* to *ioctl VIDIOC_ENUMINPUT*. A first draft of the Codec API was released.

1998-11-08: Many minor changes. Most symbols have been renamed. Some material changes to struct v4l2_capability.

1998-11-12: The read/write directon of some ioctls was misdefined.

1998-11-14: `V4L2_PIX_FMT_RGB24` changed to `V4L2_PIX_FMT_BGR24`, and `V4L2_PIX_FMT_RGB32` changed to `V4L2_PIX_FMT_BGR32`. Audio controls are now accessible with the *VIDIOC_G_CTRL* and *VIDIOC_S_CTRL* ioctls under names starting with `V4L2_CID_AUDIO`. The `V4L2_MAJOR` define was removed from `videodev.h` since it was only used once in the `videodev` kernel module. The `YUV422` and `YUV411` planar image formats were added.

1998-11-28: A few ioctl symbols changed. Interfaces for codecs and video output devices were added.

1999-01-14: A raw VBI capture interface was added.

1999-01-19: The `VIDIOC_NEXTBUF` ioctl was removed.

V4L2 Version 0.16 1999-01-31

1999-01-27: There is now one QBUF ioctl, VIDIOC_QWBUF and VIDIOC_QRBUF are gone. VIDIOC_QBUF takes a v4l2_buffer as a parameter. Added digital zoom (cropping) controls.

V4L2 Version 0.18 1999-03-16

Added a v4l to V4L2 ioctl compatibility layer to videodev.c. Driver writers, this changes how you implement your ioctl handler. See the Driver Writer's Guide. Added some more control id codes.

V4L2 Version 0.19 1999-06-05

1999-03-18: Fill in the category and catname fields of v4l2_queryctrl objects before passing them to the driver. Required a minor change to the VIDIOC_QUERYCTRL handlers in the sample drivers.

1999-03-31: Better compatibility for v4l memory capture ioctls. Requires changes to drivers to fully support new compatibility features, see Driver Writer's Guide and v4l2cap.c. Added new control IDs: V4L2_CID_HFLIP, _VFLIP. Changed V4L2_PIX_FMT_YUV422P to _YUV422P, and _YUV411P to _YUV411P.

1999-04-04: Added a few more control IDs.

1999-04-07: Added the button control type.

1999-05-02: Fixed a typo in videodev.h, and added the V4L2_CTRL_FLAG_GRAYED (later V4L2_CTRL_FLAG_GRABBED) flag.

1999-05-20: Definition of VIDIOC_G_CTRL was wrong causing a malfunction of this ioctl.

1999-06-05: Changed the value of V4L2_CID_WHITENESS.

V4L2 Version 0.20 (1999-09-10)

Version 0.20 introduced a number of changes which were *not backward compatible* with 0.19 and earlier versions. Purpose of these changes was to simplify the API, while making it more extensible and following common Linux driver API conventions.

1. Some typos in `V4L2_FMT_FLAG` symbols were fixed. *`struct v4l2_clip`* was changed for compatibility with v4l. (1999-08-30)
2. `V4L2_TUNER_SUB_LANG1` was added. (1999-09-05)
3. All ioctl() commands that used an integer argument now take a pointer to an integer. Where it makes sense, ioctls will return the actual new value in the integer pointed to by the argument, a common convention in the V4L2 API. The affected ioctls are: VIDIOC_PREVIEW, VIDIOC_STREAMON, VIDIOC_STREAMOFF, VIDIOC_S_FREQ, VIDIOC_S_INPUT, VIDIOC_S_OUTPUT, VIDIOC_S_EFFECT. For example

```
err = ioctl (fd, VIDIOC_XXX, V4L2_XXX);
```

becomes

```
int a = V4L2_XXX; err = ioctl(fd, VIDIOC_XXX, &a);
```

4. All the different get- and set-format commands were swept into one *VIDIOC_G_FMT* and *VIDIOC_S_FMT* ioctl taking a union and a type field selecting the union member as parameter. Purpose is to simplify the API by eliminating several ioctls and to allow new and driver private data streams without adding new ioctls.

   This change obsoletes the following ioctls: `VIDIOC_S_INFMT`, `VIDIOC_G_INFMT`, `VIDIOC_S_OUTFMT`, `VIDIOC_G_OUTFMT`, `VIDIOC_S_VBIFMT` and `VIDIOC_G_VBIFMT`. The image format struct v4l2_format was renamed to *struct v4l2_pix_format*, while struct v4l2_format is now the envelopping structure for all format negotiations.

5. Similar to the changes above, the `VIDIOC_G_PARM` and `VIDIOC_S_PARM` ioctls were merged with `VIDIOC_G_OUTPARM` and `VIDIOC_S_OUTPARM`. A `type` field in the new struct v4l2_streamparm selects the respective union member.

   This change obsoletes the `VIDIOC_G_OUTPARM` and `VIDIOC_S_OUTPARM` ioctls.

6. Control enumeration was simplified, and two new control flags were introduced and one dropped. The `catname` field was replaced by a `group` field.

   Drivers can now flag unsupported and temporarily unavailable controls with `V4L2_CTRL_FLAG_DISABLED` and `V4L2_CTRL_FLAG_GRABBED` respectively. The `group` name indicates a possibly narrower classification than the `category`. In other words, there may be multiple groups within a category. Controls within a group would typically be drawn within a group box. Controls in different categories might have a greater separation, or may even appear in separate windows.

7. The struct v4l2_buffer `timestamp` was changed to a 64 bit integer, containing the sampling or output time of the frame in nanoseconds. Additionally timestamps will be in absolute system time, not starting from zero at the beginning of a stream. The data type name for timestamps is stamp_t, defined as a signed 64-bit integer. Output devices should not send a buffer out until the time in the timestamp field has arrived. I would like to follow SGI's lead, and adopt a multimedia timestamping system like their UST (Unadjusted System Time). See http://web.archive.org/web/*/http://reality.sgi.com/cpirazzi_engr/lg/time/intro.html. UST uses timestamps that are 64-bit signed integers (not struct timeval's) and given in nanosecond units. The UST clock starts at zero when the system is booted and runs continuously and uniformly. It takes a little over 292 years for UST to overflow. There is no way to set the UST clock. The regular Linux time-of-day clock can be changed periodically, which would cause errors if it were being used for timestamping a multimedia stream. A real UST style clock will require some support in the kernel that is not there yet. But in anticipation, I will change the timestamp field to a 64-bit integer, and I will change the v4l2_masterclock_gettime() function (used only by drivers) to return a 64-bit integer.

8. A sequence field was added to struct v4l2_buffer. The sequence field counts captured frames, it is ignored by output devices. When a capture driver drops a frame, the sequence number of that frame is skipped.

V4L2 Version 0.20 incremental changes

1999-12-23: In struct v4l2_vbi_format the reserved1 field became offset. Previously drivers were required to clear the reserved1 field.

2000-01-13: The V4L2_FMT_FLAG_NOT_INTERLACED flag was added.

2000-07-31: The linux/poll.h header is now included by videodev.h for compatibility with the original videodev.h file.

2000-11-20: V4L2_TYPE_VBI_OUTPUT and V4L2_PIX_FMT_Y41P were added.

2000-11-25: V4L2_TYPE_VBI_INPUT was added.

2000-12-04: A couple typos in symbol names were fixed.

2001-01-18: To avoid namespace conflicts the fourcc macro defined in the videodev.h header file was renamed to v4l2_fourcc.

2001-01-25: A possible driver-level compatibility problem between the videodev.h file in Linux 2.4.0 and the videodev.h file included in the videodevX patch was fixed. Users of an earlier version of videodevX on Linux 2.4.0 should recompile their V4L and V4L2 drivers.

2001-01-26: A possible kernel-level incompatibility between the videodev.h file in the videodevX patch and the videodev.h file in Linux 2.2.x with devfs patches applied was fixed.

2001-03-02: Certain V4L ioctls which pass data in both direction although they are defined with read-only parameter, did not work correctly through the backward compatibility layer. [Solution?]

2001-04-13: Big endian 16-bit RGB formats were added.

2001-09-17: New YUV formats and the *VIDIOC_G_FREQUENCY* and *VIDIOC_S_FREQUENCY* ioctls were added. (The old VIDIOC_G_FREQ and VIDIOC_S_FREQ ioctls did not take multiple tuners into account.)

2000-09-18: V4L2_BUF_TYPE_VBI was added. This may *break compatibility* as the *VIDIOC_G_FMT* and *VIDIOC_S_FMT* ioctls may fail now if the struct v4l2_fmt type field does not contain V4L2_BUF_TYPE_VBI. In the documentation of the struct v4l2_vbi_format`, the offset field the ambiguous phrase "rising edge" was changed to "leading edge".

V4L2 Version 0.20 2000-11-23

A number of changes were made to the raw VBI interface.

1. Figures clarifying the line numbering scheme were added to the V4L2 API specification. The `start[0]` and `start[1]` fields no longer count line numbers beginning at zero. Rationale: a) The previous definition was unclear. b) The `start[]` values are ordinal numbers. c) There is no point in inventing a new line numbering scheme. We now use line number as defined by ITU-R, period. Compatibility: Add one to the start values. Applications depending on the previous semantics may not function correctly.
2. The restriction "count[0] > 0 and count[1] > 0" has been relaxed to "(count[0] + count[1]) > 0". Rationale: Drivers may allocate resources at scan line granularity and some data services are transmitted only on the first field. The comment that both `count` values will usually be equal is misleading and pointless and has been removed. This change *breaks compatibility* with earlier versions: Drivers may return `EINVAL`, applications may not function correctly.
3. Drivers are again permitted to return negative (unknown) start values as proposed earlier. Why this feature was dropped is unclear. This change may *break compatibility* with applications depending on the start values being positive. The use of `EBUSY` and `EINVAL` error codes with the *VIDIOC_S_FMT* ioctl was clarified. The `EBUSY` error code was finally documented, and the `reserved2` field which was previously mentioned only in the `videodev.h` header file.
4. New buffer types `V4L2_TYPE_VBI_INPUT` and `V4L2_TYPE_VBI_OUTPUT` were added. The former is an alias for the old `V4L2_TYPE_VBI`, the latter was missing in the `videodev.h` file.

V4L2 Version 0.20 2002-07-25

Added sliced VBI interface proposal.

V4L2 in Linux 2.5.46, 2002-10

Around October-November 2002, prior to an announced feature freeze of Linux 2.5, the API was revised, drawing from experience with V4L2 0.20. This unnamed version was finally merged into Linux 2.5.46.

1. As specified in *Related Devices*, drivers must make related device functions available under all minor device numbers.
2. The *`open()`* function requires access mode `O_RDWR` regardless of the device type. All V4L2 drivers exchanging data with applications must support the `O_NONBLOCK` flag. The `O_NOIO` flag, a V4L2 symbol which aliased the meaningless `O_TRUNC` to indicate accesses without data exchange (panel applications) was dropped. Drivers must stay in "panel mode" until the application attempts to initiate a data exchange, see *Opening and Closing Devices*.
3. The struct v4l2_capability changed dramatically. Note that also the size of the structure changed, which is encoded in the ioctl request code, thus older

V4L2 devices will respond with an EINVAL error code to the new *ioctl VIDIOC_QUERYCAP* ioctl.

There are new fields to identify the driver, a new RDS device function V4L2_CAP_RDS_CAPTURE, the V4L2_CAP_AUDIO flag indicates if the device has any audio connectors, another I/O capability V4L2_CAP_ASYNCIO can be flagged. In response to these changes the type field became a bit set and was merged into the flags field. V4L2_FLAG_TUNER was renamed to V4L2_CAP_TUNER, V4L2_CAP_VIDEO_OVERLAY replaced V4L2_FLAG_PREVIEW and V4L2_CAP_VBI_CAPTURE and V4L2_CAP_VBI_OUTPUT replaced V4L2_FLAG_DATA_SERVICE. V4L2_FLAG_READ and V4L2_FLAG_WRITE were merged into V4L2_CAP_READWRITE.

The redundant fields inputs, outputs and audios were removed. These properties can be determined as described in *Video Inputs and Outputs* and *Audio Inputs and Outputs*.

The somewhat volatile and therefore barely useful fields maxwidth, maxheight, minwidth, minheight, maxframerate were removed. This information is available as described in *Data Formats* and *Video Standards*.

V4L2_FLAG_SELECT was removed. We believe the select() function is important enough to require support of it in all V4L2 drivers exchanging data with applications. The redundant V4L2_FLAG_MONOCHROME flag was removed, this information is available as described in *Data Formats*.

4. In struct v4l2_input the assoc_audio field and the capability field and its only flag V4L2_INPUT_CAP_AUDIO was replaced by the new audioset field. Instead of linking one video input to one audio input this field reports all audio inputs this video input combines with.

   New fields are tuner (reversing the former link from tuners to video inputs), std and status.

   Accordingly struct v4l2_output lost its capability and assoc_audio fields. audioset, modulator and std where added instead.

5. The struct v4l2_audio field audio was renamed to index, for consistency with other structures. A new capability flag V4L2_AUDCAP_STEREO was added to indicated if the audio input in question supports stereo sound. V4L2_AUDCAP_EFFECTS and the corresponding V4L2_AUDMODE flags where removed. This can be easily implemented using controls. (However the same applies to AVL which is still there.)

   Again for consistency the struct v4l2_audioout field audio was renamed to index.

6. The struct v4l2_tuner input field was replaced by an index field, permitting devices with multiple tuners. The link between video inputs and tuners is now reversed, inputs point to their tuner. The std substructure became a simple set (more about this below) and moved into struct v4l2_input. A type field was added.

   Accordingly in struct v4l2_modulator the output was replaced by an index field.

In struct v4l2_frequency the `port` field was replaced by a `tuner` field containing the respective tuner or modulator index number. A tuner `type` field was added and the `reserved` field became larger for future extensions (satellite tuners in particular).

7. The idea of completely transparent video standards was dropped. Experience showed that applications must be able to work with video standards beyond presenting the user a menu. Instead of enumerating supported standards with an ioctl applications can now refer to standards by *v4l2_std_id* and symbols defined in the `videodev2.h` header file. For details see *Video Standards*. The *VIDIOC_G_STD* and *VIDIOC_S_STD* now take a pointer to this type as argument. *ioctl VIDIOC_QUERYSTD, VIDIOC_SUBDEV_QUERYSTD* was added to autodetect the received standard, if the hardware has this capability. In struct v4l2_standard an `index` field was added for *ioctl VIDIOC_ENUMSTD, VIDIOC_SUBDEV_ENUMSTD*. A *v4l2_std_id* field named `id` was added as machine readable identifier, also replacing the `transmission` field. The misleading `framerate` field was renamed to `frameperiod`. The now obsolete `colorstandard` information, originally needed to distguish between variations of standards, were removed.

   Struct `v4l2_enumstd` ceased to be. *ioctl VIDIOC_ENUMSTD, VIDIOC_SUBDEV_ENUMSTD* now takes a pointer to a struct v4l2_standard directly. The information which standards are supported by a particular video input or output moved into struct v4l2_input and struct v4l2_output fields named `std`, respectively.

8. The struct *v4l2_queryctrl* fields `category` and `group` did not catch on and/or were not implemented as expected and therefore removed.

9. The *VIDIOC_TRY_FMT* ioctl was added to negotiate data formats as with *VIDIOC_S_FMT*, but without the overhead of programming the hardware and regardless of I/O in progress.

   In struct v4l2_format the `fmt` union was extended to contain *`struct v4l2_window`*. All image format negotiations are now possible with `VIDIOC_G_FMT`, `VIDIOC_S_FMT` and `VIDIOC_TRY_FMT`; ioctl. The `VIDIOC_G_WIN` and `VIDIOC_S_WIN` ioctls to prepare for a video overlay were removed. The `type` field changed to type enum v4l2_buf_type and the buffer type names changed as follows.

Old defines	enum v4l2_buf_type
V4L2_BUF_TYPE_CAPTURE	V4L2_BUF_TYPE_VIDEO_CAPTURE
V4L2_BUF_TYPE_CODECIN	Omitted for now
V4L2_BUF_TYPE_CODECOUT	Omitted for now
V4L2_BUF_TYPE_EFFECTSIN	Omitted for now
V4L2_BUF_TYPE_EFFECTSIN2	Omitted for now
V4L2_BUF_TYPE_EFFECTSOUT	Omitted for now
V4L2_BUF_TYPE_VIDEOOUT	V4L2_BUF_TYPE_VIDEO_OUTPUT
-	V4L2_BUF_TYPE_VIDEO_OVERLAY
-	V4L2_BUF_TYPE_VBI_CAPTURE
-	V4L2_BUF_TYPE_VBI_OUTPUT
-	V4L2_BUF_TYPE_SLICED_VBI_CAPTURE
-	V4L2_BUF_TYPE_SLICED_VBI_OUTPUT
V4L2_BUF_TYPE_PRIVATE_BAS	V4L2_BUF_TYPE_PRIVATE (but this is deprecated)

10. In struct v4l2_fmtdesc a enum v4l2_buf_type field named `type` was added as in struct v4l2_format. The `VIDIOC_ENUM_FBUFFMT` ioctl is no longer needed and was removed. These calls can be replaced by *ioctl VIDIOC_ENUM_FMT* with type `V4L2_BUF_TYPE_VIDEO_OVERLAY`.

11. In *struct v4l2_pix_format* the `depth` field was removed, assuming applications which recognize the format by its four-character-code already know the color depth, and others do not care about it. The same rationale lead to the removal of the `V4L2_FMT_FLAG_COMPRESSED` flag. The `V4L2_FMT_FLAG_SWCONVECOMPRESSED` flag was removed because drivers are not supposed to convert images in kernel space. A user library of conversion functions should be provided instead. The `V4L2_FMT_FLAG_BYTESPERLINE` flag was redundant. Applications can set the `bytesperline` field to zero to get a reasonable default. Since the remaining flags were replaced as well, the `flags` field itself was removed.

    The interlace flags were replaced by a *enum v4l2_field* value in a newly added `field` field.

Old flag		*enum v4l2_field*
V4L2_FMT_FLAG_NOT_INTERLACED		?
V4L2_FMT_FLAG_INTERLACED V4L2_FMT_FLAG_COMBINED	=	V4L2_FIELD_INTERLACED
V4L2_FMT_FLAG_TOPFIELD V4L2_FMT_FLAG_ODDFIELD	=	V4L2_FIELD_TOP
V4L2_FMT_FLAG_BOTFIELD V4L2_FMT_FLAG_EVENFIELD	=	V4L2_FIELD_BOTTOM
-		V4L2_FIELD_SEQ_TB
-		V4L2_FIELD_SEQ_BT
-		V4L2_FIELD_ALTERNATE

    The color space flags were replaced by a *enum v4l2_colorspace* value in a newly added `colorspace` field, where one of `V4L2_COLORSPACE_SMPTE170M`,

V4L2_COLORSPACE_BT878, V4L2_COLORSPACE_470_SYSTEM_M or V4L2_COLORSPACE_470_SYSTEM_BG replaces V4L2_FMT_CS_601YUV.

12. In struct v4l2_requestbuffers the type field was properly defined as enum v4l2_buf_type. Buffer types changed as mentioned above. A new memory field of type enum v4l2_memory was added to distinguish between I/O methods using buffers allocated by the driver or the application. See *Input/Output* for details.

13. In struct v4l2_buffer the type field was properly defined as enum v4l2_buf_type. Buffer types changed as mentioned above. A field field of type *enum v4l2_field* was added to indicate if a buffer contains a top or bottom field. The old field flags were removed. Since no unadjusted system time clock was added to the kernel as planned, the timestamp field changed back from type stamp_t, an unsigned 64 bit integer expressing the sample time in nanoseconds, to struct timeval. With the addition of a second memory mapping method the offset field moved into union m, and a new memory field of type enum v4l2_memory was added to distinguish between I/O methods. See *Input/Output* for details.

    The V4L2_BUF_REQ_CONTIG flag was used by the V4L compatibility layer, after changes to this code it was no longer needed. The V4L2_BUF_ATTR_DEVICEMEM flag would indicate if the buffer was indeed allocated in device memory rather than DMA-able system memory. It was barely useful and so was removed.

14. In struct v4l2_framebuffer the base[3] array anticipating double- and triple-buffering in off-screen video memory, however without defining a synchronization mechanism, was replaced by a single pointer. The V4L2_FBUF_CAP_SCALEUP and V4L2_FBUF_CAP_SCALEDOWN flags were removed. Applications can determine this capability more accurately using the new cropping and scaling interface. The V4L2_FBUF_CAP_CLIPPING flag was replaced by V4L2_FBUF_CAP_LIST_CLIPPING and V4L2_FBUF_CAP_BITMAP_CLIPPING.

15. In *struct v4l2_clip* the x, y, width and height field moved into a c substructure of type *struct v4l2_rect*. The x and y fields were renamed to left and top, i. e. offsets to a context dependent origin.

16. In *struct v4l2_window* the x, y, width and height field moved into a w substructure as above. A field field of type *enum v4l2_field* was added to distinguish between field and frame (interlaced) overlay.

17. The digital zoom interface, including struct v4l2_zoomcap, struct v4l2_zoom, V4L2_ZOOM_NONCAP and V4L2_ZOOM_WHILESTREAMING was replaced by a new cropping and scaling interface. The previously unused struct v4l2_cropcap and struct v4l2_crop where redefined for this purpose. See *Image Cropping, Insertion and Scaling – the CROP API* for details.

18. In struct v4l2_vbi_format the SAMPLE_FORMAT field now contains a four-character-code as used to identify video image formats and V4L2_PIX_FMT_GREY replaces the V4L2_VBI_SF_UBYTE define. The reserved field was extended.

19. In struct v4l2_captureparm the type of the timeperframe field changed from unsigned long to struct v4l2_fract. This allows the accurate expression of

multiples of the NTSC-M frame rate 30000 / 1001. A new field `readbuffers` was added to control the driver behaviour in read I/O mode.

Similar changes were made to struct v4l2_outputparm.

20. The struct `v4l2_performance` and `VIDIOC_G_PERF` ioctl were dropped. Except when using the *read/write I/O method*, which is limited anyway, this information is already available to applications.

21. The example transformation from RGB to YCbCr color space in the old V4L2 documentation was inaccurate, this has been corrected in *Image Formats*.

V4L2 2003-06-19

1. A new capability flag `V4L2_CAP_RADIO` was added for radio devices. Prior to this change radio devices would identify solely by having exactly one tuner whose type field reads `V4L2_TUNER_RADIO`.

2. An optional driver access priority mechanism was added, see *Application Priority* for details.

3. The audio input and output interface was found to be incomplete.

   Previously the *VIDIOC_G_AUDIO* ioctl would enumerate the available audio inputs. An ioctl to determine the current audio input, if more than one combines with the current video input, did not exist. So `VIDIOC_G_AUDIO` was renamed to `VIDIOC_G_AUDIO_OLD`, this ioctl was removed on Kernel 2.6.39. The *ioctl VIDIOC_ENUMAUDIO* ioctl was added to enumerate audio inputs, while *VIDIOC_G_AUDIO* now reports the current audio input.

   The same changes were made to *VIDIOC_G_AUDOUT* and *VIDIOC_ENUMAUDOUT*.

   Until further the "videodev" module will automatically translate between the old and new ioctls, but drivers and applications must be updated to successfully compile again.

4. The *ioctl VIDIOC_OVERLAY* ioctl was incorrectly defined with write-read parameter. It was changed to write-only, while the write-read version was renamed to `VIDIOC_OVERLAY_OLD`. The old ioctl was removed on Kernel 2.6.39. Until further the "videodev" kernel module will automatically translate to the new version, so drivers must be recompiled, but not applications.

5. *Video Overlay Interface* incorrectly stated that clipping rectangles define regions where the video can be seen. Correct is that clipping rectangles define regions where *no* video shall be displayed and so the graphics surface can be seen.

6. The *VIDIOC_S_PARM* and *VIDIOC_S_CTRL* ioctls were defined with write-only parameter, inconsistent with other ioctls modifying their argument. They were changed to write-read, while a `_OLD` suffix was added to the write-only versions. The old ioctls were removed on Kernel 2.6.39. Drivers and applications assuming a constant parameter need an update.

V4L2 2003-11-05

1. In *RGB Formats* the following pixel formats were incorrectly transferred from Bill Dirks' V4L2 specification. Descriptions below refer to bytes in memory, in ascending address order.

Symbol	In this document prior to revision 0.5	Corrected
V4L2_PIX_FMT_RGB24	B, G, R	R, G, B
V4L2_PIX_FMT_BGR24	R, G, B	B, G, R
V4L2_PIX_FMT_RGB32	B, G, R, X	R, G, B, X
V4L2_PIX_FMT_BGR32	R, G, B, X	B, G, R, X

   The V4L2_PIX_FMT_BGR24 example was always correct.

   In *Image Properties* the mapping of the V4L VIDEO_PALETTE_RGB24 and VIDEO_PALETTE_RGB32 formats to V4L2 pixel formats was accordingly corrected.

2. Unrelated to the fixes above, drivers may still interpret some V4L2 RGB pixel formats differently. These issues have yet to be addressed, for details see *RGB Formats*.

V4L2 in Linux 2.6.6, 2004-05-09

1. The *ioctl VIDIOC_CROPCAP* ioctl was incorrectly defined with read-only parameter. It is now defined as write-read ioctl, while the read-only version was renamed to VIDIOC_CROPCAP_OLD. The old ioctl was removed on Kernel 2.6.39.

V4L2 in Linux 2.6.8

1. A new field input (former reserved[0]) was added to the struct v4l2_buffer. Purpose of this field is to alternate between video inputs (e. g. cameras) in step with the video capturing process. This function must be enabled with the new V4L2_BUF_FLAG_INPUT flag. The flags field is no longer read-only.

V4L2 spec erratum 2004-08-01

1. The return value of the *V4L2 open()* function was incorrectly documented.
2. Audio output ioctls end in -AUDOUT, not -AUDIOOUT.
3. In the Current Audio Input example the VIDIOC_G_AUDIO ioctl took the wrong argument.
4. The documentation of the *ioctl VIDIOC_QBUF, VIDIOC_DQBUF* and *VIDIOC_DQBUF* ioctls did not mention the struct v4l2_buffer memory field. It was also missing from examples. Also on the VIDIOC_DQBUF page the EIO error code was not documented.

V4L2 in Linux 2.6.14

1. A new sliced VBI interface was added. It is documented in *Sliced VBI Data Interface* and replaces the interface first proposed in V4L2 specification 0.8.

V4L2 in Linux 2.6.15

1. The *ioctl VIDIOC_LOG_STATUS* ioctl was added.
2. New video standards `V4L2_STD_NTSC_443`, `V4L2_STD_SECAM_LC`, `V4L2_STD_SECAM_DK` (a set of SECAM D, K and K1), and `V4L2_STD_ATSC` (a set of `V4L2_STD_ATSC_8_VSB` and `V4L2_STD_ATSC_16_VSB`) were defined. Note the `V4L2_STD_525_60` set now includes `V4L2_STD_NTSC_443`. See also *typedef v4l2_std_id*.
3. The `VIDIOC_G_COMP` and `VIDIOC_S_COMP` ioctl were renamed to `VIDIOC_G_MPEGCOMP` and `VIDIOC_S_MPEGCOMP` respectively. Their argument was replaced by a struct `v4l2_mpeg_compression` pointer. (The `VIDIOC_G_MPEGCOMP` and `VIDIOC_S_MPEGCOMP` ioctls where removed in Linux 2.6.25.)

V4L2 spec erratum 2005-11-27

The capture example in *Video Capture Example* called the *VIDIOC_S_CROP* ioctl without checking if cropping is supported. In the video standard selection example in *Video Standards* the *VIDIOC_S_STD* call used the wrong argument type.

V4L2 spec erratum 2006-01-10

1. The `V4L2_IN_ST_COLOR_KILL` flag in struct v4l2_input not only indicates if the color killer is enabled, but also if it is active. (The color killer disables color decoding when it detects no color in the video signal to improve the image quality.)
2. *VIDIOC_S_PARM* is a write-read ioctl, not write-only as stated on its reference page. The ioctl changed in 2003 as noted above.

V4L2 spec erratum 2006-02-03

1. In struct v4l2_captureparm and struct v4l2_outputparm the `timeperframe` field gives the time in seconds, not microseconds.

V4L2 spec erratum 2006-02-04

1. The clips field in *struct v4l2_window* must point to an array of *struct v4l2_clip*, not a linked list, because drivers ignore the *struct v4l2_clip*. next pointer.

V4L2 in Linux 2.6.17

1. New video standard macros were added: V4L2_STD_NTSC_M_KR (NTSC M South Korea), and the sets V4L2_STD_MN, V4L2_STD_B, V4L2_STD_GH and V4L2_STD_DK. The V4L2_STD_NTSC and V4L2_STD_SECAM sets now include V4L2_STD_NTSC_M_KR and V4L2_STD_SECAM_LC respectively.
2. A new V4L2_TUNER_MODE_LANG1_LANG2 was defined to record both languages of a bilingual program. The use of V4L2_TUNER_MODE_STEREO for this purpose is deprecated now. See the *VIDIOC_G_TUNER* section for details.

V4L2 spec erratum 2006-09-23 (Draft 0.15)

1. In various places V4L2_BUF_TYPE_SLICED_VBI_CAPTURE and V4L2_BUF_TYPE_SLICED_VBI_OUTPUT of the sliced VBI interface were not mentioned along with other buffer types.
2. In *VIDIOC_G_AUDIO* it was clarified that the struct v4l2_audio mode field is a flags field.
3. *ioctl VIDIOC_QUERYCAP* did not mention the sliced VBI and radio capability flags.
4. In *VIDIOC_G_FREQUENCY* it was clarified that applications must initialize the tuner type field of struct v4l2_frequency before calling *VIDIOC_S_FREQUENCY*.
5. The reserved array in struct v4l2_requestbuffers has 2 elements, not 32.
6. In *Video Output Interface* and *Raw VBI Data Interface* the device file names /dev/vout which never caught on were replaced by /dev/video.
7. With Linux 2.6.15 the possible range for VBI device minor numbers was extended from 224-239 to 224-255. Accordingly device file names /dev/vbi0 to /dev/vbi31 are possible now.

V4L2 in Linux 2.6.18

1. New ioctls *VIDIOC_G_EXT_CTRLS*, *VIDIOC_S_EXT_CTRLS* and *VIDIOC_TRY_EXT_CTRLS* were added, a flag to skip unsupported controls with *ioctls VIDIOC_QUERYCTRL, VIDIOC_QUERY_EXT_CTRL and VIDIOC_QUERYMENU*, new control types V4L2_CTRL_TYPE_INTEGER64 and V4L2_CTRL_TYPE_CTRL_CLASS (enum v4l2_ctrl_type), and new control flags V4L2_CTRL_FLAG_READ_ONLY, V4L2_CTRL_FLAG_UPDATE, V4L2_CTRL_FLAG_INACTIVE and V4L2_CTRL_FLAG_SLIDER (*Control Flags*). See *Extended Controls API* for details.

V4L2 in Linux 2.6.19

1. In struct v4l2_sliced_vbi_cap a buffer type field was added replacing a reserved field. Note on architectures where the size of enum types differs from int types the size of the structure changed. The *VIDIOC_G_SLICED_VBI_CAP* ioctl was redefined from being read-only to write-read. Applications must initialize the type field and clear the reserved fields now. These changes may *break the compatibility* with older drivers and applications.
2. The ioctls *ioctl VIDIOC_ENUM_FRAMESIZES* and *ioctl VIDIOC_ENUM_FRAMEINTERVALS* were added.
3. A new pixel format `V4L2_PIX_FMT_RGB444` (*RGB Formats*) was added.

V4L2 spec erratum 2006-10-12 (Draft 0.17)

1. `V4L2_PIX_FMT_HM12` (*Reserved Image Formats*) is a YUV 4:2:0, not 4:2:2 format.

V4L2 in Linux 2.6.21

1. The `videodev2.h` header file is now dual licensed under GNU General Public License version two or later, and under a 3-clause BSD-style license.

V4L2 in Linux 2.6.22

1. Two new field orders `V4L2_FIELD_INTERLACED_TB` and `V4L2_FIELD_INTERLACED_BT` were added. See *enum v4l2_field* for details.
2. Three new clipping/blending methods with a global or straight or inverted local alpha value were added to the video overlay interface. See the description of the *VIDIOC_G_FBUF* and *VIDIOC_S_FBUF* ioctls for details.

   A new `global_alpha` field was added to *struct v4l2_window*, extending the structure. This may **break compatibility** with applications using a *struct v4l2_window* directly. However the *VIDIOC_G/S/TRY_FMT* ioctls, which take a pointer to a struct v4l2_format parent structure with padding bytes at the end, are not affected.
3. The format of the `chromakey` field in *struct v4l2_window* changed from "host order RGB32" to a pixel value in the same format as the framebuffer. This may **break compatibility** with existing applications. Drivers supporting the "host order RGB32" format are not known.

V4L2 in Linux 2.6.24

1. The pixel formats V4L2_PIX_FMT_PAL8, V4L2_PIX_FMT_YUV444, V4L2_PIX_FMT_YUV555, V4L2_PIX_FMT_YUV565 and V4L2_PIX_FMT_YUV32 were added.

V4L2 in Linux 2.6.25

1. The pixel formats *V4L2_PIX_FMT_Y16* and *V4L2_PIX_FMT_SBGGR16* were added.
2. New *controls* V4L2_CID_POWER_LINE_FREQUENCY, V4L2_CID_HUE_AUTO, V4L2_CID_WHITE_BALANCE_TEMPERATURE, V4L2_CID_SHARPNESS and V4L2_CID_BACKLIGHT_COMPENSATION were added. The controls V4L2_CID_BLACK_LEVEL, V4L2_CID_WHITENESS, V4L2_CID_HCENTER and V4L2_CID_VCENTER were deprecated.
3. A *Camera controls class* was added, with the new controls V4L2_CID_EXPOSURE_AUTO, V4L2_CID_EXPOSURE_ABSOLUTE, V4L2_CID_EXPOSURE_AUTO_PRIORITY, V4L2_CID_PAN_RELATIVE, V4L2_CID_TILT_RELATIVE, V4L2_CID_PAN_RESET, V4L2_CID_TILT_RESET, V4L2_CID_PAN_ABSOLUTE, V4L2_CID_TILT_ABSOLUTE, V4L2_CID_FOCUS_ABSOLUTE, V4L2_CID_FOCUS_RELATIVE and V4L2_CID_FOCUS_AUTO.
4. The VIDIOC_G_MPEGCOMP and VIDIOC_S_MPEGCOMP ioctls, which were superseded by the *extended controls* interface in Linux 2.6.18, where finally removed from the videodev2.h header file.

V4L2 in Linux 2.6.26

1. The pixel formats V4L2_PIX_FMT_Y16 and V4L2_PIX_FMT_SBGGR16 were added.
2. Added user controls V4L2_CID_CHROMA_AGC and V4L2_CID_COLOR_KILLER.

V4L2 in Linux 2.6.27

1. The *ioctl VIDIOC_S_HW_FREQ_SEEK* ioctl and the V4L2_CAP_HW_FREQ_SEEK capability were added.
2. The pixel formats V4L2_PIX_FMT_YVYU, V4L2_PIX_FMT_PCA501, V4L2_PIX_FMT_PCA505, V4L2_PIX_FMT_PCA508, V4L2_PIX_FMT_PCA561, V4L2_PIX_FMT_SGBRG8, V4L2_PIX_FMT_PAC207 and V4L2_PIX_FMT_PJPG were added.

V4L2 in Linux 2.6.28

1. Added V4L2_MPEG_AUDIO_ENCODING_AAC and V4L2_MPEG_AUDIO_ENCODING_AC3 MPEG audio encodings.
2. Added V4L2_MPEG_VIDEO_ENCODING_MPEG_4_AVC MPEG video encoding.
3. The pixel formats V4L2_PIX_FMT_SGRBG10 and V4L2_PIX_FMT_SGRBG10DPCM8 were added.

V4L2 in Linux 2.6.29

1. The VIDIOC_G_CHIP_IDENT ioctl was renamed to VIDIOC_G_CHIP_IDENT_OLD and VIDIOC_DBG_G_CHIP_IDENT was introduced in its place. The old struct v4l2_chip_ident was renamed to struct v4l2_chip_ident_old.
2. The pixel formats V4L2_PIX_FMT_VYUY, V4L2_PIX_FMT_NV16 and V4L2_PIX_FMT_NV61 were added.
3. Added camera controls V4L2_CID_ZOOM_ABSOLUTE, V4L2_CID_ZOOM_RELATIVE, V4L2_CID_ZOOM_CONTINUOUS and V4L2_CID_PRIVACY.

V4L2 in Linux 2.6.30

1. New control flag V4L2_CTRL_FLAG_WRITE_ONLY was added.
2. New control V4L2_CID_COLORFX was added.

V4L2 in Linux 2.6.32

1. In order to be easier to compare a V4L2 API and a kernel version, now V4L2 API is numbered using the Linux Kernel version numeration.
2. Finalized the RDS capture API. See *RDS Interface* for more information.
3. Added new capabilities for modulators and RDS encoders.
4. Add description for libv4l API.
5. Added support for string controls via new type V4L2_CTRL_TYPE_STRING.
6. Added V4L2_CID_BAND_STOP_FILTER documentation.
7. Added FM Modulator (FM TX) Extended Control Class: V4L2_CTRL_CLASS_FM_TX and their Control IDs.
8. Added FM Receiver (FM RX) Extended Control Class: V4L2_CTRL_CLASS_FM_RX and their Control IDs.
9. Added Remote Controller chapter, describing the default Remote Controller mapping for media devices.

V4L2 in Linux 2.6.33

1. Added support for Digital Video timings in order to support HDTV receivers and transmitters.

V4L2 in Linux 2.6.34

1. Added `V4L2_CID_IRIS_ABSOLUTE` and `V4L2_CID_IRIS_RELATIVE` controls to the *Camera controls class*.

V4L2 in Linux 2.6.37

1. Remove the vtx (videotext/teletext) API. This API was no longer used and no hardware exists to verify the API. Nor were any userspace applications found that used it. It was originally scheduled for removal in 2.6.35.

V4L2 in Linux 2.6.39

1. The old VIDIOC_*_OLD symbols and V4L1 support were removed.
2. Multi-planar API added. Does not affect the compatibility of current drivers and applications. See *multi-planar API* for details.

V4L2 in Linux 3.1

1. VIDIOC_QUERYCAP now returns a per-subsystem version instead of a per-driver one.

   Standardize an error code for invalid ioctl.

   Added V4L2_CTRL_TYPE_BITMASK.

V4L2 in Linux 3.2

1. V4L2_CTRL_FLAG_VOLATILE was added to signal volatile controls to userspace.
2. Add selection API for extended control over cropping and composing. Does not affect the compatibility of current drivers and applications. See *selection API* for details.

V4L2 in Linux 3.3

1. Added V4L2_CID_ALPHA_COMPONENT control to the *User controls class*.
2. Added the device_caps field to struct v4l2_capabilities and added the new V4L2_CAP_DEVICE_CAPS capability.

V4L2 in Linux 3.4

1. Added *JPEG compression control class*.
2. Extended the DV Timings API: *ioctl VIDIOC_ENUM_DV_TIMINGS, VIDIOC_SUBDEV_ENUM_DV_TIMINGS*, *ioctl VIDIOC_QUERY_DV_TIMINGS* and *ioctl VIDIOC_DV_TIMINGS_CAP, VIDIOC_SUBDEV_DV_TIMINGS_CAP*.

V4L2 in Linux 3.5

1. Added integer menus, the new type will be V4L2_CTRL_TYPE_INTEGER_MENU.
2. Added selection API for V4L2 subdev interface: *ioctl VIDIOC_SUBDEV_G_SELECTION, VIDIOC_SUBDEV_S_SELECTION* and *VIDIOC_SUBDEV_S_SELECTION*.
3. Added V4L2_COLORFX_ANTIQUE, V4L2_COLORFX_ART_FREEZE, V4L2_COLORFX_AQUA, V4L2_COLORFX_SILHOUETTE, V4L2_COLORFX_SOLARIZATION, V4L2_COLORFX_VIVID and V4L2_COLORFX_ARBITRARY_CBCR menu items to the V4L2_CID_COLORFX control.
4. Added V4L2_CID_COLORFX_CBCR control.
5. Added camera controls V4L2_CID_AUTO_EXPOSURE_BIAS, V4L2_CID_AUTO_N_PRESET_WHITE_BALANCE, V4L2_CID_IMAGE_STABILIZATION, V4L2_CID_ISO_SENSITIVITY, V4L2_CID_ISO_SENSITIVITY_AUTO, V4L2_CID_EXPOSURE_METERING, V4L2_CID_SCENE_MODE, V4L2_CID_3A_LOCK, V4L2_CID_AUTO_FOCUS_START, V4L2_CID_AUTO_FOCUS_STOP, V4L2_CID_AUTO_FOCUS_STATUS and V4L2_CID_AUTO_FOCUS_RANGE.

V4L2 in Linux 3.6

1. Replaced input in struct v4l2_buffer by reserved2 and removed V4L2_BUF_FLAG_INPUT.
2. Added V4L2_CAP_VIDEO_M2M and V4L2_CAP_VIDEO_M2M_MPLANE capabilities.
3. Added support for frequency band enumerations: *ioctl VIDIOC_ENUM_FREQ_BANDS*.

V4L2 in Linux 3.9

1. Added timestamp types to `flags` field in struct v4l2_buffer. See *Buffer Flags*.
2. Added `V4L2_EVENT_CTRL_CH_RANGE` control event changes flag. See *Control Changes*.

V4L2 in Linux 3.10

1. Removed obsolete and unused DV_PRESET ioctls VIDIOC_G_DV_PRESET, VIDIOC_S_DV_PRESET, VIDIOC_QUERY_DV_PRESET and VIDIOC_ENUM_DV_PRESET. Remove the related v4l2_input/output capability flags V4L2_IN_CAP_PRESETS and V4L2_OUT_CAP_PRESETS.
2. Added new debugging ioctl *ioctl VIDIOC_DBG_G_CHIP_INFO*.

V4L2 in Linux 3.11

1. Remove obsolete `VIDIOC_DBG_G_CHIP_IDENT` ioctl.

V4L2 in Linux 3.14

1. In *`struct v4l2_rect`*, the type of `width` and `height` fields changed from _s32 to _u32.

V4L2 in Linux 3.15

1. Added Software Defined Radio (SDR) Interface.

V4L2 in Linux 3.16

1. Added event V4L2_EVENT_SOURCE_CHANGE.

V4L2 in Linux 3.17

1. Extended *`struct v4l2_pix_format`*. Added format flags.
2. Added compound control types and *VIDIOC_QUERY_EXT_CTRL*.

V4L2 in Linux 3.18

1. Added V4L2_CID_PAN_SPEED and V4L2_CID_TILT_SPEED camera controls.

V4L2 in Linux 3.19

1. Rewrote Colorspace chapter, added new *enum v4l2_ycbcr_encoding* and *enum v4l2_quantization* fields to *struct v4l2_pix_format*, *struct v4l2_pix_format_mplane* and *struct v4l2_mbus_framefmt*.

V4L2 in Linux 4.4

1. Renamed V4L2_TUNER_ADC to V4L2_TUNER_SDR. The use of V4L2_TUNER_ADC is deprecated now.
2. Added V4L2_CID_RF_TUNER_RF_GAIN RF Tuner control.
3. Added transmitter support for Software Defined Radio (SDR) Interface.

Relation of V4L2 to other Linux multimedia APIs

X Video Extension

The X Video Extension (abbreviated XVideo or just Xv) is an extension of the X Window system, implemented for example by the XFree86 project. Its scope is similar to V4L2, an API to video capture and output devices for X clients. Xv allows applications to display live video in a window, send window contents to a TV output, and capture or output still images in XPixmaps[1]. With their implementation XFree86 makes the extension available across many operating systems and architectures.

Because the driver is embedded into the X server Xv has a number of advantages over the V4L2 *video overlay interface*. The driver can easily determine the overlay target, i. e. visible graphics memory or off-screen buffers for a destructive overlay. It can program the RAMDAC for a non-destructive overlay, scaling or color-keying, or the clipping functions of the video capture hardware, always in sync with drawing operations or windows moving or changing their stacking order.

To combine the advantages of Xv and V4L a special Xv driver exists in XFree86 and XOrg, just programming any overlay capable Video4Linux device it finds. To enable it /etc/X11/XF86Config must contain these lines:

```
Section "Module"
    Load "v4l"
EndSection
```

As of XFree86 4.2 this driver still supports only V4L ioctls, however it should work just fine with all V4L2 devices through the V4L2 backward-compatibility layer.

[1] This is not implemented in XFree86.

Since V4L2 permits multiple opens it is possible (if supported by the V4L2 driver) to capture video while an X client requested video overlay. Restrictions of simultaneous capturing and overlay are discussed in *Video Overlay Interface* apply.

Only marginally related to V4L2, XFree86 extended Xv to support hardware YUV to RGB conversion and scaling for faster video playback, and added an interface to MPEG-2 decoding hardware. This API is useful to display images captured with V4L2 devices.

Digital Video

V4L2 does not support digital terrestrial, cable or satellite broadcast. A separate project aiming at digital receivers exists. You can find its homepage at https://linuxtv.org. The Linux DVB API has no connection to the V4L2 API except that drivers for hybrid hardware may support both.

Audio Interfaces

[to do - OSS/ALSA]

Experimental API Elements

The following V4L2 API elements are currently experimental and may change in the future.

- *ioctl VIDIOC_DBG_G_REGISTER, VIDIOC_DBG_S_REGISTER* and *VIDIOC_DBG_S_REGISTER* ioctls.
- *ioctl VIDIOC_DBG_G_CHIP_INFO* ioctl.

Obsolete API Elements

The following V4L2 API elements were superseded by new interfaces and should not be implemented in new drivers.

- `VIDIOC_G_MPEGCOMP` and `VIDIOC_S_MPEGCOMP` ioctls. Use Extended Controls, *Extended Controls API*.
- VIDIOC_G_DV_PRESET, VIDIOC_S_DV_PRESET, VIDIOC_ENUM_DV_PRESETS and VIDIOC_QUERY_DV_PRESET ioctls. Use the DV Timings API (*Digital Video (DV) Timings*).
- `VIDIOC_SUBDEV_G_CROP` and `VIDIOC_SUBDEV_S_CROP` ioctls. Use `VIDIOC_SUBDEV_G_SELECTION` and `VIDIOC_SUBDEV_S_SELECTION`, *ioctl VIDIOC_SUBDEV_G_SELECTION, VIDIOC_SUBDEV_S_SELECTION*.

8.2.7 Function Reference

V4L2 close()

Name

v4l2-close - Close a V4L2 device

Synopsis

```
#include <unistd.h>
```

int **close**(int fd)

Arguments

fd
File descriptor returned by *open()*.

Description

Closes the device. Any I/O in progress is terminated and resources associated with the file descriptor are freed. However data format parameters, current input or output, control values or other properties remain unchanged.

Return Value

The function returns 0 on success, -1 on failure and the `errno` is set appropriately. Possible error codes:

EBADF
`fd` is not a valid open file descriptor.

V4L2 ioctl()

Name

v4l2-ioctl - Program a V4L2 device

Synopsis

```
#include <sys/ioctl.h>
```

`int ioctl(int fd, int request, void *argp)`

Arguments

fd
: File descriptor returned by *open()*.

request
: V4L2 ioctl request code as defined in the `videodev2.h` header file, for example VIDIOC_QUERYCAP.

argp
: Pointer to a function parameter, usually a structure.

Description

The *ioctl()* function is used to program V4L2 devices. The argument `fd` must be an open file descriptor. An ioctl `request` has encoded in it whether the argument is an input, output or read/write parameter, and the size of the argument `argp` in bytes. Macros and defines specifying V4L2 ioctl requests are located in the `videodev2.h` header file. Applications should use their own copy, not include the version in the kernel sources on the system they compile on. All V4L2 ioctl requests, their respective function and parameters are specified in *Function Reference*.

Return Value

On success 0 is returned, on error -1 and the `errno` variable is set appropriately. The generic error codes are described at the *Generic Error Codes* chapter.

When an ioctl that takes an output or read/write parameter fails, the parameter remains unmodified.

ioctl VIDIOC_CREATE_BUFS

Name

VIDIOC_CREATE_BUFS - Create buffers for Memory Mapped or User Pointer or DMA Buffer I/O

Synopsis

VIDIOC_CREATE_BUFS

`int ioctl(int fd, VIDIOC_CREATE_BUFS, struct v4l2_create_buffers *argp)`

Arguments

fd
: File descriptor returned by *open()*.

argp
: Pointer to struct *v4l2_create_buffers*.

Description

This ioctl is used to create buffers for *memory mapped* or *user pointer* or *DMA buffer* I/O. It can be used as an alternative or in addition to the *ioctl VIDIOC_REQBUFS* ioctl, when a tighter control over buffers is required. This ioctl can be called multiple times to create buffers of different sizes.

To allocate the device buffers applications must initialize the relevant fields of the struct *v4l2_create_buffers* structure. The `count` field must be set to the number of requested buffers, the `memory` field specifies the requested I/O method and the `reserved` array must be zeroed.

The `format` field specifies the image format that the buffers must be able to handle. The application has to fill in this struct *v4l2_format*. Usually this will be done using the *VIDIOC_TRY_FMT* or *VIDIOC_G_FMT* ioctls to ensure that the requested format is supported by the driver. Based on the format's `type` field the requested buffer size (for single-planar) or plane sizes (for multi-planar formats) will be used for the allocated buffers. The driver may return an error if the size(s) are not supported by the hardware (usually because they are too small).

The buffers created by this ioctl will have as minimum size the size defined by the `format.pix.sizeimage` field (or the corresponding fields for other format types). Usually if the `format.pix.sizeimage` field is less than the minimum required for the given format, then an error will be returned since drivers will typically not allow this. If it is larger, then the value will be used as-is. In other words, the driver may reject the requested size, but if it is accepted the driver will use it unchanged.

When the ioctl is called with a pointer to this structure the driver will attempt to allocate up to the requested number of buffers and store the actual number allocated and the starting index in the `count` and the `index` fields respectively. On return `count` can be smaller than the number requested.

type **v4l2_create_buffers**

Table 106: struct v4l2_create_buffers

__u32	index	The starting buffer index, returned by the driver.
__u32	count	The number of buffers requested or granted. If count == 0, then *ioctl VIDIOC_CREATE_BUFS* will set index to the current number of created buffers, and it will check the validity of memory and format.type. If those are invalid -1 is returned and errno is set to EINVAL error code, otherwise *ioctl VIDIOC_CREATE_BUFS* returns 0. It will never set errno to EBUSY error code in this particular case.
__u32	memory	Applications set this field to V4L2_MEMORY_MMAP, V4L2_MEMORY_DMABUF or V4L2_MEMORY_USERPTR. See v4l2_memory
struct *v4l2_format*	format	Filled in by the application, preserved by the driver.
__u32	capabilities	Set by the driver. If 0, then the driver doesn't support capabilities. In that case all you know is that the driver is guaranteed to support V4L2_MEMORY_MMAP and *might* support other v4l2_memory types. It will not support any other capabilities. See *here* for a list of the capabilities. If you want to just query the capabilities without making any other changes, then set count to 0, memory to V4L2_MEMORY_MMAP and format.type to the buffer type.
__u32	reserved[7]	A place holder for future extensions. Drivers and applications must set the array to zero.

Return Value

On success 0 is returned, on error -1 and the errno variable is set appropriately. The generic error codes are described at the *Generic Error Codes* chapter.

ENOMEM
: No memory to allocate buffers for *memory mapped* I/O.

EINVAL
: The buffer type (format.type field), requested I/O method (memory) or format (format field) is not valid.

ioctl VIDIOC_CROPCAP

Name

VIDIOC_CROPCAP - Information about the video cropping and scaling abilities

Synopsis

VIDIOC_CROPCAP

`int ioctl(int fd, VIDIOC_CROPCAP, struct v4l2_cropcap *argp)`

Arguments

fd
File descriptor returned by *open()*.

argp
Pointer to struct *v4l2_cropcap*.

Description

Applications use this function to query the cropping limits, the pixel aspect of images and to calculate scale factors. They set the `type` field of a v4l2_cropcap structure to the respective buffer (stream) type and call the *ioctl VIDIOC_CROPCAP* ioctl with a pointer to this structure. Drivers fill the rest of the structure. The results are constant except when switching the video standard. Remember this switch can occur implicit when switching the video input or output.

This ioctl must be implemented for video capture or output devices that support cropping and/or scaling and/or have non-square pixels, and for overlay devices.

type **v4l2_cropcap**

Table 107: struct v4l2_cropcap

__u32	type	Type of the data stream, set by the application. Only these types are valid here: V4L2_BUF_TYPE_VIDEO_CAPTURE, V4L2_BUF_TYPE_VIDEO_CAPTURE_MPLANE, V4L2_BUF_TYPE_VIDEO_OUTPUT, V4L2_BUF_TYPE_VIDEO_OUTPUT_MPLANE and V4L2_BUF_TYPE_VIDEO_OVERLAY. See *v4l2_buf_type* and the note below.
struct *v4l2_rect*	bounds	Defines the window within capturing or output is possible, this may exclude for example the horizontal and vertical blanking areas. The cropping rectangle cannot exceed these limits. Width and height are defined in pixels, the driver writer is free to choose origin and units of the coordinate system in the analog domain.
struct *v4l2_rect*	defrect	Default cropping rectangle, it shall cover the "whole picture". Assuming pixel aspect 1/1 this could be for example a 640 × 480 rectangle for NTSC, a 768 × 576 rectangle for PAL and SECAM centered over the active picture area. The same co-ordinate system as for bounds is used.
struct *v4l2_fract*	pixelaspect	This is the pixel aspect (y / x) when no scaling is applied, the ratio of the actual sampling frequency and the frequency required to get square pixels. When cropping coordinates refer to square pixels, the driver sets pixelaspect to 1/1. Other common values are 54/59 for PAL and SECAM, 11/10 for NTSC sampled according to [*ITU BT.601*].

Note: Unfortunately in the case of multiplanar buffer types (V4L2_BUF_TYPE_VIDEO_CAPTURE_MPLANE and V4L2_BUF_TYPE_VIDEO_OUTPUT_MPLANE) this API was messed up with regards to how the *v4l2_cropcap* type field should be filled in. Some drivers only accepted the _MPLANE buffer type while other drivers only accepted a non-multiplanar buffer type (i.e. without the _MPLANE at the end).

Starting with kernel 4.13 both variations are allowed.

Table 108: struct v4l2_rect

__s32	left	Horizontal offset of the top, left corner of the rectangle, in pixels.
__s32	top	Vertical offset of the top, left corner of the rectangle, in pixels.
__u32	width	Width of the rectangle, in pixels.
__u32	height	Height of the rectangle, in pixels.

Return Value

On success 0 is returned, on error -1 and the `errno` variable is set appropriately. The generic error codes are described at the *Generic Error Codes* chapter.

EINVAL
The struct *v4l2_cropcap* `type` is invalid.

ENODATA
Cropping is not supported for this input or output.

ioctl VIDIOC_DBG_G_CHIP_INFO

Name

VIDIOC_DBG_G_CHIP_INFO - Identify the chips on a TV card

Synopsis

VIDIOC_DBG_G_CHIP_INFO

`int ioctl(int fd, VIDIOC_DBG_G_CHIP_INFO, struct v4l2_dbg_chip_info *argp)`

Arguments

fd
File descriptor returned by *open()*.

argp
Pointer to struct *v4l2_dbg_chip_info*.

Description

Note: This is an *Experimental API Elements* interface and may change in the future.

For driver debugging purposes this ioctl allows test applications to query the driver about the chips present on the TV card. Regular applications must not use it. When you found a chip specific bug, please contact the linux-media mailing list (https://linuxtv.org/lists.php) so it can be fixed.

Additionally the Linux kernel must be compiled with the `CONFIG_VIDEO_ADV_DEBUG` option to enable this ioctl.

To query the driver applications must initialize the `match.type` and `match.addr` or `match.name` fields of a struct *`v4l2_dbg_chip_info`* and call *ioctl VIDIOC_DBG_G_CHIP_INFO* with a pointer to this structure. On success the driver stores information about the selected chip in the `name` and `flags` fields.

When `match.type` is `V4L2_CHIP_MATCH_BRIDGE`, `match.addr` selects the nth bridge 'chip' on the TV card. You can enumerate all chips by starting at zero and incrementing `match.addr` by one until *ioctl VIDIOC_DBG_G_CHIP_INFO* fails with an `EINVAL` error code. The number zero always selects the bridge chip itself, e. g. the chip connected to the PCI or USB bus. Non-zero numbers identify specific parts of the bridge chip such as an AC97 register block.

When `match.type` is `V4L2_CHIP_MATCH_SUBDEV`, `match.addr` selects the nth subdevice. This allows you to enumerate over all sub-devices.

On success, the `name` field will contain a chip name and the `flags` field will contain `V4L2_CHIP_FL_READABLE` if the driver supports reading registers from the device or `V4L2_CHIP_FL_WRITABLE` if the driver supports writing registers to the device.

We recommended the v4l2-dbg utility over calling this ioctl directly. It is available from the LinuxTV v4l-dvb repository; see https://linuxtv.org/repo/ for access instructions.

Table 109: struct v4l2_dbg_match

__u32	type	See *Chip Match Types* for a list of possible types.
union {	(anonymous)	
__u32	addr	Match a chip by this number, interpreted according to the type field.
char	name[32]	Match a chip by this name, interpreted according to the type field. Currently unused.
}		

type **v4l2_dbg_chip_info**

Table 110: struct v4l2_dbg_chip_info

struct v4l2_dbg_match	match	How to match the chip, see *struct v4l2_dbg_match*.
char	name[32]	The name of the chip.
__u32	flags	Set by the driver. If V4L2_CHIP_FL_READABLE is set, then the driver supports reading registers from the device. If V4L2_CHIP_FL_WRITABLE is set, then it supports writing registers.
__u32	reserved[8]	Reserved fields, both application and driver must set these to 0.

Table 111: Chip Match Types

V4L2_CHIP_MATCH_BRIDGE	0	Match the nth chip on the card, zero for the bridge chip. Does not match sub-devices.
V4L2_CHIP_MATCH_SUBDEV	4	Match the nth sub-device.

Return Value

On success 0 is returned, on error -1 and the errno variable is set appropriately. The generic error codes are described at the *Generic Error Codes* chapter.

EINVAL
: The match_type is invalid or no device could be matched.

ioctl VIDIOC_DBG_G_REGISTER, VIDIOC_DBG_S_REGISTER

Name

VIDIOC_DBG_G_REGISTER - VIDIOC_DBG_S_REGISTER - Read or write hardware registers

Synopsis

VIDIOC_DBG_G_REGISTER

`int ioctl(int fd, VIDIOC_DBG_G_REGISTER, struct v4l2_dbg_register *argp)`

VIDIOC_DBG_S_REGISTER

`int ioctl(int fd, VIDIOC_DBG_S_REGISTER, const struct v4l2_dbg_register *argp)`

Arguments

fd
File descriptor returned by *open()*.

argp
Pointer to struct *v4l2_dbg_register*.

Description

Note: This is an *Experimental API Elements* interface and may change in the future.

For driver debugging purposes these ioctls allow test applications to access hardware registers directly. Regular applications must not use them.

Since writing or even reading registers can jeopardize the system security, its stability and damage the hardware, both ioctls require superuser privileges. Additionally the Linux kernel must be compiled with the `CONFIG_VIDEO_ADV_DEBUG` option to enable these ioctls.

To write a register applications must initialize all fields of a struct *v4l2_dbg_register* except for `size` and call `VIDIOC_DBG_S_REGISTER` with a pointer to this structure. The `match.type` and `match.addr` or `match.name` fields select a chip on the TV card, the `reg` field specifies a register number and the `val` field the value to be written into the register.

To read a register applications must initialize the `match.type`, `match.addr` or `match.name` and `reg` fields, and call `VIDIOC_DBG_G_REGISTER` with a pointer to this structure. On success the driver stores the register value in the `val` field and the size (in bytes) of the value in `size`.

When `match.type` is `V4L2_CHIP_MATCH_BRIDGE`, `match.addr` selects the nth non-sub-device chip on the TV card. The number zero always selects the host chip, e. g. the chip connected to the PCI or USB bus. You can find out which chips are present with the *ioctl VIDIOC_DBG_G_CHIP_INFO* ioctl.

When `match.type` is `V4L2_CHIP_MATCH_SUBDEV`, `match.addr` selects the nth sub-device.

These ioctls are optional, not all drivers may support them. However when a driver supports these ioctls it must also support *ioctl VIDIOC_DBG_G_CHIP_INFO*. Conversely it may support `VIDIOC_DBG_G_CHIP_INFO` but not these ioctls.

`VIDIOC_DBG_G_REGISTER` and `VIDIOC_DBG_S_REGISTER` were introduced in Linux 2.6.21, but their API was changed to the one described here in kernel 2.6.29.

We recommended the v4l2-dbg utility over calling these ioctls directly. It is available from the LinuxTV v4l-dvb repository; see https://linuxtv.org/repo/ for access instructions.

type **`v4l2_dbg_match`**

Table 112: struct v4l2_dbg_match

__u32	`type`	See *Chip Match Types* for a list of possible types.
union {	(anonymous)	
__u32	`addr`	Match a chip by this number, interpreted according to the `type` field.
char	`name[32]`	Match a chip by this name, interpreted according to the `type` field. Currently unused.
}		

type **`v4l2_dbg_register`**

Table 113: struct v4l2_dbg_register

struct v4l2_dbg_match	`match`	How to match the chip, see *`v4l2_dbg_match`*.
__u32	`size`	The register size in bytes.
__u64	`reg`	A register number.
__u64	`val`	The value read from, or to be written into the register.

Table 114: Chip Match Types

V4L2_CHIP_MATCH_BRIDGE	0	Match the nth chip on the card, zero for the bridge chip. Does not match sub-devices.
V4L2_CHIP_MATCH_SUBDEV	4	Match the nth sub-device.

Return Value

On success 0 is returned, on error -1 and the `errno` variable is set appropriately. The generic error codes are described at the *Generic Error Codes* chapter.

EPERM
Insufficient permissions. Root privileges are required to execute these ioctls.

ioctl VIDIOC_DECODER_CMD, VIDIOC_TRY_DECODER_CMD

Name

VIDIOC_DECODER_CMD - VIDIOC_TRY_DECODER_CMD - Execute an decoder command

Synopsis

VIDIOC_DECODER_CMD

`int ioctl(int fd, VIDIOC_DECODER_CMD, struct v4l2_decoder_cmd *argp)`

VIDIOC_TRY_DECODER_CMD

`int ioctl(int fd, VIDIOC_TRY_DECODER_CMD, struct v4l2_decoder_cmd *argp)`

Arguments

fd
File descriptor returned by *open()*.

argp
pointer to struct *v4l2_decoder_cmd*.

Description

These ioctls control an audio/video (usually MPEG-) decoder. VIDIOC_DECODER_CMD sends a command to the decoder, VIDIOC_TRY_DECODER_CMD can be used to try a command without actually executing it. To send a command applications must initialize all fields of a struct *v4l2_decoder_cmd* and call VIDIOC_DECODER_CMD or VIDIOC_TRY_DECODER_CMD with a pointer to this structure.

The cmd field must contain the command code. Some commands use the flags field for additional information.

A *write()* or *ioctl VIDIOC_STREAMON, VIDIOC_STREAMOFF* call sends an implicit START command to the decoder if it has not been started yet. Applies to both queues of mem2mem decoders.

A *close()* or *VIDIOC_STREAMOFF* call of a streaming file descriptor sends an implicit immediate STOP command to the decoder, and all buffered data is discarded. Applies to both queues of mem2mem decoders.

In principle, these ioctls are optional, not all drivers may support them. They were introduced in Linux 3.3. They are, however, mandatory for stateful mem2mem decoders (as further documented in *Memory-to-Memory Stateful Video Decoder Interface*).

type **v4l2_decoder_cmd**

Table 115: struct v4l2_decoder_cmd

__u32	cmd		The decoder command, see *Decoder Commands*.

continues on next page

Table 115 – continued from previous page

__u32	flags		Flags to go with the command. If no flags are defined for this command, drivers and applications must set this field to zero.
union {	(anonymous)		
struct	start		Structure containing additional data for the V4L2_DEC_CMD_START command.

continues on next page

Table 115 – continued from previous page

	__s32	speed	Playback speed and direction. The playback speed is defined as speed/1000 of the normal speed. So 1000 is normal playback. Negative numbers denote reverse playback, so -1000 does reverse playback at normal speed. Speeds -1, 0 and 1 have special meanings: speed 0 is shorthand for 1000 (normal playback). A speed of 1 steps just one frame for-

Table 115 – continued from previous page

	__u32	format	Format restrictions. This field is set by the driver, not the application. Possible values are V4L2_DEC_START_FMT_NONE if there are no format restrictions or V4L2_DEC_START_FMT_GOP if the decoder operates on full GOPs (*Group Of Pictures*). This is usually the case for reverse playback: the decoder needs full GOPs, which it can then play in reverse order. So to implement reverse playback the appli-

Table 115 – continued from previous page

struct	stop		Structure containing additional data for the V4L2_DEC_CMD_STOP command.
	__u64	pts	Stop playback at this pts or immediately if the playback is already past that timestamp. Leave to 0 if you want to stop after the last frame was decoded.
struct	raw		

continues on next page

Table 115 – continued from previous page

	__u32	data[1	Reserved for future extensions. Drivers and applications must set the array to zero.
}			

Table 116: Decoder Commands

V4L2_DEC_CMD_START	0	Start the decoder. When the decoder is already running or paused, this command will just change the playback speed. That means that calling V4L2_DEC_CMD_START when the decoder was paused will *not* resume the decoder. You have to explicitly call V4L2_DEC_CMD_RESUME for that. This command has one flag: V4L2_DEC_CMD_START_MUTE_AUDIO. If set, then audio will be muted when playing back at a non-standard speed. For a device implementing the *Memory-to-Memory Stateful Video Decoder Interface*, once the drain sequence is initiated with the V4L2_DEC_CMD_STOP command, it must be driven to completion before this command can be invoked. Any attempt to invoke the command while the drain sequence is in progress will trigger an EBUSY error code. The command may be also used to restart the decoder in case of an implicit stop initiated by the decoder itself, without the V4L2_DEC_CMD_STOP being called explicitly. See *Memory-to-Memory Stateful Video Decoder Interface* for more details.
V4L2_DEC_CMD_STOP	1	Stop the decoder. When the decoder is already stopped, this command does nothing. This command has two flags: if V4L2_DEC_CMD_STOP_TO_BLACK is set, then the decoder will set the picture to black after it stopped decoding. Otherwise the last image will repeat. If V4L2_DEC_CMD_STOP_IMMEDIATELY is set, then the decoder stops immediately (ignoring the pts value), otherwise it will keep decoding until timestamp >= pts or until the last of the pending data from its internal buffers was decoded. For a device implementing the *Memory-to-Memory Stateful Video Decoder Interface*, the command will initiate the drain sequence as documented in *Memory-to-Memory Stateful Video Decoder Interface*. No flags or other arguments are accepted in this case. Any attempt to invoke the command again before the sequence completes will trigger an EBUSY error code.
V4L2_DEC_CMD_PAUSE	2	Pause the decoder. When the decoder has not been started yet, the driver will return an EPERM error code. When the decoder is already paused, this command does nothing. This command has one flag: if V4L2_DEC_CMD_PAUSE_TO_BLACK is set, then set the decoder output to black when paused.
V4L2_DEC_CMD_RESUME	3	Resume decoding after a PAUSE command. When the decoder has not been started yet, the driver will return an EPERM error code. When the decoder is already running, this command does nothing. No flags are defined for this command.
V4L2_DEC_CMD_FLUSH	4	Flush any held capture buffers. Only valid for stateless decoders. This command is typically used when the application reached the end of the stream and the last output buffer had the V4L2_BUF_FLAG_M2M_HOLD_CAPTURE_BUF flag set. This would prevent dequeueing the capture buffer containing the last decoded frame. So this command can be used to explicitly flush that final decoded frame. This command does nothing if there are no held capture buffers.

Return Value

On success 0 is returned, on error -1 and the `errno` variable is set appropriately. The generic error codes are described at the *Generic Error Codes* chapter.

EBUSY
A drain sequence of a device implementing the *Memory-to-Memory Stateful Video Decoder Interface* is still in progress. It is not allowed to issue another decoder command until it completes.

EINVAL
The `cmd` field is invalid.

EPERM
The application sent a PAUSE or RESUME command when the decoder was not running.

ioctl VIDIOC_DQEVENT

Name

VIDIOC_DQEVENT - Dequeue event

Synopsis

`VIDIOC_DQEVENT`

```
int ioctl(int fd, VIDIOC_DQEVENT, struct v4l2_event *argp)
```

Arguments

`fd`
File descriptor returned by *`open()`*.

`argp`
Pointer to struct *`v4l2_event`*.

Description

Dequeue an event from a video device. No input is required for this ioctl. All the fields of the struct *`v4l2_event`* structure are filled by the driver. The file handle will also receive exceptions which the application may get by e.g. using the select system call.

type **`v4l2_event`**

Table 117: struct v4l2_event

__u32	type	Type of the event, see *Event Types*.
union {	u	
struct *v4l2_event_vs*	vsync	Event data for event V4L2_EVENT_VSYNC.
struct *v4l2_event_ct*	ctrl	Event data for event V4L2_EVENT_CTRL.
struct *v4l2_event_fr*	frame_sync	Event data for event V4L2_EVENT_FRAME_SYNC.
struct *v4l2_event_mo*	motion_det	Event data for event V4L2_EVENT_MOTION_DET.
struct *v4l2_event_sr*	src_change	Event data for event V4L2_EVENT_SOURCE_CHANGE.
__u8	data[64]	Event data. Defined by the event type. The union should be used to define easily accessible type for events.
}		
__u32	pending	Number of pending events excluding this one.
__u32	sequence	Event sequence number. The sequence number is incremented for every subscribed event that takes place. If sequence numbers are not contiguous it means that events have

Table 118: Event Types

V4L2_EVENT_ALL	0	All events. V4L2_EVENT_ALL is valid only for VIDIOC_UNSUBSCRIBE_EVENT for unsubscribing all events at once.
V4L2_EVENT_VSYNC	1	This event is triggered on the vertical sync. This event has a struct *v4l2_event_vsync* associated with it.
V4L2_EVENT_EOS	2	This event is triggered when the end of a stream is reached. This is typically used with MPEG decoders to report to the application when the last of the MPEG stream has been decoded.
V4L2_EVENT_CTRL	3	This event requires that the id matches the control ID from which you want to receive events. This event is triggered if the control's value changes, if a button control is pressed or if the control's flags change. This event has a struct *v4l2_event_ctrl* associated with it. This struct contains much of the same information as struct *v4l2_queryctrl* and struct *v4l2_control*. If the event is generated due to a call to *VIDIOC_S_CTRL* or *VIDIOC_S_EXT_CTRLS*, then the event will *not* be sent to the file handle that called the ioctl function. This prevents nasty feedback loops. If you *do* want to get the event, then set the V4L2_EVENT_SUB_FL_ALLOW_FEEDBACK flag. This event type will ensure that no information is lost when more events are raised than there is room internally. In that case the struct *v4l2_event_ctrl* of the second-oldest event is kept, but the changes field of the second-oldest event is ORed with the changes field of the oldest event.
V4L2_EVENT_FRAME_SYNC	4	Triggered immediately when the reception of a frame has begun. This event has a struct *v4l2_event_frame_sync* associated with it. If the hardware needs to be stopped in the case of a buffer underrun it might not be able to generate this event. In such cases the frame_sequence field in struct *v4l2_event_frame_sync* will not be incremented. This causes two consecutive frame sequence numbers to have n times frame interval in between them.

continues on next page

Table 118 – continued from previous page

V4L2_EVENT_SOURCE_CHANGE	5	This event is triggered when a source parameter change is detected during runtime by the video device. It can be a runtime resolution change triggered by a video decoder or the format change happening on an input connector. This event requires that the id matches the input index (when used with a video device node) or the pad index (when used with a subdevice node) from which you want to receive events. This event has a struct *v4l2_event_src_change* associated with it. The changes bitfield denotes what has changed for the subscribed pad. If multiple events occurred before application could dequeue them, then the changes will have the ORed value of all the events generated.
V4L2_EVENT_MOTION_DET	6	Triggered whenever the motion detection state for one or more of the regions changes. This event has a struct *v4l2_event_motion_det* associated with it.
V4L2_EVENT_PRIVATE_START	0x0800000(	Base event number for driver-private events.

type **v4l2_event_vsync**

Table 119: struct v4l2_event_vsync

__u8	field	The upcoming field. See enum *v4l2_field*.

type **v4l2_event_ctrl**

Table 120: struct v4l2_event_ctrl

__u32	changes	A bit-mask that tells what has changed. See *Control Changes*.
__u32	type	The type of the control. See enum *v4l2_ctrl_type*.
union {	(anonymous)	
__s32	value	The 32-bit value of the control for 32-bit control types. This is 0 for string controls since the value of a string cannot be passed using *ioctl VIDIOC_DQEVENT*.
__s64	value64	The 64-bit value of the control for 64-bit control types.
}		
__u32	flags	The control flags. See *Control Flags*.
__s32	minimum	The minimum value of the control. See

type **v4l2_event_frame_sync**

Table 121: struct v4l2_event_frame_sync

__u32	frame_sequence	The sequence number of the frame being received.

type **v4l2_event_src_change**

Table 122: struct v4l2_event_src_change

__u32	changes	A bitmask that tells what has changed. See *Source Changes*.

type **v4l2_event_motion_det**

Table 123: struct v4l2_event_motion_det

__u32	flags	Currently only one flag is available: if V4L2_EVENT_MD_FL_HAVE_FRAME_SEQ is set, then the frame_sequence field is valid, otherwise that field should be ignored.
__u32	frame_sequence	The sequence number of the frame being received. Only valid if the V4L2_EVENT_MD_FL_HAVE_FRAME_SEQ flag was set.
__u32	region_mask	The bitmask of the regions that reported motion. There is at least one region. If this field is 0, then no motion was detected at all. If there is no V4L2_CID_DETECT_MD_REGION_GRID control (see *Detect Control Reference*) to assign a different region to each cell in the motion detection grid, then that all cells are automatically assigned to the default region 0.

Table 124: Control Changes

V4L2_EVENT_CTRL_CH_VALUE	0x0001	This control event was triggered because the value of the control changed. Special cases: Volatile controls do no generate this event; If a control has the V4L2_CTRL_FLAG_EXECUTE_ON_WRITE flag set, then this event is sent as well, regardless its value.
V4L2_EVENT_CTRL_CH_FLAGS	0x0002	This control event was triggered because the control flags changed.
V4L2_EVENT_CTRL_CH_RANGE	0x0004	This control event was triggered because the minimum, maximum, step or the default value of the control changed.

Table 125: Source Changes

V4L2_EVENT_SRC_CH_RESOLUTION	0x0001	This event gets triggered when a resolution change is detected at an input. This can come from an input connector or from a video decoder. Applications will have to query the new resolution (if any, the signal may also have been lost). For stateful decoders follow the guidelines in *Memory-to-Memory Stateful Video Decoder Interface*. Video Capture devices have to query the new timings using *ioctl VIDIOC_QUERY_DV_TIMINGS* or *VIDIOC_QUERYSTD*. *Important*: even if the new video timings appear identical to the old ones, receiving this event indicates that there was an issue with the video signal and you must stop and restart streaming (*VIDIOC_STREAMOFF* followed by *VIDIOC_STREAMON*). The reason is that many Video Capture devices are not able to recover from a temporary loss of signal and so restarting streaming I/O is required in order for the hardware to synchronize to the video signal.

Return Value

On success 0 is returned, on error -1 and the `errno` variable is set appropriately. The generic error codes are described at the *Generic Error Codes* chapter.

ioctl VIDIOC_DV_TIMINGS_CAP, VIDIOC_SUBDEV_DV_TIMINGS_CAP

Name

VIDIOC_DV_TIMINGS_CAP - VIDIOC_SUBDEV_DV_TIMINGS_CAP - The capabilities of the Digital Video receiver/transmitter

Synopsis

VIDIOC_DV_TIMINGS_CAP

`int ioctl(int fd, VIDIOC_DV_TIMINGS_CAP, struct v4l2_dv_timings_cap *argp)`

VIDIOC_SUBDEV_DV_TIMINGS_CAP

`int ioctl(int fd, VIDIOC_SUBDEV_DV_TIMINGS_CAP, struct v4l2_dv_timings_cap *argp)`

Arguments

fd
File descriptor returned by *open()*.

argp
Pointer to struct *v4l2_dv_timings_cap*.

Description

To query the capabilities of the DV receiver/transmitter applications initialize the pad field to 0, zero the reserved array of struct *v4l2_dv_timings_cap* and call the VIDIOC_DV_TIMINGS_CAP ioctl on a video node and the driver will fill in the structure.

Note: Drivers may return different values after switching the video input or output.

When implemented by the driver DV capabilities of subdevices can be queried by calling the VIDIOC_SUBDEV_DV_TIMINGS_CAP ioctl directly on a subdevice node. The capabilities are specific to inputs (for DV receivers) or outputs (for DV transmitters), applications must specify the desired pad number in the struct *v4l2_dv_timings_cap* pad field and zero the reserved array. Attempts to query capabilities on a pad that doesn't support them will return an EINVAL error code.

type **v4l2_bt_timings_cap**

Table 126: struct v4l2_bt_timings_cap

__u32	min_width	Minimum width of the active video in pixels.
__u32	max_width	Maximum width of the active video in pixels.
__u32	min_height	Minimum height of the active video in lines.
__u32	max_height	Maximum height of the active video in lines.
__u64	min_pixelcloc	Minimum pixelclock frequency in Hz.
__u64	max_pixelcloc	Maximum pixelclock frequency in Hz.
__u32	standards	The video standard(s) supported by the hardware. See *DV BT Timing standards* for a list of standards.
__u32	capabilities	Several flags giving more information about the capabilities. See *DV BT Timing capabilities* for a description of the flags.
__u32	reserved[16]	Reserved for future extensions. Drivers must set the array to zero.

type **v4l2_dv_timings_cap**

Table 127: struct v4l2_dv_timings_cap

__u32	type	Type of DV timings as listed in *DV Timing types*.
__u32	pad	Pad number as reported by the media controller API. This field is only used when operating on a subdevice node. When operating on a video node applications must set this field to zero.
__u32	reserved[2]	Reserved for future extensions. Drivers and applications must set the array to zero.
union {	(anonymous)	
struct *v4l2_*	bt	BT.656/1120 timings capabilities of the hardware.
__u32	raw_data[32]	
}		

Table 128: DV BT Timing capabilities

Flag	Description
V4L2_DV_BT_CAP_INTERLACED	Interlaced formats are supported.
V4L2_DV_BT_CAP_PROGRESSIVE	Progressive formats are supported.
V4L2_DV_BT_CAP_REDUCED_BLANKIN(	CVT/GTF specific: the timings can make use of reduced blanking (CVT) or the 'Secondary GTF' curve (GTF).
V4L2_DV_BT_CAP_CUSTOM	Can support non-standard timings, i.e. timings not belonging to the standards set in the standards field.

Return Value

On success 0 is returned, on error -1 and the `errno` variable is set appropriately. The generic error codes are described at the *Generic Error Codes* chapter.

ioctl VIDIOC_ENCODER_CMD, VIDIOC_TRY_ENCODER_CMD

Name

VIDIOC_ENCODER_CMD - VIDIOC_TRY_ENCODER_CMD - Execute an encoder command

Synopsis

VIDIOC_ENCODER_CMD

`int ioctl(int fd, VIDIOC_ENCODER_CMD, struct v4l2_encoder_cmd *argp)`

VIDIOC_TRY_ENCODER_CMD

`int ioctl(int fd, VIDIOC_TRY_ENCODER_CMD, struct v4l2_encoder_cmd *argp)`

Arguments

fd
File descriptor returned by *open()*.

argp
Pointer to struct *v4l2_encoder_cmd*.

Description

These ioctls control an audio/video (usually MPEG-) encoder. `VIDIOC_ENCODER_CMD` sends a command to the encoder, `VIDIOC_TRY_ENCODER_CMD` can be used to try a command without actually executing it.

To send a command applications must initialize all fields of a struct *v4l2_encoder_cmd* and call `VIDIOC_ENCODER_CMD` or `VIDIOC_TRY_ENCODER_CMD` with a pointer to this structure.

The `cmd` field must contain the command code. Some commands use the `flags` field for additional information.

After a STOP command, *read()* calls will read the remaining data buffered by the driver. When the buffer is empty, *read()* will return zero and the next *read()* call will restart the encoder.

A *read()* or *VIDIOC_STREAMON* call sends an implicit START command to the encoder if it has not been started yet. Applies to both queues of mem2mem encoders.

A *close()* or *VIDIOC_STREAMOFF* call of a streaming file descriptor sends an implicit immediate STOP to the encoder, and all buffered data is discarded. Applies to both queues of mem2mem encoders.

These ioctls are optional, not all drivers may support them. They were introduced in Linux 2.6.21. They are, however, mandatory for stateful mem2mem encoders (as further documented in *Memory-to-Memory Stateful Video Encoder Interface*).

type **v4l2_encoder_cmd**

Table 129: struct v4l2_encoder_cmd

__u32	cmd	The encoder command, see *Encoder Commands*.
__u32	flags	Flags to go with the command, see *Encoder Command Flags*. If no flags are defined for this command, drivers and applications must set this field to zero.
__u32	data[8]	Reserved for future extensions. Drivers and applications must set the array to zero.

Table 130: Encoder Commands

V4L2_ENC_CMD_START	0	Start the encoder. When the encoder is already running or paused, this command does nothing. No flags are defined for this command. For a device implementing the *Memory-to-Memory Stateful Video Encoder Interface*, once the drain sequence is initiated with the V4L2_ENC_CMD_STOP command, it must be driven to completion before this command can be invoked. Any attempt to invoke the command while the drain sequence is in progress will trigger an EBUSY error code. See *Memory-to-Memory Stateful Video Encoder Interface* for more details.
V4L2_ENC_CMD_STOP	1	Stop the encoder. When the V4L2_ENC_CMD_STOP_AT_GOP_END flag is set, encoding will continue until the end of the current *Group Of Pictures*, otherwise encoding will stop immediately. When the encoder is already stopped, this command does nothing. For a device implementing the *Memory-to-Memory Stateful Video Encoder Interface*, the command will initiate the drain sequence as documented in *Memory-to-Memory Stateful Video Encoder Interface*. No flags or other arguments are accepted in this case. Any attempt to invoke the command again before the sequence completes will trigger an EBUSY error code.
V4L2_ENC_CMD_PAUSE	2	Pause the encoder. When the encoder has not been started yet, the driver will return an EPERM error code. When the encoder is already paused, this command does nothing. No flags are defined for this command.
V4L2_ENC_CMD_RESUME	3	Resume encoding after a PAUSE command. When the encoder has not been started yet, the driver will return an EPERM error code. When the encoder is already running, this command does nothing. No flags are defined for this command.

Table 131: Encoder Command Flags

V4L2_ENC_CMD_STOP_AT_GOP_END	0x0001	Stop encoding at the end of the current *Group Of Pictures*, rather than immediately. Does not apply to *Memory-to-Memory Stateful Video Encoder Interface*.

Return Value

On success 0 is returned, on error -1 and the `errno` variable is set appropriately. The generic error codes are described at the *Generic Error Codes* chapter.

EBUSY
A drain sequence of a device implementing the *Memory-to-Memory Stateful Video Encoder Interface* is still in progress. It is not allowed to issue another encoder command until it completes.

EINVAL
The `cmd` field is invalid.

EPERM
The application sent a PAUSE or RESUME command when the encoder was not running.

ioctl VIDIOC_ENUMAUDIO

Name

VIDIOC_ENUMAUDIO - Enumerate audio inputs

Synopsis

`VIDIOC_ENUMAUDIO`

`int ioctl(int fd, VIDIOC_ENUMAUDIO, struct v4l2_audio *argp)`

Arguments

`fd`
File descriptor returned by *`open()`*.

`argp`
Pointer to struct *`v4l2_audio`*.

Description

To query the attributes of an audio input applications initialize the `index` field and zero out the `reserved` array of a struct *`v4l2_audio`* and call the *ioctl VIDIOC_ENUMAUDIO* ioctl with a pointer to this structure. Drivers fill the rest of the structure or return an `EINVAL` error code when the index is out of bounds. To enumerate all audio inputs applications shall begin at index zero, incrementing by one until the driver returns `EINVAL`.

See *VIDIOC_G_AUDIO* for a description of struct *`v4l2_audio`*.

Return Value

On success 0 is returned, on error -1 and the `errno` variable is set appropriately. The generic error codes are described at the *Generic Error Codes* chapter.

EINVAL
The number of the audio input is out of bounds.

ioctl VIDIOC_ENUMAUDOUT

Name

VIDIOC_ENUMAUDOUT - Enumerate audio outputs

Synopsis

VIDIOC_ENUMAUDOUT

```
int ioctl(int fd, VIDIOC_ENUMAUDOUT, struct v4l2_audioout *argp)
```

Arguments

fd
File descriptor returned by *open()*.

argp
Pointer to struct *v4l2_audioout*.

Description

To query the attributes of an audio output applications initialize the `index` field and zero out the `reserved` array of a struct *v4l2_audioout* and call the `VIDIOC_G_AUDOUT` ioctl with a pointer to this structure. Drivers fill the rest of the structure or return an `EINVAL` error code when the index is out of bounds. To enumerate all audio outputs applications shall begin at index zero, incrementing by one until the driver returns `EINVAL`.

Note: Connectors on a TV card to loop back the received audio signal to a sound card are not audio outputs in this sense.

See *VIDIOC_G_AUDIOout* for a description of struct *v4l2_audioout*.

Return Value

On success 0 is returned, on error -1 and the `errno` variable is set appropriately. The generic error codes are described at the *Generic Error Codes* chapter.

EINVAL
The number of the audio output is out of bounds.

ioctl VIDIOC_ENUM_DV_TIMINGS, VIDIOC_SUBDEV_ENUM_DV_TIMINGS

Name

VIDIOC_ENUM_DV_TIMINGS - VIDIOC_SUBDEV_ENUM_DV_TIMINGS - Enumerate supported Digital Video timings

Synopsis

`VIDIOC_ENUM_DV_TIMINGS`

`int ioctl(int fd, VIDIOC_ENUM_DV_TIMINGS, struct v4l2_enum_dv_timings *argp)`

`VIDIOC_SUBDEV_ENUM_DV_TIMINGS`

`int ioctl(int fd, VIDIOC_SUBDEV_ENUM_DV_TIMINGS, struct v4l2_enum_dv_timings *argp)`

Arguments

`fd`
File descriptor returned by *`open()`*.

`argp`
Pointer to struct *`v4l2_enum_dv_timings`*.

Description

While some DV receivers or transmitters support a wide range of timings, others support only a limited number of timings. With this ioctl applications can enumerate a list of known supported timings. Call *ioctl VIDIOC_DV_TIMINGS_CAP, VIDIOC_SUBDEV_DV_TIMINGS_CAP* to check if it also supports other standards or even custom timings that are not in this list.

To query the available timings, applications initialize the `index` field, set the `pad` field to 0, zero the reserved array of struct *`v4l2_enum_dv_timings`* and call the `VIDIOC_ENUM_DV_TIMINGS` ioctl on a video node with a pointer to this structure. Drivers fill the rest of the structure or return an `EINVAL` error code when the index is out of bounds. To enumerate all supported DV timings, applications shall begin at index zero, incrementing by one until the driver returns `EINVAL`.

Note: Drivers may enumerate a different set of DV timings after switching the video input or output.

When implemented by the driver DV timings of subdevices can be queried by calling the `VIDIOC_SUBDEV_ENUM_DV_TIMINGS` ioctl directly on a subdevice node. The DV timings are specific to inputs (for DV receivers) or outputs (for DV transmitters), applications must specify the desired pad number in the struct *v4l2_enum_dv_timings* pad field. Attempts to enumerate timings on a pad that doesn't support them will return an `EINVAL` error code.

type **v4l2_enum_dv_timings**

Table 132: struct v4l2_enum_dv_timings

__u32	index	Number of the DV timings, set by the application.
__u32	pad	Pad number as reported by the media controller API. This field is only used when operating on a subdevice node. When operating on a video node applications must set this field to zero.
__u32	reserved[2]	Reserved for future extensions. Drivers and applications must set the array to zero.
struct *v4l2_dv_timings*	timings	The timings.

Return Value

On success 0 is returned, on error -1 and the `errno` variable is set appropriately. The generic error codes are described at the *Generic Error Codes* chapter.

EINVAL
The struct *v4l2_enum_dv_timings* `index` is out of bounds or the `pad` number is invalid.

ENODATA
Digital video presets are not supported for this input or output.

ioctl VIDIOC_ENUM_FMT

Name

VIDIOC_ENUM_FMT - Enumerate image formats

Synopsis

VIDIOC_ENUM_FMT

`int ioctl(int fd, VIDIOC_ENUM_FMT, struct v4l2_fmtdesc *argp)`

Arguments

fd
File descriptor returned by *open()*.

argp
Pointer to struct *v4l2_fmtdesc*.

Description

To enumerate image formats applications initialize the `type`, `mbus_code` and `index` fields of struct *v4l2_fmtdesc* and call the *ioctl VIDIOC_ENUM_FMT* ioctl with a pointer to this structure. Drivers fill the rest of the structure or return an `EINVAL` error code. All formats are enumerable by beginning at index zero and incrementing by one until `EINVAL` is returned. If applicable, drivers shall return formats in preference order, where preferred formats are returned before (that is, with lower `index` value) less-preferred formats.

Depending on the `V4L2_CAP_IO_MC` *capability*, the `mbus_code` field is handled differently:

1) `V4L2_CAP_IO_MC` is not set (also known as a 'video-node-centric' driver)

   Applications shall initialize the `mbus_code` field to zero and drivers shall ignore the value of the field.

   Drivers shall enumerate all image formats.

   Note: After switching the input or output the list of enumerated image formats may be different.

2) `V4L2_CAP_IO_MC` is set (also known as an 'MC-centric' driver)

   If the `mbus_code` field is zero, then all image formats shall be enumerated.

   If the `mbus_code` field is initialized to a valid (non-zero) *media bus format code*, then drivers shall restrict enumeration to only the image formats that can produce (for video output devices) or be produced from (for video capture devices) that media bus code. If the `mbus_code` is unsupported by the driver, then `EINVAL` shall be returned.

   Regardless of the value of the `mbus_code` field, the enumerated image formats shall not depend on the active configuration of the video device or device pipeline.

type **v4l2_fmtdesc**

Table 133: struct v4l2_fmtdesc

__u32	index	Number of the format in the enumeration, set by the application. This is in no way related to the pixelformat field.
__u32	type	Type of the data stream, set by the application. Only these types are valid here: V4L2_BUF_TYPE_VIDEO_CAPTURE, V4L2_BUF_TYPE_VIDEO_CAPTURE_MPLANE, V4L2_BUF_TYPE_VIDEO_OUTPUT, V4L2_BUF_TYPE_VIDEO_OUTPUT_MPLANE, V4L2_BUF_TYPE_VIDEO_OVERLAY, V4L2_BUF_TYPE_SDR_CAPTURE, V4L2_BUF_TYPE_SDR_OUTPUT, V4L2_BUF_TYPE_META_CAPTURE and V4L2_BUF_TYPE_META_OUTPUT. See *v4l2_buf_type*.
__u32	flags	See *Image Format Description Flags*
__u8	description[32]	Description of the format, a NUL-terminated ASCII string. This information is intended for the user, for example: "YUV 4:2:2".
__u32	pixelformat	The image format identifier. This is a four character code as computed by the v4l2_fourcc() macro:

```
#define v4l2_fourcc(a,b,c,d)
(((__u32)(a)<<0)|((__u32)(b)<<8)|((__u32)(c)<<16)|((__u32)(d)<<24))
```

Several image formats are already defined by this specification in *Image Formats*.

> **Attention:** These codes are not the same as those used in the Windows world.

__u32	mbus_code	Media bus code restricting the enumerated formats, set by the application. Only applicable to drivers that advertise the V4L2_CAP_IO_MC *capability*, shall be 0 otherwise.
__u32	reserved[3]	Reserved for future extensions. Drivers must set the array to zero.

Table 134: Image Format Description Flags

V4L2_FMT_FLAG_COMPRESSED	0x0001	This is a compressed format.
V4L2_FMT_FLAG_EMULATED	0x0002	This format is not native to the device but emulated through software (usually libv4l2), where possible try to use a native format instead for better performance.
V4L2_FMT_FLAG_CONTINUOUS_BYTE	0x0004	The hardware decoder for this compressed bytestream format (aka coded format) is capable of parsing a continuous bytestream. Applications do not need to parse the bytestream themselves to find the boundaries between frames/fields. This flag can only be used in combination with the V4L2_FMT_FLAG_COMPRESSED flag, since this applies to compressed formats only. This flag is valid for stateful decoders only.
V4L2_FMT_FLAG_DYN_RESOLUTION	0x0008	Dynamic resolution switching is supported by the device for this compressed bytestream format (aka coded format). It will notify the user via the event V4L2_EVENT_SOURCE_CHANGE when changes in the video parameters are detected. This flag can only be used in combination with the V4L2_FMT_FLAG_COMPRESSED flag, since this applies to compressed formats only. This flag is valid for stateful codecs only.
V4L2_FMT_FLAG_ENC_CAP_FRAME_I	0x0010	The hardware encoder supports setting the CAPTURE coded frame interval separately from the OUTPUT raw frame interval. Setting the OUTPUT raw frame interval with *VIDIOC_S_PARM* also sets the CAPTURE coded frame interval to the same value. If this flag is set, then the CAPTURE coded frame interval can be set to a different value afterwards. This is typically used for offline encoding where the OUTPUT raw frame interval is used as a hint for reserving hardware encoder resources and the CAPTURE coded frame interval is the actual frame rate embedded in the encoded video stream. This flag can only be used in combination with the V4L2_FMT_FLAG_COMPRESSED flag, since this applies to compressed formats only. This flag is valid for stateful encoders only.
V4L2_FMT_FLAG_CSC_COLORSPACE	0x0020	The driver allows the application to try to change the default colorspace. This flag is relevant only for capture devices. The application can ask to configure the colorspace of the capture device when calling the *VIDIOC_S_FMT* ioctl with *V4L2_PIX_FMT_FLAG_SET_CSC* set.
V4L2_FMT_FLAG_CSC_XFER_FUNC	0x0040	The driver allows the application to try to change the default transfer function.

Return Value

On success 0 is returned, on error -1 and the `errno` variable is set appropriately. The generic error codes are described at the *Generic Error Codes* chapter.

EINVAL
: The struct *v4l2_fmtdesc* `type` is not supported or the `index` is out of bounds.

  If `V4L2_CAP_IO_MC` is set and the specified `mbus_code` is unsupported, then also return this error code.

ioctl VIDIOC_ENUM_FRAMESIZES

Name

VIDIOC_ENUM_FRAMESIZES - Enumerate frame sizes

Synopsis

VIDIOC_ENUM_FRAMESIZES

`int ioctl(int fd, VIDIOC_ENUM_FRAMESIZES, struct v4l2_frmsizeenum *argp)`

Arguments

fd
: File descriptor returned by *open()*.

argp
: Pointer to struct *v4l2_frmsizeenum* that contains an index and pixel format and receives a frame width and height.

Description

This ioctl allows applications to enumerate all frame sizes (i. e. width and height in pixels) that the device supports for the given pixel format.

The supported pixel formats can be obtained by using the *ioctl VIDIOC_ENUM_FMT* function.

The return value and the content of the `v4l2_frmsizeenum.type` field depend on the type of frame sizes the device supports. Here are the semantics of the function for the different cases:

- **Discrete:** The function returns success if the given index value (zero-based) is valid. The application should increase the index by one for each call until `EINVAL` is returned. The `v4l2_frmsizeenum.type` field is set to `V4L2_FRMSIZE_TYPE_DISCRETE` by the driver. Of the union only the `discrete` member is valid.

- **Step-wise:** The function returns success if the given index value is zero and EINVAL for any other index value. The v4l2_frmsizeenum.type field is set to V4L2_FRMSIZE_TYPE_STEPWISE by the driver. Of the union only the stepwise member is valid.
- **Continuous:** This is a special case of the step-wise type above. The function returns success if the given index value is zero and EINVAL for any other index value. The v4l2_frmsizeenum.type field is set to V4L2_FRMSIZE_TYPE_CONTINUOUS by the driver. Of the union only the stepwise member is valid and the step_width and step_height values are set to 1.

When the application calls the function with index zero, it must check the type field to determine the type of frame size enumeration the device supports. Only for the V4L2_FRMSIZE_TYPE_DISCRETE type does it make sense to increase the index value to receive more frame sizes.

Note: The order in which the frame sizes are returned has no special meaning. In particular does it not say anything about potential default format sizes.

Applications can assume that the enumeration data does not change without any interaction from the application itself. This means that the enumeration data is consistent if the application does not perform any other ioctl calls while it runs the frame size enumeration.

Structs

In the structs below, *IN* denotes a value that has to be filled in by the application, *OUT* denotes values that the driver fills in. The application should zero out all members except for the *IN* fields.

type **v4l2_frmsize_discrete**

Table 135: struct v4l2_frmsize_discrete

__u32	width	Width of the frame [pixel].
__u32	height	Height of the frame [pixel].

type **v4l2_frmsize_stepwise**

Table 136: struct v4l2_frmsize_stepwise

__u32	min_width	Minimum frame width [pixel].
__u32	max_width	Maximum frame width [pixel].
__u32	step_width	Frame width step size [pixel].
__u32	min_height	Minimum frame height [pixel].
__u32	max_height	Maximum frame height [pixel].
__u32	step_height	Frame height step size [pixel].

type **v4l2_frmsizeenum**

Table 137: struct v4l2_frmsizeenum

__u32	index	IN: Index of the given frame size in the enumeration.
__u32	pixel_format	IN: Pixel format for which the frame sizes are enumerated.
__u32	type	OUT: Frame size type the device supports.
union {	(anonymous)	OUT: Frame size with the given index.
struct *v4l2_fı*	discrete	
struct *v4l2_fı*	stepwise	
}		
__u32	reserved[2]	Reserved space for future use. Must be zeroed by drivers and applications.

Enums

type **v4l2_frmsizetypes**

Table 138: enum v4l2_frmsizetypes

V4L2_FRMSIZE_TYPE_DISCRETE	1	Discrete frame size.
V4L2_FRMSIZE_TYPE_CONTINUOUS	2	Continuous frame size.
V4L2_FRMSIZE_TYPE_STEPWISE	3	Step-wise defined frame size.

Return Value

On success 0 is returned, on error -1 and the `errno` variable is set appropriately. The generic error codes are described at the *Generic Error Codes* chapter.

ioctl VIDIOC_ENUM_FRAMEINTERVALS

Name

VIDIOC_ENUM_FRAMEINTERVALS - Enumerate frame intervals

Synopsis

VIDIOC_ENUM_FRAMEINTERVALS

`int ioctl(int fd, VIDIOC_ENUM_FRAMEINTERVALS, struct v4l2_frmivalenum *argp)`

Arguments

fd
: File descriptor returned by *open()*.

argp
: Pointer to struct *v4l2_frmivalenum* that contains a pixel format and size and receives a frame interval.

Description

This ioctl allows applications to enumerate all frame intervals that the device supports for the given pixel format and frame size.

The supported pixel formats and frame sizes can be obtained by using the *ioctl VIDIOC_ENUM_FMT* and *ioctl VIDIOC_ENUM_FRAMESIZES* functions.

The return value and the content of the `v4l2_frmivalenum.type` field depend on the type of frame intervals the device supports. Here are the semantics of the function for the different cases:

- **Discrete:** The function returns success if the given index value (zero-based) is valid. The application should increase the index by one for each call until `EINVAL` is returned. The *v4l2_frmivalenum.type* field is set to *V4L2_FRMIVAL_TYPE_DISCRETE* by the driver. Of the union only the *discrete* member is valid.
- **Step-wise:** The function returns success if the given index value is zero and `EINVAL` for any other index value. The `v4l2_frmivalenum.type` field is set to `V4L2_FRMIVAL_TYPE_STEPWISE` by the driver. Of the union only the `stepwise` member is valid.

- **Continuous:** This is a special case of the step-wise type above. The function returns success if the given index value is zero and EINVAL for any other index value. The v4l2_frmivalenum.type field is set to V4L2_FRMIVAL_TYPE_CONTINUOUS by the driver. Of the union only the stepwise member is valid and the step value is set to 1.

When the application calls the function with index zero, it must check the type field to determine the type of frame interval enumeration the device supports. Only for the V4L2_FRMIVAL_TYPE_DISCRETE type does it make sense to increase the index value to receive more frame intervals.

Note: The order in which the frame intervals are returned has no special meaning. In particular does it not say anything about potential default frame intervals.

Applications can assume that the enumeration data does not change without any interaction from the application itself. This means that the enumeration data is consistent if the application does not perform any other ioctl calls while it runs the frame interval enumeration.

Note: **Frame intervals and frame rates:** The V4L2 API uses frame intervals instead of frame rates. Given the frame interval the frame rate can be computed as follows:

```
frame_rate = 1 / frame_interval
```

Structs

In the structs below, *IN* denotes a value that has to be filled in by the application, *OUT* denotes values that the driver fills in. The application should zero out all members except for the *IN* fields.

type **v4l2_frmival_stepwise**

Table 139: struct v4l2_frmival_stepwise

struct *v4l2_fract*	min	Minimum frame interval [s].
struct *v4l2_fract*	max	Maximum frame interval [s].
struct *v4l2_fract*	step	Frame interval step size [s].

type **v4l2_frmivalenum**

Table 140: struct v4l2_frmivalenum

__u32	index	IN: Index of the given frame interval in the enumeration.
__u32	pixel_format	IN: Pixel format for which the frame intervals are enumerated.
__u32	width	IN: Frame width for which the frame intervals are enumerated.
__u32	height	IN: Frame height for which the frame intervals are enumerated.
__u32	type	OUT: Frame interval type the device supports.
union {	(anonymous)	OUT: Frame interval with the given index.
struct *v4l2_fra*	discrete	Frame interval [s].
struct *v4l2_frm*	stepwise	
}		
__u32	reserved[2]	Reserved space for future use. Must be zeroed by drivers and applications.

Enums

type **v4l2_frmivaltypes**

Table 141: enum v4l2_frmivaltypes

V4L2_FRMIVAL_TYPE_DISCRETE	1	Discrete frame interval.
V4L2_FRMIVAL_TYPE_CONTINUOUS	2	Continuous frame interval.
V4L2_FRMIVAL_TYPE_STEPWISE	3	Step-wise defined frame interval.

Return Value

On success 0 is returned, on error -1 and the errno variable is set appropriately. The generic error codes are described at the *Generic Error Codes* chapter.

ioctl VIDIOC_ENUM_FREQ_BANDS

Name

VIDIOC_ENUM_FREQ_BANDS - Enumerate supported frequency bands

Synopsis

VIDIOC_ENUM_FREQ_BANDS

`int ioctl(int fd, VIDIOC_ENUM_FREQ_BANDS, struct v4l2_frequency_band *argp)`

Arguments

fd
: File descriptor returned by *open()*.

argp
: Pointer to struct *v4l2_frequency_band*.

Description

Enumerates the frequency bands that a tuner or modulator supports. To do this applications initialize the `tuner`, `type` and `index` fields, and zero out the `reserved` array of a struct *v4l2_frequency_band* and call the *ioctl VIDIOC_ENUM_FREQ_BANDS* ioctl with a pointer to this structure.

This ioctl is supported if the `V4L2_TUNER_CAP_FREQ_BANDS` capability of the corresponding tuner/modulator is set.

type **v4l2_frequency_band**

Table 142: struct v4l2_frequency_band

__u32	tuner	The tuner or modulator index number. This is the same value as in the struct *v4l2_input* tuner field and the struct *v4l2_tuner* index field, or the struct *v4l2_output* modulator field and the struct *v4l2_modulator* index field.
__u32	type	The tuner type. This is the same value as in the struct *v4l2_tuner* type field. The type must be set to V4L2_TUNER_RADIO for /dev/radioX device nodes, and to V4L2_TUNER_ANALOG_TV for all others. Set this field to V4L2_TUNER_RADIO for modulators (currently only radio modulators are supported). See *v4l2_tuner_type*
__u32	index	Identifies the frequency band, set by the application.
__u32	capability	The tuner/modulator capability flags for this frequency band, see *Tuner and Modulator Capability Flags*. The V4L2_TUNER_CAP_LOW or V4L2_TUNER_CAP_1HZ capability must be the same for all frequency bands of the selected tuner/modulator. So either all bands have that capability set, or none of them have that capability.
__u32	rangelow	The lowest tunable frequency in units of 62.5 kHz, or if the capability flag V4L2_TUNER_CAP_LOW is set, in units of 62.5 Hz, for this frequency band. A 1 Hz unit is used when the capability flag V4L2_TUNER_CAP_1HZ is set.
__u32	rangehigh	The highest tunable frequency in units of 62.5 kHz, or if the capability flag V4L2_TUNER_CAP_LOW is set, in units of 62.5 Hz, for this frequency band. A 1 Hz unit is used when the capability flag V4L2_TUNER_CAP_1HZ is set.
__u32	modulation	The supported modulation systems of this frequency band. See *Band Modulation Systems*. **Note:** Currently only one modulation system per frequency band is supported. More work will need to be done if multiple modulation systems are possible. Contact the linux-media mailing list (https://linuxtv.org/lists.php) if you need such functionality.
__u32	reserved[9]	Reserved for future extensions. Applications and drivers must set the array to zero.

Table 143: Band Modulation Systems

V4L2_BAND_MODULATION_VSB	0x02	Vestigial Sideband modulation, used for analog TV.
V4L2_BAND_MODULATION_FM	0x04	Frequency Modulation, commonly used for analog radio.
V4L2_BAND_MODULATION_AM	0x08	Amplitude Modulation, commonly used for analog radio.

Return Value

On success 0 is returned, on error -1 and the `errno` variable is set appropriately. The generic error codes are described at the *Generic Error Codes* chapter.

EINVAL
The `tuner` or `index` is out of bounds or the `type` field is wrong.

ioctl VIDIOC_ENUMINPUT

Name

VIDIOC_ENUMINPUT - Enumerate video inputs

Synopsis

`VIDIOC_ENUMINPUT`

`int ioctl(int fd, VIDIOC_ENUMINPUT, struct v4l2_input *argp)`

Arguments

`fd`
File descriptor returned by *`open()`*.

`argp`
Pointer to struct *`v4l2_input`*.

Description

To query the attributes of a video input applications initialize the `index` field of struct *`v4l2_input`* and call the *ioctl VIDIOC_ENUMINPUT* with a pointer to this structure. Drivers fill the rest of the structure or return an `EINVAL` error code when the index is out of bounds. To enumerate all inputs applications shall begin at index zero, incrementing by one until the driver returns `EINVAL`.

type **`v4l2_input`**

Table 144: struct v4l2_input

__u32	index	Identifies the input, set by the application.
__u8	name[32]	Name of the video input, a NUL-terminated ASCII string, for example: "Vin (Composite 2)". This information is intended for the user, preferably the connector label on the device itself.
__u32	type	Type of the input, see *Input Types*.
__u32	audioset	Drivers can enumerate up to 32 video and audio inputs. This field shows which audio inputs were selectable as audio source if this was the currently selected video input. It is a bit mask. The LSB corresponds to audio input 0, the MSB to input 31. Any number of bits can be set, or none. When the driver does not enumerate audio inputs no bits must be set. Applications shall not interpret this as lack of audio support. Some drivers automatically select audio sources and do not enumerate them since there is no choice anyway. For details on audio inputs and how to select the current input see *Audio Inputs and Outputs*.
__u32	tuner	Capture devices can have zero or more tuners (RF demodulators). When the `type` is set to `V4L2_INPUT_TYPE_TUNER` this is an RF connector and this field identifies the tuner. It corresponds to struct *v4l2_tuner* field `index`. For details on tuners see *Tuners and Modulators*.
v4l2_std_id	std	Every video input supports one or more different video standards. This field is a set of all supported standards. For details on video standards and how to switch see *Video Standards*.
__u32	status	This field provides status information about the input. See *Input Status Flags* for flags. With the exception of the sensor orientation bits `status` is only valid when this is the current input.
__u32	capabilities	This field provides capabilities for the input. See *Input capabilities* for flags.
__u32	reserved[3]	Reserved for future extensions. Drivers must set the array to zero.

Table 145: Input Types

V4L2_INPUT_TYPE_TUNER	1	This input uses a tuner (RF demodulator).
V4L2_INPUT_TYPE_CAMERA	2	Any non-tuner video input, for example Composite Video, S-Video, HDMI, camera sensor. The naming as `_TYPE_CAMERA` is historical, today we would have called it `_TYPE_VIDEO`.
V4L2_INPUT_TYPE_TOUCH	3	This input is a touch device for capturing raw touch data.

Table 146: Input Status Flags

General		
V4L2_IN_ST_NO_POWER	0x00000001	Attached device is off.
V4L2_IN_ST_NO_SIGNAL	0x00000002	
V4L2_IN_ST_NO_COLOR	0x00000004	The hardware supports color decoding, but does not detect color modulation in the signal.
Sensor Orientation		
V4L2_IN_ST_HFLIP	0x00000010	The input is connected to a device that produces a signal that is flipped horizontally and does not correct this before passing the signal to userspace.
V4L2_IN_ST_VFLIP	0x00000020	The input is connected to a device that produces a signal that is flipped vertically and does not correct this before passing the signal to userspace. .. note:: A 180 degree rotation is the same as HFLIP | VFLIP
Analog Video		
V4L2_IN_ST_NO_H_LOCK	0x00000100	No horizontal sync lock.
V4L2_IN_ST_COLOR_KILL	0x00000200	A color killer circuit automatically disables color decoding when it detects no color modulation. When this flag is set the color killer is enabled *and* has shut off color decoding.
V4L2_IN_ST_NO_V_LOCK	0x00000400	No vertical sync lock.
V4L2_IN_ST_NO_STD_LOC	0x00000800	No standard format lock in case of auto-detection format by the component.
Digital Video		
V4L2_IN_ST_NO_SYNC	0x00010000	No synchronization lock.
V4L2_IN_ST_NO_EQU	0x00020000	No equalizer lock.
V4L2_IN_ST_NO_CARRIER	0x00040000	Carrier recovery failed.
VCR and Set-Top Box		
V4L2_IN_ST_MACROVISIO	0x01000000	Macrovision is an analog copy prevention system mangling the video signal to confuse video recorders. When this flag is set Macrovision has been detected.
V4L2_IN_ST_NO_ACCESS	0x02000000	Conditional access denied.
V4L2_IN_ST_VTR	0x04000000	VTR time constant. [?]

Table 147: Input capabilities

V4L2_IN_CAP_DV_TIMINGS	0x0000000[illegible]	This input supports setting video timings by using VIDIOC_S_DV_TIMINGS.
V4L2_IN_CAP_STD	0x0000000[illegible]	This input supports setting the TV standard by using VIDIOC_S_STD.
V4L2_IN_CAP_NATIVE_SIZE	0x0000000[illegible]	This input supports setting the native size using the V4L2_SEL_TGT_NATIVE_SIZE selection target, see *Common selection definitions*.

Return Value

On success 0 is returned, on error -1 and the `errno` variable is set appropriately. The generic error codes are described at the *Generic Error Codes* chapter.

EINVAL
The struct *v4l2_input* `index` is out of bounds.

ioctl VIDIOC_ENUMOUTPUT

Name

VIDIOC_ENUMOUTPUT - Enumerate video outputs

Synopsis

VIDIOC_ENUMOUTPUT

`int ioctl(int fd, VIDIOC_ENUMOUTPUT, struct v4l2_output *argp)`

Arguments

fd
File descriptor returned by *open()*.

argp
Pointer to struct *v4l2_output*.

Description

To query the attributes of a video outputs applications initialize the index field of struct *v4l2_output* and call the *ioctl VIDIOC_ENUMOUTPUT* with a pointer to this structure. Drivers fill the rest of the structure or return an EINVAL error code when the index is out of bounds. To enumerate all outputs applications shall begin at index zero, incrementing by one until the driver returns EINVAL.

type **v4l2_output**

Table 148: struct v4l2_output

__u32	index	Identifies the output, set by the application.
__u8	name[32]	Name of the video output, a NUL-terminated ASCII string, for example: "Vout". This information is intended for the user, preferably the connector label on the device itself.
__u32	type	Type of the output, see *Output Type*.
__u32	audioset	Drivers can enumerate up to 32 video and audio outputs. This field shows which audio outputs were selectable as the current output if this was the currently selected video output. It is a bit mask. The LSB corresponds to audio output 0, the MSB to output 31. Any number of bits can be set, or none. When the driver does not enumerate audio outputs no bits must be set. Applications shall not interpret this as lack of audio support. Drivers may automatically select audio outputs without enumerating them. For details on audio outputs and how to select the current output see *Audio Inputs and Outputs*.
__u32	modulator	Output devices can have zero or more RF modulators. When the type is V4L2_OUTPUT_TYPE_MODULATOR this is an RF connector and this field identifies the modulator. It corresponds to struct *v4l2_modulator* field index. For details on modulators see *Tuners and Modulators*.
v4l2_std_id	std	Every video output supports one or more different video standards. This field is a set of all supported standards. For details on video standards and how to switch see *Video Standards*.
__u32	capabilities	This field provides capabilities for the output. See *Output capabilities* for flags.
__u32	reserved[3]	Reserved for future extensions. Drivers must set the array to zero.

Table 149: Output Type

V4L2_OUTPUT_TYPE_MODULATOR	1	This output is an analog TV modulator.
V4L2_OUTPUT_TYPE_ANALOG	2	Any non-modulator video output, for example Composite Video, S-Video, HDMI. The naming as _TYPE_ANALOG is historical, today we would have called it _TYPE_VIDEO.
V4L2_OUTPUT_TYPE_ANALOGVGAOVERL	3	The video output will be copied to a *video overlay*.

Table 150: Output capabilities

V4L2_OUT_CAP_DV_TIMINGS	0x0000000	This output supports setting video timings by using VIDIOC_S_DV_TIMINGS.
V4L2_OUT_CAP_STD	0x0000000	This output supports setting the TV standard by using VIDIOC_S_STD.
V4L2_OUT_CAP_NATIVE_SIZE	0x0000000	This output supports setting the native size using the V4L2_SEL_TGT_NATIVE_SIZE selection target, see *Common selection definitions*.

Return Value

On success 0 is returned, on error -1 and the errno variable is set appropriately. The generic error codes are described at the *Generic Error Codes* chapter.

EINVAL
The struct *v4l2_output* index is out of bounds.

ioctl VIDIOC_ENUMSTD, VIDIOC_SUBDEV_ENUMSTD

Name

VIDIOC_ENUMSTD - VIDIOC_SUBDEV_ENUMSTD - Enumerate supported video standards

Synopsis

VIDIOC_ENUMSTD

```
int ioctl(int fd, VIDIOC_ENUMSTD, struct v4l2_standard *argp)
```

VIDIOC_SUBDEV_ENUMSTD

```
int ioctl(int fd, VIDIOC_SUBDEV_ENUMSTD, struct v4l2_standard *argp)
```

Arguments

fd
  File descriptor returned by *open()*.

argp
  Pointer to struct *v4l2_standard*.

Description

To query the attributes of a video standard, especially a custom (driver defined) one, applications initialize the index field of struct *v4l2_standard* and call the *ioctl VIDIOC_ENUMSTD, VIDIOC_SUBDEV_ENUMSTD* ioctl with a pointer to this structure. Drivers fill the rest of the structure or return an EINVAL error code when the index is out of bounds. To enumerate all standards applications shall begin at index zero, incrementing by one until the driver returns EINVAL. Drivers may enumerate a different set of standards after switching the video input or output.[1]

type **v4l2_standard**

Table 151: struct v4l2_standard

__u32	index	Number of the video standard, set by the application.
v4l2_std_id	id	The bits in this field identify the standard as one of the common standards listed in *typedef v4l2_std_id*, or if bits 32 to 63 are set as custom standards. Multiple bits can be set if the hardware does not distinguish between these standards, however separate indices do not indicate the opposite. The id must be unique. No other enumerated struct *v4l2_standard* structure, for this input or output anyway, can contain the same set of bits.
__u8	name[24]	Name of the standard, a NUL-terminated ASCII string, for example: “PAL-B/G”, “NTSC Japan” . This information is intended for the user.
struct *v4l2_fract*	frameperiod	The frame period (not field period) is numerator / denominator. For example M/NTSC has a frame period of 1001 / 30000 seconds.
__u32	framelines	Total lines per frame including blanking, e. g. 625 for B/PAL.
__u32	reserved[4]	Reserved for future extensions. Drivers must set the array to zero.

[1] The supported standards may overlap and we need an unambiguous set to find the current standard returned by *VIDIOC_G_STD*.

type **v4l2_fract**

Table 152: struct v4l2_fract

__u32	numerator	
__u32	denominator	

Table 153: typedef v4l2_std_id

__u64	v4l2_std_id	This type is a set, each bit representing another video standard as listed below and in *Video Standards (based on itu470)*. The 32 most significant bits are reserved for custom (driver defined) video standards.

```
#define V4L2_STD_PAL_B          ((v4l2_std_id)0x00000001)
#define V4L2_STD_PAL_B1         ((v4l2_std_id)0x00000002)
#define V4L2_STD_PAL_G          ((v4l2_std_id)0x00000004)
#define V4L2_STD_PAL_H          ((v4l2_std_id)0x00000008)
#define V4L2_STD_PAL_I          ((v4l2_std_id)0x00000010)
#define V4L2_STD_PAL_D          ((v4l2_std_id)0x00000020)
#define V4L2_STD_PAL_D1         ((v4l2_std_id)0x00000040)
#define V4L2_STD_PAL_K          ((v4l2_std_id)0x00000080)

#define V4L2_STD_PAL_M          ((v4l2_std_id)0x00000100)
#define V4L2_STD_PAL_N          ((v4l2_std_id)0x00000200)
#define V4L2_STD_PAL_Nc         ((v4l2_std_id)0x00000400)
#define V4L2_STD_PAL_60         ((v4l2_std_id)0x00000800)
```

`V4L2_STD_PAL_60` is a hybrid standard with 525 lines, 60 Hz refresh rate, and PAL color modulation with a 4.43 MHz color subcarrier. Some PAL video recorders can play back NTSC tapes in this mode for display on a 50/60 Hz agnostic PAL TV.

```
#define V4L2_STD_NTSC_M         ((v4l2_std_id)0x00001000)
#define V4L2_STD_NTSC_M_JP      ((v4l2_std_id)0x00002000)
#define V4L2_STD_NTSC_443       ((v4l2_std_id)0x00004000)
```

`V4L2_STD_NTSC_443` is a hybrid standard with 525 lines, 60 Hz refresh rate, and NTSC color modulation with a 4.43 MHz color subcarrier.

```
#define V4L2_STD_NTSC_M_KR      ((v4l2_std_id)0x00008000)

#define V4L2_STD_SECAM_B        ((v4l2_std_id)0x00010000)
#define V4L2_STD_SECAM_D        ((v4l2_std_id)0x00020000)
#define V4L2_STD_SECAM_G        ((v4l2_std_id)0x00040000)
#define V4L2_STD_SECAM_H        ((v4l2_std_id)0x00080000)
#define V4L2_STD_SECAM_K        ((v4l2_std_id)0x00100000)
#define V4L2_STD_SECAM_K1       ((v4l2_std_id)0x00200000)
#define V4L2_STD_SECAM_L        ((v4l2_std_id)0x00400000)
#define V4L2_STD_SECAM_LC       ((v4l2_std_id)0x00800000)
```

(continues on next page)

(continued from previous page)

```
/* ATSC/HDTV */
#define V4L2_STD_ATSC_8_VSB     ((v4l2_std_id)0x01000000)
#define V4L2_STD_ATSC_16_VSB    ((v4l2_std_id)0x02000000)
```

V4L2_STD_ATSC_8_VSB and V4L2_STD_ATSC_16_VSB are U.S. terrestrial digital TV standards. Presently the V4L2 API does not support digital TV. See also the Linux DVB API at https://linuxtv.org.

```
#define V4L2_STD_PAL_BG         (V4L2_STD_PAL_B         |
                 V4L2_STD_PAL_B1        |
                 V4L2_STD_PAL_G)
#define V4L2_STD_B              (V4L2_STD_PAL_B         |
                 V4L2_STD_PAL_B1        |
                 V4L2_STD_SECAM_B)
#define V4L2_STD_GH             (V4L2_STD_PAL_G         |
                 V4L2_STD_PAL_H         |
                 V4L2_STD_SECAM_G       |
                 V4L2_STD_SECAM_H)
#define V4L2_STD_PAL_DK         (V4L2_STD_PAL_D         |
                 V4L2_STD_PAL_D1        |
                 V4L2_STD_PAL_K)
#define V4L2_STD_PAL            (V4L2_STD_PAL_BG        |
                 V4L2_STD_PAL_DK        |
                 V4L2_STD_PAL_H         |
                 V4L2_STD_PAL_I)
#define V4L2_STD_NTSC           (V4L2_STD_NTSC_M        |
                 V4L2_STD_NTSC_M_JP     |
                 V4L2_STD_NTSC_M_KR)
#define V4L2_STD_MN             (V4L2_STD_PAL_M         |
                 V4L2_STD_PAL_N         |
                 V4L2_STD_PAL_Nc        |
                 V4L2_STD_NTSC)
#define V4L2_STD_SECAM_DK       (V4L2_STD_SECAM_D       |
                 V4L2_STD_SECAM_K       |
                 V4L2_STD_SECAM_K1)
#define V4L2_STD_DK             (V4L2_STD_PAL_DK        |
                 V4L2_STD_SECAM_DK)

#define V4L2_STD_SECAM          (V4L2_STD_SECAM_B       |
                 V4L2_STD_SECAM_G       |
                 V4L2_STD_SECAM_H       |
                 V4L2_STD_SECAM_DK      |
                 V4L2_STD_SECAM_L       |
                 V4L2_STD_SECAM_LC)

#define V4L2_STD_525_60         (V4L2_STD_PAL_M         |
                 V4L2_STD_PAL_60        |
                 V4L2_STD_NTSC          |
```

(continues on next page)

(continued from previous page)

```
                 V4L2_STD_NTSC_443)
#define V4L2_STD_625_50         (V4L2_STD_PAL           |
                 V4L2_STD_PAL_N         |
                 V4L2_STD_PAL_Nc        |
                 V4L2_STD_SECAM)

#define V4L2_STD_UNKNOWN        0
#define V4L2_STD_ALL            (V4L2_STD_525_60        |
                 V4L2_STD_625_50)
```

Table 154: Video Standards (based on ITU BT.470)

Characteristics	M/NTSC[2]	M/PAL	N/PAL[3]	B, B1, G/PAL	D, D1, K/PAL	H/PAL	I/PAL	B, G/SECAM	D, K/SECAM	K1/SECAM	L/SECAM
Frame lines	525		625								
Frame period (s)	1001/30000		1/25								
Chrominance sub-carrier frequency (Hz)	3579545 ± 10	3579611.49 ± 10	4433618.75 ± 5 (3582056.25 ± 5)	4433618.75 ± 5				4433618.75 ± 1	f_{OR} = 4406250 ± 2000, f_{OB} = 4250000 ± 2000		
Nominal radio-frequency channel bandwidth (MHz)	6	6	6	B: 7; B1, G: 8	8	8	8	8	8	8	8
Sound carrier relative to vision carrier (MHz)	4.5	4.5	4.5	5.5 ± 0.001[4][5][6][7]	6.5 ± 0.001	5.5	5.9996 ± 0.0005	5.5 ± 0.001	6.5 ± 0.001	6.5	6.5[8]

Return Value

On success 0 is returned, on error -1 and the `errno` variable is set appropriately. The generic error codes are described at the *Generic Error Codes* chapter.

EINVAL
    The struct `v4l2_standard` index is out of bounds.

ENODATA
    Standard video timings are not supported for this input or output.

[2] Japan uses a standard similar to M/NTSC (V4L2_STD_NTSC_M_JP).

[3] The values in brackets apply to the combination N/PAL a.k.a. N_C used in Argentina (V4L2_STD_PAL_Nc).

[4] In the Federal Republic of Germany, Austria, Italy, the Netherlands, Slovakia and Switzerland a system of two sound carriers is used, the frequency of the second carrier being 242.1875 kHz above the frequency of the first sound carrier. For stereophonic sound transmissions a similar system is used in Australia.

[5] New Zealand uses a sound carrier displaced 5.4996 ± 0.0005 MHz from the vision carrier.

[6] In Denmark, Finland, New Zealand, Sweden and Spain a system of two sound carriers is used. In Iceland, Norway and Poland the same system is being introduced. The second carrier is 5.85 MHz above the vision carrier and is DQPSK modulated with 728 kbit/s sound and data multiplex. (NICAM system)

[7] In the United Kingdom, a system of two sound carriers is used. The second sound carrier is 6.552 MHz above the vision carrier and is DQPSK modulated with a 728 kbit/s sound and data multiplex able to carry two sound channels. (NICAM system)

[8] In France, a digital carrier 5.85 MHz away from the vision carrier may be used in addition to the main sound carrier. It is modulated in differentially encoded QPSK with a 728 kbit/s sound and data multiplexer capable of carrying two sound channels. (NICAM system)

ioctl VIDIOC_EXPBUF

Name

VIDIOC_EXPBUF - Export a buffer as a DMABUF file descriptor.

Synopsis

VIDIOC_EXPBUF

`int ioctl(int fd, VIDIOC_EXPBUF, struct v4l2_exportbuffer *argp)`

Arguments

fd
: File descriptor returned by *open()*.

argp
: Pointer to struct *v4l2_exportbuffer*.

Description

This ioctl is an extension to the *memory mapping* I/O method, therefore it is available only for `V4L2_MEMORY_MMAP` buffers. It can be used to export a buffer as a DMABUF file at any time after buffers have been allocated with the *ioctl VIDIOC_REQBUFS* ioctl.

To export a buffer, applications fill struct *v4l2_exportbuffer*. The `type` field is set to the same buffer type as was previously used with struct *v4l2_requestbuffers* `type`. Applications must also set the `index` field. Valid index numbers range from zero to the number of buffers allocated with *ioctl VIDIOC_REQBUFS* (struct *v4l2_requestbuffers* `count`) minus one. For the multi-planar API, applications set the `plane` field to the index of the plane to be exported. Valid planes range from zero to the maximal number of valid planes for the currently active format. For the single-planar API, applications must set `plane` to zero. Additional flags may be posted in the `flags` field. Refer to a manual for open() for details. Currently only O_CLOEXEC, O_RDONLY, O_WRONLY, and O_RDWR are supported. All other fields must be set to zero. In the case of multi-planar API, every plane is exported separately using multiple *ioctl VIDIOC_EXPBUF* calls.

After calling *ioctl VIDIOC_EXPBUF* the `fd` field will be set by a driver. This is a DMABUF file descriptor. The application may pass it to other DMABUF-aware devices. Refer to *DMABUF importing* for details about importing DMABUF files into V4L2 nodes. It is recommended to close a DMABUF file when it is no longer used to allow the associated memory to be reclaimed.

Examples

```
int buffer_export(int v4lfd, enum v4l2_buf_type bt, int index, int␣
↪*dmafd)
{
    struct v4l2_exportbuffer expbuf;

    memset(&expbuf, 0, sizeof(expbuf));
    expbuf.type = bt;
    expbuf.index = index;
    if (ioctl(v4lfd, VIDIOC_EXPBUF, &expbuf) == -1) {
        perror("VIDIOC_EXPBUF");
        return -1;
    }

    *dmafd = expbuf.fd;

    return 0;
}
```

```
int buffer_export_mp(int v4lfd, enum v4l2_buf_type bt, int index,
    int dmafd[], int n_planes)
{
    int i;

    for (i = 0; i < n_planes; ++i) {
        struct v4l2_exportbuffer expbuf;

        memset(&expbuf, 0, sizeof(expbuf));
        expbuf.type = bt;
        expbuf.index = index;
        expbuf.plane = i;
        if (ioctl(v4lfd, VIDIOC_EXPBUF, &expbuf) == -1) {
            perror("VIDIOC_EXPBUF");
            while (i)
                close(dmafd[--i]);
            return -1;
        }
        dmafd[i] = expbuf.fd;
    }

    return 0;
}
```

type **v4l2_exportbuffer**

Table 155: struct v4l2_exportbuffer

__u32	type	Type of the buffer, same as struct *v4l2_format* type or struct *v4l2_requestbuffers* type, set by the application. See *v4l2_buf_type*
__u32	index	Number of the buffer, set by the application. This field is only used for *memory mapping* I/O and can range from zero to the number of buffers allocated with the *ioctl VIDIOC_REQBUFS* and/or *ioctl VIDIOC_CREATE_BUFS* ioctls.
__u32	plane	Index of the plane to be exported when using the multi-planar API. Otherwise this value must be set to zero.
__u32	flags	Flags for the newly created file, currently only O_CLOEXEC, O_RDONLY, O_WRONLY, and O_RDWR are supported, refer to the manual of open() for more details.
__s32	fd	The DMABUF file descriptor associated with a buffer. Set by the driver.
__u32	reserved[11]	Reserved field for future use. Drivers and applications must set the array to zero.

Return Value

On success 0 is returned, on error -1 and the errno variable is set appropriately. The generic error codes are described at the *Generic Error Codes* chapter.

EINVAL
A queue is not in MMAP mode or DMABUF exporting is not supported or flags or type or index or plane fields are invalid.

ioctl VIDIOC_G_AUDIO, VIDIOC_S_AUDIO

Name

VIDIOC_G_AUDIO - VIDIOC_S_AUDIO - Query or select the current audio input and its attributes

Synopsis

VIDIOC_G_AUDIO

`int ioctl(int fd, VIDIOC_G_AUDIO, struct v4l2_audio *argp)`

VIDIOC_S_AUDIO

`int ioctl(int fd, VIDIOC_S_AUDIO, const struct v4l2_audio *argp)`

Arguments

fd
    File descriptor returned by *open()*.

argp
    Pointer to struct *v4l2_audio*.

Description

To query the current audio input applications zero out the `reserved` array of a struct *v4l2_audio* and call the *VIDIOC_G_AUDIO* ioctl with a pointer to this structure. Drivers fill the rest of the structure or return an `EINVAL` error code when the device has no audio inputs, or none which combine with the current video input.

Audio inputs have one writable property, the audio mode. To select the current audio input *and* change the audio mode, applications initialize the `index` and `mode` fields, and the `reserved` array of a struct *v4l2_audio* structure and call the *VIDIOC_S_AUDIO* ioctl. Drivers may switch to a different audio mode if the request cannot be satisfied. However, this is a write-only ioctl, it does not return the actual new audio mode.

type **v4l2_audio**

Table 156: struct v4l2_audio

__u32	index	Identifies the audio input, set by the driver or application.
__u8	name[32]	Name of the audio input, a NUL-terminated ASCII string, for example: "Line In". This information is intended for the user, preferably the connector label on the device itself.
__u32	capability	Audio capability flags, see *Audio Capability Flags*.
__u32	mode	Audio mode flags set by drivers and applications (on *VIDIOC_S_AUDIO* ioctl), see *Audio Mode Flags*.
__u32	reserved[2]	Reserved for future extensions. Drivers and applications must set the array to zero.

Table 157: Audio Capability Flags

V4L2_AUDCAP_STEREO	0x00001	This is a stereo input. The flag is intended to automatically disable stereo recording etc. when the signal is always monaural. The API provides no means to detect if stereo is *received*, unless the audio input belongs to a tuner.
V4L2_AUDCAP_AVL	0x00002	Automatic Volume Level mode is supported.

Table 158: Audio Mode Flags

V4L2_AUDMODE_AVL	0x00001	AVL mode is on.

Return Value

On success 0 is returned, on error -1 and the `errno` variable is set appropriately. The generic error codes are described at the *Generic Error Codes* chapter.

EINVAL
No audio inputs combine with the current video input, or the number of the selected audio input is out of bounds or it does not combine.

ioctl VIDIOC_G_AUDOUT, VIDIOC_S_AUDOUT

Name

VIDIOC_G_AUDOUT - VIDIOC_S_AUDOUT - Query or select the current audio output

Synopsis

VIDIOC_G_AUDOUT

`int ioctl(int fd, VIDIOC_G_AUDOUT, struct v4l2_audioout *argp)`

VIDIOC_S_AUDOUT

`int ioctl(int fd, VIDIOC_S_AUDOUT, const struct v4l2_audioout *argp)`

Arguments

fd
File descriptor returned by *open()*.

argp
Pointer to struct *v4l2_audioout*.

Description

To query the current audio output applications zero out the `reserved` array of a struct *v4l2_audioout* and call the `VIDIOC_G_AUDOUT` ioctl with a pointer to this structure. Drivers fill the rest of the structure or return an `EINVAL` error code when the device has no audio inputs, or none which combine with the current video output.

Audio outputs have no writable properties. Nevertheless, to select the current audio output applications can initialize the `index` field and `reserved` array (which in the future may contain writable properties) of a struct *v4l2_audioout* structure and call the `VIDIOC_S_AUDOUT` ioctl. Drivers switch to the requested output or return the `EINVAL` error code when the index is out of bounds. This is a write-only ioctl, it does not return the current audio output attributes as `VIDIOC_G_AUDOUT` does.

Note: Connectors on a TV card to loop back the received audio signal to a sound card are not audio outputs in this sense.

type **v4l2_audioout**

Table 159: struct v4l2_audioout

__u32	`index`	Identifies the audio output, set by the driver or application.
__u8	`name[32]`	Name of the audio output, a NUL-terminated ASCII string, for example: "Line Out". This information is intended for the user, preferably the connector label on the device itself.
__u32	`capability`	Audio capability flags, none defined yet. Drivers must set this field to zero.
__u32	`mode`	Audio mode, none defined yet. Drivers and applications (on `VIDIOC_S_AUDOUT`) must set this field to zero.
__u32	`reserved[2]`	Reserved for future extensions. Drivers and applications must set the array to zero.

Return Value

On success 0 is returned, on error -1 and the `errno` variable is set appropriately. The generic error codes are described at the *Generic Error Codes* chapter.

EINVAL
: No audio outputs combine with the current video output, or the number of the selected audio output is out of bounds or it does not combine.

ioctl VIDIOC_G_CROP, VIDIOC_S_CROP

Name

VIDIOC_G_CROP - VIDIOC_S_CROP - Get or set the current cropping rectangle

Synopsis

`VIDIOC_G_CROP`

`int ioctl(int fd, VIDIOC_G_CROP, struct v4l2_crop *argp)`

`VIDIOC_S_CROP`

`int ioctl(int fd, VIDIOC_S_CROP, const struct v4l2_crop *argp)`

Arguments

`fd`
: File descriptor returned by *`open()`*.

`argp`
: Pointer to struct *`v4l2_crop`*.

Description

To query the cropping rectangle size and position applications set the `type` field of a struct *`v4l2_crop`* structure to the respective buffer (stream) type and call the *VIDIOC_G_CROP* ioctl with a pointer to this structure. The driver fills the rest of the structure or returns the `EINVAL` error code if cropping is not supported.

To change the cropping rectangle applications initialize the `type` and struct *`v4l2_rect`* substructure named `c` of a v4l2_crop structure and call the *VIDIOC_S_CROP* ioctl with a pointer to this structure.

The driver first adjusts the requested dimensions against hardware limits, i. e. the bounds given by the capture/output window, and it rounds to the closest possible values of horizontal and vertical offset, width and height. In particular the driver must round the vertical offset of the cropping rectangle to frame lines modulo two, such that the field order cannot be confused.

Second the driver adjusts the image size (the opposite rectangle of the scaling process, source or target depending on the data direction) to the closest size possible while maintaining the current horizontal and vertical scaling factor.

Finally the driver programs the hardware with the actual cropping and image parameters. *VIDIOC_S_CROP* is a write-only ioctl, it does not return the actual parameters. To query them applications must call *VIDIOC_G_CROP* and *ioctl VIDIOC_G_FMT, VIDIOC_S_FMT, VIDIOC_TRY_FMT*. When the parameters are unsuitable the application may modify the cropping or image parameters and repeat the cycle until satisfactory parameters have been negotiated.

When cropping is not supported then no parameters are changed and *VIDIOC_S_CROP* returns the `EINVAL` error code.

type **v4l2_crop**

Table 160: struct v4l2_crop

__u32	`type`	Type of the data stream, set by the application. Only these types are valid here: `V4L2_BUF_TYPE_VIDEO_CAPTURE`, `V4L2_BUF_TYPE_VIDEO_CAPTURE_MPLANE`, `V4L2_BUF_TYPE_VIDEO_OUTPUT`, `V4L2_BUF_TYPE_VIDEO_OUTPUT_MPLANE` and `V4L2_BUF_TYPE_VIDEO_OVERLAY`. See *v4l2_buf_type* and the note below.
struct *v4l2_rect*	`c`	Cropping rectangle. The same co-ordinate system as for struct *v4l2_cropcap* `bounds` is used.

Note: Unfortunately in the case of multiplanar buffer types (`V4L2_BUF_TYPE_VIDEO_CAPTURE_MPLANE` and `V4L2_BUF_TYPE_VIDEO_OUTPUT_MPLANE`) this API was messed up with regards to how the *v4l2_crop* `type` field should be filled in. Some drivers only accepted the `_MPLANE` buffer type while other drivers only accepted a non-multiplanar buffer type (i.e. without the `_MPLANE` at the end).

Starting with kernel 4.13 both variations are allowed.

Return Value

On success 0 is returned, on error -1 and the `errno` variable is set appropriately. The generic error codes are described at the *Generic Error Codes* chapter.

ENODATA
    Cropping is not supported for this input or output.

ioctl VIDIOC_G_CTRL, VIDIOC_S_CTRL

Name

VIDIOC_G_CTRL - VIDIOC_S_CTRL - Get or set the value of a control

Synopsis

VIDIOC_G_CTRL

`int ioctl(int fd, VIDIOC_G_CTRL, struct v4l2_control *argp)`

VIDIOC_S_CTRL

`int ioctl(int fd, VIDIOC_S_CTRL, struct v4l2_control *argp)`

Arguments

fd
File descriptor returned by *open()*.

argp
Pointer to struct *v4l2_control*.

Description

To get the current value of a control applications initialize the `id` field of a struct *v4l2_control* and call the *VIDIOC_G_CTRL* ioctl with a pointer to this structure. To change the value of a control applications initialize the `id` and `value` fields of a struct *v4l2_control* and call the *VIDIOC_S_CTRL* ioctl.

When the `id` is invalid drivers return an `EINVAL` error code. When the `value` is out of bounds drivers can choose to take the closest valid value or return an `ERANGE` error code, whatever seems more appropriate. However, *VIDIOC_S_CTRL* is a write-only ioctl, it does not return the actual new value. If the `value` is inappropriate for the control (e.g. if it refers to an unsupported menu index of a menu control), then EINVAL error code is returned as well.

These ioctls work only with user controls. For other control classes the *VIDIOC_G_EXT_CTRLS*, *VIDIOC_S_EXT_CTRLS* or *VIDIOC_TRY_EXT_CTRLS* must be used.

type **v4l2_control**

Table 161: struct v4l2_control

__u32	`id`	Identifies the control, set by the application.
__s32	`value`	New value or current value.

Return Value

On success 0 is returned, on error -1 and the `errno` variable is set appropriately. The generic error codes are described at the *Generic Error Codes* chapter.

EINVAL
: The struct *v4l2_control* `id` is invalid or the `value` is inappropriate for the given control (i.e. if a menu item is selected that is not supported by the driver according to *VIDIOC_QUERYMENU*).

ERANGE
: The struct *v4l2_control* `value` is out of bounds.

EBUSY
: The control is temporarily not changeable, possibly because another applications took over control of the device function this control belongs to.

EACCES
: Attempt to set a read-only control or to get a write-only control.

ioctl VIDIOC_G_DV_TIMINGS, VIDIOC_S_DV_TIMINGS

Name

VIDIOC_G_DV_TIMINGS - VIDIOC_S_DV_TIMINGS - VIDIOC_SUBDEV_G_DV_TIMINGS - VIDIOC_SUBDEV_S_DV_TIMINGS - Get or set DV timings for input or output

Synopsis

VIDIOC_G_DV_TIMINGS

`int ioctl(int fd, VIDIOC_G_DV_TIMINGS, struct v4l2_dv_timings *argp)`

VIDIOC_S_DV_TIMINGS

`int ioctl(int fd, VIDIOC_S_DV_TIMINGS, struct v4l2_dv_timings *argp)`

VIDIOC_SUBDEV_G_DV_TIMINGS

`int ioctl(int fd, VIDIOC_SUBDEV_G_DV_TIMINGS, struct v4l2_dv_timings *argp)`

VIDIOC_SUBDEV_S_DV_TIMINGS

`int ioctl(int fd, VIDIOC_SUBDEV_S_DV_TIMINGS, struct v4l2_dv_timings *argp)`

Arguments

fd
File descriptor returned by *open()*.

argp
Pointer to struct *v4l2_dv_timings*.

Description

To set DV timings for the input or output, applications use the *VIDIOC_S_DV_TIMINGS* ioctl and to get the current timings, applications use the *VIDIOC_G_DV_TIMINGS* ioctl. The detailed timing information is filled in using the structure struct *v4l2_dv_timings*. These ioctls take a pointer to the struct *v4l2_dv_timings* structure as argument. If the ioctl is not supported or the timing values are not correct, the driver returns EINVAL error code.

Calling VIDIOC_SUBDEV_S_DV_TIMINGS on a subdev device node that has been registered in read-only mode is not allowed. An error is returned and the errno variable is set to -EPERM.

The linux/v4l2-dv-timings.h header can be used to get the timings of the formats in the *CEA-861-E* and *VESA DMT* standards. If the current input or output does not support DV timings (e.g. if *ioctl VIDIOC_ENUMINPUT* does not set the V4L2_IN_CAP_DV_TIMINGS flag), then ENODATA error code is returned.

Return Value

On success 0 is returned, on error -1 and the errno variable is set appropriately. The generic error codes are described at the *Generic Error Codes* chapter.

EINVAL
This ioctl is not supported, or the *VIDIOC_S_DV_TIMINGS* parameter was unsuitable.

ENODATA
Digital video timings are not supported for this input or output.

EBUSY
The device is busy and therefore can not change the timings.

EPERM
VIDIOC_SUBDEV_S_DV_TIMINGS has been called on a read-only subdevice.

type **v4l2_bt_timings**

Table 162: struct v4l2_bt_timings

__u32	width	Width of the active video in pixels.
__u32	height	Height of the active video frame in lines. So for interlaced formats the height of the active video in each field is height/2.
__u32	interlaced	Progressive (V4L2_DV_PROGRESSIVE) or interlaced (V4L2_DV_INTERLACED).
__u32	polarities	This is a bit mask that defines polarities of sync signals. bit 0 (V4L2_DV_VSYNC_POS_POL) is for vertical sync polarity and bit 1 (V4L2_DV_HSYNC_POS_POL) is for horizontal sync polarity. If the bit is set (1) it is positive polarity and if is cleared (0), it is negative polarity.
__u64	pixelclock	Pixel clock in Hz. Ex. 74.25MHz->74250000
__u32	hfrontporch	Horizontal front porch in pixels
__u32	hsync	Horizontal sync length in pixels
__u32	hbackporch	Horizontal back porch in pixels
__u32	vfrontporch	Vertical front porch in lines. For interlaced formats this refers to the odd field (aka field 1).
__u32	vsync	Vertical sync length in lines. For interlaced formats this refers to the odd field (aka field 1).
__u32	vbackporch	Vertical back porch in lines. For interlaced formats this refers to the odd field (aka field 1).
__u32	il_vfrontporch	Vertical front porch in lines for the even field (aka field 2) of interlaced field formats. Must be 0 for progressive formats.
__u32	il_vsync	Vertical sync length in lines for the even field (aka field 2) of interlaced field formats. Must be 0 for progressive formats.
__u32	il_vbackporch	Vertical back porch in lines for the even field (aka field 2) of interlaced field formats. Must be 0 for progressive formats.
__u32	standards	The video standard(s) this format belongs to. This will be filled in by the driver. Applications must set this to 0. See *DV BT Timing standards* for a list of standards.
__u32	flags	Several flags giving more information about the format. See *DV BT Timing flags* for a description of the flags.
struct *v4l2_fract*	picture_aspect	The picture aspect if the pixels are not square. Only valid if the V4L2_DV_FL_HAS_PICTURE_ASPECT flag is set.
__u8	cea861_vic	The Video Identification Code according to the CEA-861 standard. Only valid if the V4L2_DV_FL_HAS_CEA861_VIC flag is set.
__u8	hdmi_vic	The Video Identification Code according to the HDMI standard. Only valid if the V4L2_DV_FL_HAS_HDMI_VIC flag is set.
__u8	reserved[46]	Reserved for future extensions. Drivers and applications must set the array to zero.

type **v4l2_dv_timings**

Table 163: struct v4l2_dv_timings

__u32	type	Type of DV timings as listed in *DV Timing types*.
union {	(anonymous)	
struct *v4l2_bt_timings*	bt	Timings defined by BT.656/1120 specifications
__u32	reserved[32]	
}		

Table 164: DV Timing types

Timing type	value	Description
V4L2_DV_BT_656_1120	0	BT.656/1120 timings

Table 165: DV BT Timing standards

Timing standard	Description
V4L2_DV_BT_STD_CEA86	The timings follow the CEA-861 Digital TV Profile standard
V4L2_DV_BT_STD_DMT	The timings follow the VESA Discrete Monitor Timings standard
V4L2_DV_BT_STD_CVT	The timings follow the VESA Coordinated Video Timings standard
V4L2_DV_BT_STD_GTF	The timings follow the VESA Generalized Timings Formula standard
V4L2_DV_BT_STD_SDI	The timings follow the SDI Timings standard. There are no horizontal syncs/porches at all in this format. Total blanking timings must be set in hsync or vsync fields only.

Table 166: DV BT Timing flags

Flag	Description
V4L2_DV_FL_REDUCED_BLANKING	CVT/GTF specific: the timings use reduced blanking (CVT) or the ‘Secondary GTF’ curve (GTF). In both cases the horizontal and/or vertical blanking intervals are reduced, allowing a higher resolution over the same bandwidth. This is a read-only flag, applications must not set this.
V4L2_DV_FL_CAN_REDUCE_FPS	CEA-861 specific: set for CEA-861 formats with a framerate that is a multiple of six. These formats can be optionally played at 1 / 1.001 speed to be compatible with 60 Hz based standards such as NTSC and PAL-M that use a framerate of 29.97 frames per second. If the transmitter can’t generate such frequencies, then the flag will also be cleared. This is a read-only flag, applications must not set this.
V4L2_DV_FL_REDUCED_FPS	CEA-861 specific: only valid for video transmitters or video receivers that have the V4L2_DV_FL_CAN_DETECT_REDUCED_FPS set. This flag is cleared otherwise. It is also only valid for formats with the V4L2_DV_FL_CAN_REDUCE_FPS flag set, for other formats the flag will be cleared by the driver. If the application sets this flag for a transmitter, then the pixelclock used to set up the transmitter is divided by 1.001 to make it compatible with NTSC framerates. If the transmitter can’t generate such frequencies, then the flag will be cleared. If a video receiver detects that the format uses a reduced framerate, then it will set this flag to signal this to the application.
V4L2_DV_FL_HALF_LINE	Specific to interlaced formats: if set, then the vertical frontporch of field 1 (aka the odd field) is really one half-line longer and the vertical backporch of field 2 (aka the even field) is really one half-line shorter, so each field has exactly the same number of half-lines. Whether half-lines can be detected or used depends on the hardware.
V4L2_DV_FL_IS_CE_VIDEO	If set, then this is a Consumer Electronics (CE) video format. Such formats differ from other formats (commonly called IT formats) in that if R’G’B’ encoding is used then by default the R’G’B’ values use limited range (i.e. 16-235) as opposed to full range (i.e. 0-255). All formats defined in CEA-861 except for the 640x480p59.94 format are CE formats.
V4L2_DV_FL_FIRST_FIELD_EXTRA_LI	Some formats like SMPTE-125M have an interlaced signal with a odd total height. For these formats, if this flag is set, the first field has the extra line. Else, it is the second field.
V4L2_DV_FL_HAS_PICTURE_ASPECT	If set, then the picture_aspect field is valid. Otherwise assume that the pixels are square, so the picture aspect ratio is the same as the width to height ratio.
V4L2_DV_FL_HAS_CEA861_VIC	If set, then the cea861_vic field is valid and contains the Video Identification Code as per the CEA-861 standard.
V4L2_DV_FL_HAS_HDMI_VIC	If set, then the hdmi_vic field is valid and contains the

ioctl VIDIOC_G_EDID, VIDIOC_S_EDID, VIDIOC_SUBDEV_G_EDID, VIDIOC_SUBDEV_S_EDID

Name

VIDIOC_G_EDID - VIDIOC_S_EDID - VIDIOC_SUBDEV_G_EDID - VIDIOC_SUBDEV_S_EDID - Get or set the EDID of a video receiver/transmitter

Synopsis

VIDIOC_G_EDID

`int ioctl(int fd, VIDIOC_G_EDID, struct v4l2_edid *argp)`

VIDIOC_S_EDID

`int ioctl(int fd, VIDIOC_S_EDID, struct v4l2_edid *argp)`

VIDIOC_SUBDEV_G_EDID

`int ioctl(int fd, VIDIOC_SUBDEV_G_EDID, struct v4l2_edid *argp)`

VIDIOC_SUBDEV_S_EDID

`int ioctl(int fd, VIDIOC_SUBDEV_S_EDID, struct v4l2_edid *argp)`

Arguments

fd
: File descriptor returned by *open()*.

argp
: Pointer to struct *v4l2_edid*.

Description

These ioctls can be used to get or set an EDID associated with an input from a receiver or an output of a transmitter device. They can be used with subdevice nodes (/dev/v4l-subdevX) or with video nodes (/dev/videoX).

When used with video nodes the pad field represents the input (for video capture devices) or output (for video output devices) index as is returned by *ioctl VIDIOC_ENUMINPUT* and *ioctl VIDIOC_ENUMOUTPUT* respectively. When used with subdevice nodes the pad field represents the input or output pad of the subdevice. If there is no EDID support for the given pad value, then the EINVAL error code will be returned.

To get the EDID data the application has to fill in the pad, start_block, blocks and edid fields, zero the reserved array and call *VIDIOC_G_EDID*. The current EDID from block start_block and of size blocks will be placed in the memory edid points to. The edid pointer must point to memory at least blocks * 128 bytes large (the size of one block is 128 bytes).

If there are fewer blocks than specified, then the driver will set `blocks` to the actual number of blocks. If there are no EDID blocks available at all, then the error code `ENODATA` is set.

If blocks have to be retrieved from the sink, then this call will block until they have been read.

If `start_block` and `blocks` are both set to 0 when *VIDIOC_G_EDID* is called, then the driver will set `blocks` to the total number of available EDID blocks and it will return 0 without copying any data. This is an easy way to discover how many EDID blocks there are.

Note: If there are no EDID blocks available at all, then the driver will set `blocks` to 0 and it returns 0.

To set the EDID blocks of a receiver the application has to fill in the `pad`, `blocks` and `edid` fields, set `start_block` to 0 and zero the `reserved` array. It is not possible to set part of an EDID, it is always all or nothing. Setting the EDID data is only valid for receivers as it makes no sense for a transmitter.

The driver assumes that the full EDID is passed in. If there are more EDID blocks than the hardware can handle then the EDID is not written, but instead the error code `E2BIG` is set and `blocks` is set to the maximum that the hardware supports. If `start_block` is any value other than 0 then the error code `EINVAL` is set.

To disable an EDID you set `blocks` to 0. Depending on the hardware this will drive the hotplug pin low and/or block the source from reading the EDID data in some way. In any case, the end result is the same: the EDID is no longer available.

type **`v4l2_edid`**

Table 167: struct v4l2_edid

__u32	`pad`	Pad for which to get/set the EDID blocks. When used with a video device node the pad represents the input or output index as returned by *ioctl VIDIOC_ENUMINPUT* and *ioctl VIDIOC_ENUMOUTPUT* respectively.
__u32	`start_block`	Read the EDID from starting with this block. Must be 0 when setting the EDID.
__u32	`blocks`	The number of blocks to get or set. Must be less or equal to 256 (the maximum number of blocks as defined by the standard). When you set the EDID and `blocks` is 0, then the EDID is disabled or erased.
__u32	`reserved[5]`	Reserved for future extensions. Applications and drivers must set the array to zero.
__u8 *	`edid`	Pointer to memory that contains the EDID. The minimum size is `blocks` * 128.

Return Value

On success 0 is returned, on error -1 and the `errno` variable is set appropriately. The generic error codes are described at the *Generic Error Codes* chapter.

ENODATA
The EDID data is not available.

E2BIG
The EDID data you provided is more than the hardware can handle.

ioctl VIDIOC_G_ENC_INDEX

Name

VIDIOC_G_ENC_INDEX - Get meta data about a compressed video stream

Synopsis

VIDIOC_G_ENC_INDEX

`int ioctl(int fd, VIDIOC_G_ENC_INDEX, struct v4l2_enc_idx *argp)`

Arguments

fd
File descriptor returned by *open()*.

argp
Pointer to struct *v4l2_enc_idx*.

Description

The *VIDIOC_G_ENC_INDEX* ioctl provides meta data about a compressed video stream the same or another application currently reads from the driver, which is useful for random access into the stream without decoding it.

To read the data applications must call *VIDIOC_G_ENC_INDEX* with a pointer to a struct *v4l2_enc_idx*. On success the driver fills the `entry` array, stores the number of elements written in the `entries` field, and initializes the `entries_cap` field.

Each element of the `entry` array contains meta data about one picture. A *VIDIOC_G_ENC_INDEX* call reads up to `V4L2_ENC_IDX_ENTRIES` entries from a driver buffer, which can hold up to `entries_cap` entries. This number can be lower or higher than `V4L2_ENC_IDX_ENTRIES`, but not zero. When the application fails to read the meta data in time the oldest entries will be lost. When the buffer is empty or no capturing/encoding is in progress, `entries` will be zero.

Currently this ioctl is only defined for MPEG-2 program streams and video elementary streams.

type **v4l2_enc_idx**

Table 168: struct v4l2_enc_idx

__u32	entries	The number of entries the driver stored in the entry array.
__u32	entries_cap	The number of entries the driver can buffer. Must be greater than zero.
__u32	reserved[4]	Reserved for future extensions. Drivers must set the array to zero.
struct *v4l2_enc_idx_entr*	entry[V4L2_ENC_IDX_ENTRIE	Meta data about a compressed video stream. Each element of the array corresponds to one picture, sorted in ascending order by their offset.

type **v4l2_enc_idx_entry**

Table 169: struct v4l2_enc_idx_entry

__u64	offset	The offset in bytes from the beginning of the compressed video stream to the beginning of this picture, that is a *PES packet header* as defined in *ISO 13818-1* or a *picture header* as defined in *ISO 13818-2*. When the encoder is stopped, the driver resets the offset to zero.
__u64	pts	The 33 bit *Presentation Time Stamp* of this picture as defined in *ISO 13818-1*.
__u32	length	The length of this picture in bytes.
__u32	flags	Flags containing the coding type of this picture, see *Index Entry Flags*.
__u32	reserved[2]	Reserved for future extensions. Drivers must set the array to zero.

Table 170: Index Entry Flags

V4L2_ENC_IDX_FRAME_I	0x00	This is an Intra-coded picture.
V4L2_ENC_IDX_FRAME_P	0x01	This is a Predictive-coded picture.
V4L2_ENC_IDX_FRAME_B	0x02	This is a Bidirectionally predictive-coded picture.
V4L2_ENC_IDX_FRAME_MASK	0x0F	*AND* the flags field with this mask to obtain the picture coding type.

Return Value

On success 0 is returned, on error -1 and the `errno` variable is set appropriately. The generic error codes are described at the *Generic Error Codes* chapter.

ioctl VIDIOC_G_EXT_CTRLS, VIDIOC_S_EXT_CTRLS, VIDIOC_TRY_EXT_CTRLS

Name

VIDIOC_G_EXT_CTRLS - VIDIOC_S_EXT_CTRLS - VIDIOC_TRY_EXT_CTRLS - Get or set the value of several controls, try control values

Synopsis

VIDIOC_G_EXT_CTRLS

`int ioctl(int fd, VIDIOC_G_EXT_CTRLS, struct v4l2_ext_controls *argp)`

VIDIOC_S_EXT_CTRLS

`int ioctl(int fd, VIDIOC_S_EXT_CTRLS, struct v4l2_ext_controls *argp)`

VIDIOC_TRY_EXT_CTRLS

`int ioctl(int fd, VIDIOC_TRY_EXT_CTRLS, struct v4l2_ext_controls *argp)`

Arguments

fd
File descriptor returned by *open()*.

argp
Pointer to struct *v4l2_ext_controls*.

Description

These ioctls allow the caller to get or set multiple controls atomically. Control IDs are grouped into control classes (see *Control classes*) and all controls in the control array must belong to the same control class.

Applications must always fill in the `count`, `which`, `controls` and `reserved` fields of struct *v4l2_ext_controls*, and initialize the struct *v4l2_ext_control* array pointed to by the `controls` fields.

To get the current value of a set of controls applications initialize the `id`, `size` and `reserved2` fields of each struct *v4l2_ext_control* and call the *VIDIOC_G_EXT_CTRLS* ioctl. String controls controls must also set the `string` field.

Controls of compound types (V4L2_CTRL_FLAG_HAS_PAYLOAD is set) must set the ptr field.

If the size is too small to receive the control result (only relevant for pointer-type controls like strings), then the driver will set size to a valid value and return an ENOSPC error code. You should re-allocate the memory to this new size and try again. For the string type it is possible that the same issue occurs again if the string has grown in the meantime. It is recommended to call *ioctls VIDIOC_QUERYCTRL, VIDIOC_QUERY_EXT_CTRL and VIDIOC_QUERYMENU* first and use maximum+1 as the new size value. It is guaranteed that that is sufficient memory.

N-dimensional arrays are set and retrieved row-by-row. You cannot set a partial array, all elements have to be set or retrieved. The total size is calculated as elems * elem_size. These values can be obtained by calling *VIDIOC_QUERY_EXT_CTRL*.

To change the value of a set of controls applications initialize the id, size, reserved2 and value/value64/string/ptr fields of each struct *v4l2_ext_control* and call the *VIDIOC_S_EXT_CTRLS* ioctl. The controls will only be set if *all* control values are valid.

To check if a set of controls have correct values applications initialize the id, size, reserved2 and value/value64/string/ptr fields of each struct *v4l2_ext_control* and call the *VIDIOC_TRY_EXT_CTRLS* ioctl. It is up to the driver whether wrong values are automatically adjusted to a valid value or if an error is returned.

When the id or which is invalid drivers return an EINVAL error code. When the value is out of bounds drivers can choose to take the closest valid value or return an ERANGE error code, whatever seems more appropriate. In the first case the new value is set in struct *v4l2_ext_control*. If the new control value is inappropriate (e.g. the given menu index is not supported by the menu control), then this will also result in an EINVAL error code error.

If request_fd is set to a not-yet-queued *request* file descriptor and which is set to V4L2_CTRL_WHICH_REQUEST_VAL, then the controls are not applied immediately when calling *VIDIOC_S_EXT_CTRLS*, but instead are applied by the driver for the buffer associated with the same request. If the device does not support requests, then EACCES will be returned. If requests are supported but an invalid request file descriptor is given, then EINVAL will be returned.

An attempt to call *VIDIOC_S_EXT_CTRLS* for a request that has already been queued will result in an EBUSY error.

If request_fd is specified and which is set to V4L2_CTRL_WHICH_REQUEST_VAL during a call to *VIDIOC_G_EXT_CTRLS*, then it will return the values of the controls at the time of request completion. If the request is not yet completed, then this will result in an EACCES error.

The driver will only set/get these controls if all control values are correct. This prevents the situation where only some of the controls were set/get. Only low-level errors (e. g. a failed i2c command) can still cause this situation.

type **v4l2_ext_control**

Table 171: struct v4l2_ext_control

__u32	id	Identifies the control, set by the application.
__u32	size	The total size in bytes of the payload of this control. This is normally 0, but for pointer controls this should be set to the size of the memory containing the payload, or that will receive the payload. If *VIDIOC_G_EXT_CTRLS* finds that this value is less than is required

type **v4l2_ext_controls**

Table 172: struct v4l2_ext_controls

union {	(anonymous)	
__u32	ctrl_class	The control class to which all controls belong, see *Control classes*. Drivers that use a kernel framework for handling controls will also accept a value of 0 here, meaning that the controls can belong to any control class. Whether drivers support this can be tested by setting ctrl_class to 0 and calling *VIDIOC_TRY_EXT_CTRLS* with a count of 0. If that succeeds, then the driver supports this feature.

continues on next page

Table 172 – continued from previous page

__u32	which	Which value of the control to get/set/try. V4L2_CTRL_WHICH_CUR_VAL will return the current value of the control, V4L2_CTRL_WHICH_DEF_VAL will return the default value of the control and V4L2_CTRL_WHICH_REQUEST_VAL indicates that these controls have to be retrieved from a request or tried/set for a request. In the latter case the request_fd field contains the file descriptor of the request that should be used. If the device does not support requests, then EACCES will be returned. **Note:** When using V4L2_CTRL_WHICH_DEF_VAL be aware that you can only get the de-

Table 172 – continued from previous page

}		
__u32	count	The number of controls in the controls array. May also be zero.

continues on next page

Table 172 – continued from previous page

__u32	error_idx	Set by the driver in case of an error. If the error is associated with a particular control, then error_idx is set to the index of that control. If the error is not related to a specific control, or the validation step failed (see below), then error_idx is set to count. The value is undefined if the ioctl returned 0 (success). Before controls are read from/written to hardware a validation step takes place: this checks if all controls in the list are valid controls, if no attempt is made to write to a read-only control or read from a

Table 172 – continued from previous page

__s32	request_fd	File descriptor of the request to be used by this operation. Only valid if `which` is set to `V4L2_CTRL_WHICH_REQUEST_VAL`. If the device does not support requests, then `EACCES` will be returned. If requests are supported but an invalid request file descriptor is given, then `EINVAL` will be returned.
__u32	reserved[1	Reserved for future extensions. Drivers and applications must set the array to zero.

continues on next page

Table 172 – continued from previous page

struct *v4l2_ext_control* *	controls	Pointer to an array of count v4l2_ext_control structures. Ignored if count equals zero.

Table 173: Control classes

V4L2_CTRL_CLASS_USER	0x980000	The class containing user controls. These controls are described in *User Controls*. All controls that can be set using the *VIDIOC_S_CTRL* and *VIDIOC_G_CTRL* ioctl belong to this class.
V4L2_CTRL_CLASS_MPEG	0x990000	The class containing MPEG compression controls. These controls are described in *Codec Control Reference*.
V4L2_CTRL_CLASS_CAMERA	0x9a0000	The class containing camera controls. These controls are described in *Camera Control Reference*.
V4L2_CTRL_CLASS_FM_TX	0x9b0000	The class containing FM Transmitter (FM TX) controls. These controls are described in *FM Transmitter Control Reference*.
V4L2_CTRL_CLASS_FLASH	0x9c0000	The class containing flash device controls. These controls are described in *Flash Control Reference*.
V4L2_CTRL_CLASS_JPEG	0x9d0000	The class containing JPEG compression controls. These controls are described in *JPEG Control Reference*.
V4L2_CTRL_CLASS_IMAGE_SOURCE	0x9e0000	The class containing image source controls. These controls are described in *Image Source Control Reference*.
V4L2_CTRL_CLASS_IMAGE_PROC	0x9f0000	The class containing image processing controls. These controls are described in *Image Process Control Reference*.
V4L2_CTRL_CLASS_FM_RX	0xa10000	The class containing FM Receiver (FM RX) controls. These controls are described in *FM Receiver Control Reference*.
V4L2_CTRL_CLASS_RF_TUNER	0xa20000	The class containing RF tuner controls. These controls are described in *RF Tuner Control Reference*.

Return Value

On success 0 is returned, on error -1 and the `errno` variable is set appropriately. The generic error codes are described at the *Generic Error Codes* chapter.

EINVAL
: The struct *v4l2_ext_control* `id` is invalid, or the struct *v4l2_ext_controls* `which` is invalid, or the struct *v4l2_ext_control* `value` was inappropriate (e.g. the given menu index is not supported by the driver), or the `which` field was set to `V4L2_CTRL_WHICH_REQUEST_VAL` but the given `request_fd` was invalid or `V4L2_CTRL_WHICH_REQUEST_VAL` is not supported by the kernel. This error code is also returned by the *VIDIOC_S_EXT_CTRLS* and *VIDIOC_TRY_EXT_CTRLS* ioctls if two or more control values are in conflict.

ERANGE
: The struct *v4l2_ext_control* `value` is out of bounds.

EBUSY
: The control is temporarily not changeable, possibly because another applications took over control of the device function this control belongs to, or (if the `which` field was set to `V4L2_CTRL_WHICH_REQUEST_VAL`) the request was queued but not yet completed.

ENOSPC
: The space reserved for the control's payload is insufficient. The field `size` is set to a value that is enough to store the payload and this error code is returned.

EACCES
: Attempt to try or set a read-only control, or to get a write-only control, or to get a control from a request that has not yet been completed.

  Or the `which` field was set to `V4L2_CTRL_WHICH_REQUEST_VAL` but the device does not support requests.

ioctl VIDIOC_G_FBUF, VIDIOC_S_FBUF

Name

VIDIOC_G_FBUF - VIDIOC_S_FBUF - Get or set frame buffer overlay parameters

Synopsis

`VIDIOC_G_FBUF`

`int ioctl(int fd, VIDIOC_G_FBUF, struct v4l2_framebuffer *argp)`

`VIDIOC_S_FBUF`

`int ioctl(int fd, VIDIOC_S_FBUF, const struct v4l2_framebuffer *argp)`

Arguments

fd
File descriptor returned by *open()*.

argp
Pointer to struct *v4l2_framebuffer*.

Description

Applications can use the *VIDIOC_G_FBUF* and *VIDIOC_S_FBUF* ioctl to get and set the framebuffer parameters for a *Video Overlay* or *Video Output Overlay* (OSD). The type of overlay is implied by the device type (capture or output device) and can be determined with the *ioctl VIDIOC_QUERYCAP* ioctl. One /dev/videoN device must not support both kinds of overlay.

The V4L2 API distinguishes destructive and non-destructive overlays. A destructive overlay copies captured video images into the video memory of a graphics card. A non-destructive overlay blends video images into a VGA signal or graphics into a video signal. *Video Output Overlays* are always non-destructive.

To get the current parameters applications call the *VIDIOC_G_FBUF* ioctl with a pointer to a struct *v4l2_framebuffer* structure. The driver fills all fields of the structure or returns an EINVAL error code when overlays are not supported.

To set the parameters for a *Video Output Overlay*, applications must initialize the `flags` field of a struct *v4l2_framebuffer*. Since the framebuffer is implemented on the TV card all other parameters are determined by the driver. When an application calls *VIDIOC_S_FBUF* with a pointer to this structure, the driver prepares for the overlay and returns the framebuffer parameters as *VIDIOC_G_FBUF* does, or it returns an error code.

To set the parameters for a *non-destructive Video Overlay*, applications must initialize the `flags` field, the `fmt` substructure, and call *VIDIOC_S_FBUF*. Again the driver prepares for the overlay and returns the framebuffer parameters as *VIDIOC_G_FBUF* does, or it returns an error code.

For a *destructive Video Overlay* applications must additionally provide a `base` address. Setting up a DMA to a random memory location can jeopardize the system security, its stability or even damage the hardware, therefore only the superuser can set the parameters for a destructive video overlay.

type **v4l2_framebuffer**

Table 174: struct v4l2_framebuffer

__u32	capability		Overlay capability flags set by the driver, see *Frame Buffer Capability Flags*.
__u32	flags		Overlay control flags set by application and driver, see *Frame Buffer Flags*

continues on next page

Table 174 – continued from previous page

void *	base		Physical base address of the framebuffer, that is the address of the pixel in the top left corner of the framebuffer.[1]
			This field is irrelevant to *non-destructive Video Overlays*. For *destructive Video Overlays* applications must provide a base address. The driver may accept only base addresses which are a multiple of two, four or eight bytes. For *Video Output Overlays* the driver must return a valid base address, so applications can find the corresponding Linux framebuffer device (see *Video Output Overlay Interface*).
struct	fmt		Layout of the frame buffer.
	__u32	width	Width of the frame buffer in pixels.
	__u32	height	Height of the frame buffer in pixels.
	__u32	pixelformat	The pixel format of the framebuffer.
			For *non-destructive Video Overlays* this field only defines a format for the struct *v4l2_window* chromakey field.
			For *destructive Video Overlays* applications must initialize this field. For *Video Output Overlays* the driver must return a valid format.
			Usually this is an RGB format (for example *V4L2_PIX_FMT_RGB565*) but YUV formats (only packed YUV formats when chroma keying is used, not including V4L2_PIX_FMT_YUYV and V4L2_PIX_FMT_UYVY) and the V4L2_PIX_FMT_PAL8 format are also permitted. The behavior of the driver when an application requests a compressed format is undefined. See *Image Formats* for information on pixel formats.
	enum *v4l2_field*	field	Drivers and applications shall ignore this field. If applicable, the field order is selected with the *VIDIOC_S_FMT* ioctl, using the field field of struct *v4l2_window*.
	__u32	bytesperline	Distance in bytes between the leftmost pixels in two adjacent lines.

continues on next page

Table 174 – continued from previous page

This field is irrelevant to *non-destructive Video Overlays*. For *destructive Video Overlays* both applications and drivers can set this field to request padding bytes at the end of each line. Drivers however may ignore the requested value, returning `width` times bytes-per-pixel or a larger value required by the hardware. That implies applications can just set this field to zero to get a reasonable default. For *Video Output Overlays* the driver must return a valid value. Video hardware may access padding bytes, therefore they must reside in accessible memory. Consider for example the case where padding bytes after the last line of an image cross a system page boundary. Capture devices may write padding bytes, the value is undefined. Output devices ignore the contents of padding bytes. When the image format is planar the `bytesperline` value applies to the first plane and is divided by the same factor as the `width` field for the other planes. For example the Cb and Cr planes of a YUV 4:2:0 image have half as many padding bytes following each line as the Y plane. To avoid ambiguities drivers must return a `bytesperline` value rounded up to a multiple of the scale factor.			
	__u32	sizeimage	This field is irrelevant to *non-destructive Video Overlays*. For *destructive Video Overlays* applications must initialize this field. For *Video Output Overlays* the driver must return a valid format. Together with `base` it defines the framebuffer memory accessible by the driver.
	enum *v4l2_colorspace*	colorspace	This information supplements the `pixelformat` and must be set by the driver, see *Colorspaces*.
	__u32	priv	Reserved. Drivers and applications must set this field to zero.

[1] A physical base address may not suit all platforms. GK notes in theory we should pass something like PCI device + memory region + offset instead. If you encounter problems please discuss on the linux-media mailing list: https://linuxtv.org/lists.php.

Table 175: Frame Buffer Capability Flags

V4L2_FBUF_CAP_EXTERNOVERLAY	0x0001	The device is capable of non-destructive overlays. When the driver clears this flag, only destructive overlays are supported. There are no drivers yet which support both destructive and non-destructive overlays. Video Output Overlays are in practice always non-destructive.
V4L2_FBUF_CAP_CHROMAKEY	0x0002	The device supports clipping by chroma-keying the images. That is, image pixels replace pixels in the VGA or video signal only where the latter assume a certain color. Chroma-keying makes no sense for destructive overlays.
V4L2_FBUF_CAP_LIST_CLIPPING	0x0004	The device supports clipping using a list of clip rectangles.
V4L2_FBUF_CAP_BITMAP_CLIPPING	0x0008	The device supports clipping using a bit mask.
V4L2_FBUF_CAP_LOCAL_ALPHA	0x0010	The device supports clipping/blending using the alpha channel of the framebuffer or VGA signal. Alpha blending makes no sense for destructive overlays.
V4L2_FBUF_CAP_GLOBAL_ALPHA	0x0020	The device supports alpha blending using a global alpha value. Alpha blending makes no sense for destructive overlays.
V4L2_FBUF_CAP_LOCAL_INV_ALPHA	0x0040	The device supports clipping/blending using the inverted alpha channel of the framebuffer or VGA signal. Alpha blending makes no sense for destructive overlays.
V4L2_FBUF_CAP_SRC_CHROMAKEY	0x0080	The device supports Source Chroma-keying. Video pixels with the chroma-key colors are replaced by framebuffer pixels, which is exactly opposite of V4L2_FBUF_CAP_CHROMAKEY

Table 176: Frame Buffer Flags

V4L2_FBUF_FLAG_PRIMARY	0x0001	The framebuffer is the primary graphics surface. In other words, the overlay is destructive. This flag is typically set by any driver that doesn’t have the V4L2_FBUF_CAP_EXTERNOVERLAY capability and it is cleared otherwise.

continues on next page

Table 176 – continued from previous page

V4L2_FBUF_FLAG_OVERLAY	0x0002	If this flag is set for a video capture device, then the driver will set the initial overlay size to cover the full framebuffer size, otherwise the existing overlay size (as set by *VIDIOC_S_FMT*) will be used. Only one video capture driver (bttv) supports this flag. The use of this flag for capture devices is deprecated. There is no way to detect which drivers support this flag, so the only reliable method of setting the overlay size is through *VIDIOC_S_FMT*. If this flag is set for a video output device, then the video output overlay window is relative to the top-left corner of the framebuffer and restricted to the size of the framebuffer. If it is cleared, then the video output overlay window is relative to the video output display.
V4L2_FBUF_FLAG_CHROMAKEY	0x0004	Use chroma-keying. The chroma-key color is determined by the `chromakey` field of struct *v4l2_window* and negotiated with the *VIDIOC_S_FMT* ioctl, see *Video Overlay Interface* and *Video Output Overlay Interface*.
There are no flags to enable clipping using a list of clip rectangles or a bitmap. These methods are negotiated with the *VIDIOC_S_FMT* ioctl, see *Video Overlay Interface* and *Video Output Overlay Interface*.		
V4L2_FBUF_FLAG_LOCAL_ALPHA	0x0008	Use the alpha channel of the framebuffer to clip or blend framebuffer pixels with video images. The blend function is: output = framebuffer pixel * alpha + video pixel * (1 - alpha). The actual alpha depth depends on the framebuffer pixel format.
V4L2_FBUF_FLAG_GLOBAL_ALPHA	0x0010	Use a global alpha value to blend the framebuffer with video images. The blend function is: output = (framebuffer pixel * alpha + video pixel * (255 - alpha)) / 255. The alpha value is determined by the `global_alpha` field of struct *v4l2_window* and negotiated with the *VIDIOC_S_FMT* ioctl, see *Video Overlay Interface* and *Video Output Overlay Interface*.
V4L2_FBUF_FLAG_LOCAL_INV_ALPH	0x0020	Like `V4L2_FBUF_FLAG_LOCAL_ALPHA`, use the alpha channel of the framebuffer to clip or blend framebuffer pixels with video images, but with an inverted alpha value. The blend function is: output = framebuffer pixel * (1 - alpha) + video pixel * alpha. The actual alpha depth depends on the framebuffer pixel format.

continues on next page

Table 176 – continued from previous page

V4L2_FBUF_FLAG_SRC_CHROMAKEY	0x0040	Use source chroma-keying. The source chroma-key color is determined by the chromakey field of struct *v4l2_window* and negotiated with the *VIDIOC_S_FMT* ioctl, see *Video Overlay Interface* and *Video Output Overlay Interface*. Both chroma-keying are mutual exclusive to each other, so same chromakey field of struct *v4l2_window* is being used.

Return Value

On success 0 is returned, on error -1 and the errno variable is set appropriately. The generic error codes are described at the *Generic Error Codes* chapter.

EPERM
: *VIDIOC_S_FBUF* can only be called by a privileged user to negotiate the parameters for a destructive overlay.

EINVAL
: The *VIDIOC_S_FBUF* parameters are unsuitable.

ioctl VIDIOC_G_FMT, VIDIOC_S_FMT, VIDIOC_TRY_FMT

Name

VIDIOC_G_FMT - VIDIOC_S_FMT - VIDIOC_TRY_FMT - Get or set the data format, try a format

Synopsis

VIDIOC_G_FMT

```
int ioctl(int fd, VIDIOC_G_FMT, struct v4l2_format *argp)
```

VIDIOC_S_FMT

```
int ioctl(int fd, VIDIOC_S_FMT, struct v4l2_format *argp)
```

VIDIOC_TRY_FMT

```
int ioctl(int fd, VIDIOC_TRY_FMT, struct v4l2_format *argp)
```

Arguments

fd
File descriptor returned by *open()*.

argp
Pointer to struct *v4l2_format*.

Description

These ioctls are used to negotiate the format of data (typically image format) exchanged between driver and application.

To query the current parameters applications set the `type` field of a struct *v4l2_format* to the respective buffer (stream) type. For example video capture devices use `V4L2_BUF_TYPE_VIDEO_CAPTURE` or `V4L2_BUF_TYPE_VIDEO_CAPTURE_MPLANE`. When the application calls the *VIDIOC_G_FMT* ioctl with a pointer to this structure the driver fills the respective member of the `fmt` union. In case of video capture devices that is either the struct *v4l2_pix_format* `pix` or the struct *v4l2_pix_format_mplane* `pix_mp` member. When the requested buffer type is not supported drivers return an `EINVAL` error code.

To change the current format parameters applications initialize the `type` field and all fields of the respective `fmt` union member. For details see the documentation of the various devices types in *Interfaces*. Good practice is to query the current parameters first, and to modify only those parameters not suitable for the application. When the application calls the *VIDIOC_S_FMT* ioctl with a pointer to a struct *v4l2_format* structure the driver checks and adjusts the parameters against hardware abilities. Drivers should not return an error code unless the `type` field is invalid, this is a mechanism to fathom device capabilities and to approach parameters acceptable for both the application and driver. On success the driver may program the hardware, allocate resources and generally prepare for data exchange. Finally the *VIDIOC_S_FMT* ioctl returns the current format parameters as *VIDIOC_G_FMT* does. Very simple, inflexible devices may even ignore all input and always return the default parameters. However all V4L2 devices exchanging data with the application must implement the *VIDIOC_G_FMT* and *VIDIOC_S_FMT* ioctl. When the requested buffer type is not supported drivers return an EINVAL error code on a *VIDIOC_S_FMT* attempt. When I/O is already in progress or the resource is not available for other reasons drivers return the `EBUSY` error code.

The *VIDIOC_TRY_FMT* ioctl is equivalent to *VIDIOC_S_FMT* with one exception: it does not change driver state. It can also be called at any time, never returning `EBUSY`. This function is provided to negotiate parameters, to learn about hardware limitations, without disabling I/O or possibly time consuming hardware preparations. Although strongly recommended drivers are not required to implement this ioctl.

The format as returned by *VIDIOC_TRY_FMT* must be identical to what *VIDIOC_S_FMT* returns for the same input or output.

type **v4l2_format**

Table 177: struct v4l2_format

__u32	type	Type of the data stream, see *v4l2_buf_type*.
union {	fmt	
struct *v4l2_p*	pix	Definition of an image format, see *Image Formats*, used by video capture and output devices.
struct *v4l2_p*	pix_mp	Definition of an image format, see *Image Formats*, used by video capture and output devices that support the *multi-planar version of the API*.
struct *v4l2_w*	win	Definition of an overlaid image, see *Video Overlay Interface*, used by video overlay devices.
struct *v4l2_v*	vbi	Raw VBI capture or output parameters. This is discussed in more detail in *Raw VBI Data Interface*. Used by raw VBI capture and output devices.
struct *v4l2_s*	sliced	Sliced VBI capture or output parameters. See *Sliced VBI Data Interface* for details. Used by sliced VBI capture and output devices.
struct *v4l2_s*	sdr	Definition of a data format, see *Image For-*

Return Value

On success 0 is returned, on error -1 and the `errno` variable is set appropriately. The generic error codes are described at the *Generic Error Codes* chapter.

EINVAL
: The struct *v4l2_format* `type` field is invalid or the requested buffer type not supported.

EBUSY
: The device is busy and cannot change the format. This could be because or the device is streaming or buffers are allocated or queued to the driver. Relevant for *VIDIOC_S_FMT* only.

ioctl VIDIOC_G_FREQUENCY, VIDIOC_S_FREQUENCY

Name

VIDIOC_G_FREQUENCY - VIDIOC_S_FREQUENCY - Get or set tuner or modulator radio frequency

Synopsis

VIDIOC_G_FREQUENCY

`int ioctl(int fd, VIDIOC_G_FREQUENCY, struct v4l2_frequency *argp)`

VIDIOC_S_FREQUENCY

`int ioctl(int fd, VIDIOC_S_FREQUENCY, const struct v4l2_frequency *argp)`

Arguments

fd
: File descriptor returned by *open()*.

argp
: Pointer to struct *v4l2_frequency*.

Description

To get the current tuner or modulator radio frequency applications set the `tuner` field of a struct *v4l2_frequency* to the respective tuner or modulator number (only input devices have tuners, only output devices have modulators), zero out the `reserved` array and call the *VIDIOC_G_FREQUENCY* ioctl with a pointer to this structure. The driver stores the current frequency in the `frequency` field.

To change the current tuner or modulator radio frequency applications initialize the `tuner`, `type` and `frequency` fields, and the `reserved` array of a struct *v4l2_frequency* and call the *VIDIOC_S_FREQUENCY* ioctl with a pointer to this structure. When the requested frequency is not possible the driver assumes the closest possible value. However *VIDIOC_S_FREQUENCY* is a write-only ioctl, it does not return the actual new frequency.

type **v4l2_frequency**

Table 178: struct v4l2_frequency

__u32	tuner	The tuner or modulator index number. This is the same value as in the struct *v4l2_input* tuner field and the struct *v4l2_tuner* index field, or the struct *v4l2_output* modulator field and the struct *v4l2_modulator* index field.
__u32	type	The tuner type. This is the same value as in the struct *v4l2_tuner* type field. The type must be set to V4L2_TUNER_RADIO for /dev/radioX device nodes, and to V4L2_TUNER_ANALOG_TV for all others. Set this field to V4L2_TUNER_RADIO for modulators (currently only radio modulators are supported). See *v4l2_tuner_type*
__u32	frequency	Tuning frequency in units of 62.5 kHz, or if the struct *v4l2_tuner* or struct *v4l2_modulator* capability flag V4L2_TUNER_CAP_LOW is set, in units of 62.5 Hz. A 1 Hz unit is used when the capability flag V4L2_TUNER_CAP_1HZ is set.
__u32	reserved[8]	Reserved for future extensions. Drivers and applications must set the array to zero.

Return Value

On success 0 is returned, on error -1 and the errno variable is set appropriately. The generic error codes are described at the *Generic Error Codes* chapter.

EINVAL
The tuner index is out of bounds or the value in the type field is wrong.

EBUSY
A hardware seek is in progress.

ioctl VIDIOC_G_INPUT, VIDIOC_S_INPUT

Name

VIDIOC_G_INPUT - VIDIOC_S_INPUT - Query or select the current video input

Synopsis

VIDIOC_G_INPUT

`int ioctl(int fd, VIDIOC_G_INPUT, int *argp)`

VIDIOC_S_INPUT

`int ioctl(int fd, VIDIOC_S_INPUT, int *argp)`

Arguments

fd
File descriptor returned by *open()*.

argp
Pointer an integer with input index.

Description

To query the current video input applications call the *VIDIOC_G_INPUT* ioctl with a pointer to an integer where the driver stores the number of the input, as in the struct *v4l2_input* `index` field. This ioctl will fail only when there are no video inputs, returning `EINVAL`.

To select a video input applications store the number of the desired input in an integer and call the *VIDIOC_S_INPUT* ioctl with a pointer to this integer. Side effects are possible. For example inputs may support different video standards, so the driver may implicitly switch the current standard. Because of these possible side effects applications must select an input before querying or negotiating any other parameters.

Information about video inputs is available using the *ioctl VIDIOC_ENUMINPUT* ioctl.

Return Value

On success 0 is returned, on error -1 and the `errno` variable is set appropriately. The generic error codes are described at the *Generic Error Codes* chapter.

EINVAL
The number of the video input is out of bounds.

ioctl VIDIOC_G_JPEGCOMP, VIDIOC_S_JPEGCOMP

Name

VIDIOC_G_JPEGCOMP - VIDIOC_S_JPEGCOMP

Synopsis

VIDIOC_G_JPEGCOMP

`int ioctl(int fd, VIDIOC_G_JPEGCOMP, v4l2_jpegcompression *argp)`

VIDIOC_S_JPEGCOMP

`int ioctl(int fd, VIDIOC_S_JPEGCOMP, const v4l2_jpegcompression *argp)`

Arguments

fd
: File descriptor returned by *open()*.

argp
: Pointer to struct *v4l2_jpegcompression*.

Description

These ioctls are **deprecated**. New drivers and applications should use *JPEG class controls* for image quality and JPEG markers control.

[to do]

Ronald Bultje elaborates:

APP is some application-specific information. The application can set it itself, and it'll be stored in the JPEG-encoded fields (eg; interlacing information for in an AVI or so). COM is the same, but it's comments, like 'encoded by me' or so.

jpeg_markers describes whether the huffman tables, quantization tables and the restart interval information (all JPEG-specific stuff) should be stored in the JPEG-encoded fields. These define how the JPEG field is encoded. If you omit them, applications assume you've used standard encoding. You usually do want to add them.

type **v4l2_jpegcompression**

Table 179: struct v4l2_jpegcompression

int	quality	Deprecated. If *V4L2_CID_JPEG_COMPRESSION_QUALITY* control is exposed by a driver applications should use it instead and ignore this field.
int	APPn	
int	APP_len	
char	APP_data[60]	
int	COM_len	
char	COM_data[60]	
__u32	jpeg_markers	See *JPEG Markers Flags*. Deprecated. If *V4L2_CID_JPEG_ACTIVE_MARKER* control is exposed by a driver applications should use it instead and ignore this field.

Table 180: JPEG Markers Flags

V4L2_JPEG_MARKER_DHT	(1<<3)	Define Huffman Tables
V4L2_JPEG_MARKER_DQT	(1<<4)	Define Quantization Tables
V4L2_JPEG_MARKER_DRI	(1<<5)	Define Restart Interval
V4L2_JPEG_MARKER_COM	(1<<6)	Comment segment
V4L2_JPEG_MARKER_APP	(1<<7)	App segment, driver will always use APP0

Return Value

On success 0 is returned, on error -1 and the `errno` variable is set appropriately. The generic error codes are described at the *Generic Error Codes* chapter.

ioctl VIDIOC_G_MODULATOR, VIDIOC_S_MODULATOR

Name

VIDIOC_G_MODULATOR - VIDIOC_S_MODULATOR - Get or set modulator attributes

Synopsis

VIDIOC_G_MODULATOR

`int ioctl(int fd, VIDIOC_G_MODULATOR, struct v4l2_modulator *argp)`

VIDIOC_S_MODULATOR

`int ioctl(int fd, VIDIOC_S_MODULATOR, const struct v4l2_modulator *argp)`

Arguments

fd
File descriptor returned by *open()*.

argp
Pointer to struct *v4l2_modulator*.

Description

To query the attributes of a modulator applications initialize the `index` field and zero out the `reserved` array of a struct *v4l2_modulator* and call the *VIDIOC_G_MODULATOR* ioctl with a pointer to this structure. Drivers fill the rest of the structure or return an `EINVAL` error code when the index is out of bounds. To enumerate all modulators applications shall begin at index zero, incrementing by one until the driver returns EINVAL.

Modulators have two writable properties, an audio modulation set and the radio frequency. To change the modulated audio subprograms, applications initialize the `index` and `txsubchans` fields and the `reserved` array and call the *VIDIOC_S_MODULATOR* ioctl. Drivers may choose a different audio modulation if the request cannot be satisfied. However this is a write-only ioctl, it does not return the actual audio modulation selected.

SDR specific modulator types are `V4L2_TUNER_SDR` and `V4L2_TUNER_RF`. For SDR devices `txsubchans` field must be initialized to zero. The term 'modulator' means SDR transmitter in this context.

To change the radio frequency the *VIDIOC_S_FREQUENCY* ioctl is available.

type **v4l2_modulator**

Table 181: struct v4l2_modulator

__u32	index	Identifies the modulator, set by the application.
__u8	name[32]	Name of the modulator, a NUL-terminated ASCII string. This information is intended for the user.
__u32	capability	Modulator capability flags. No flags are defined for this field, the tuner flags in struct *v4l2_tuner* are used accordingly. The audio flags indicate the ability to encode audio subprograms. They will *not* change for example with the current video standard.
__u32	rangelow	The lowest tunable frequency in units of 62.5 KHz, or if the capability flag V4L2_TUNER_CAP_LOW is set, in units of 62.5 Hz, or if the capability flag V4L2_TUNER_CAP_1HZ is set, in units of 1 Hz.
__u32	rangehigh	The highest tunable frequency in units of 62.5 KHz, or if the capability flag V4L2_TUNER_CAP_LOW is set, in units of 62.5 Hz, or if the capability flag V4L2_TUNER_CAP_1HZ is set, in units of 1 Hz.
__u32	txsubchans	With this field applications can determine how audio subcarriers shall be modulated. It contains a set of flags as defined in *Modulator Audio Transmission Flags*. **Note:** The tuner rxsubchans flags are reused, but the semantics are different. Video output devices are assumed to have an analog or PCM audio input with 1-3 channels. The txsubchans flags select one or more channels for modulation, together with some audio subprogram indicator, for example, a stereo pilot tone.
__u32	type	Type of the modulator, see *v4l2_tuner_type*.
__u32	reserved[3]	Reserved for future extensions. Drivers and applications must set the array to zero.

Table 182: Modulator Audio Transmission Flags

V4L2_TUNER_SUB_MONO	0x0001	Modulate channel 1 as mono audio, when the input has more channels, a down-mix of channel 1 and 2. This flag does not combine with V4L2_TUNER_SUB_STEREO or V4L2_TUNER_SUB_LANG1.
V4L2_TUNER_SUB_STEREO	0x0002	Modulate channel 1 and 2 as left and right channel of a stereo audio signal. When the input has only one channel or two channels and V4L2_TUNER_SUB_SAP is also set, channel 1 is encoded as left and right channel. This flag does not combine with V4L2_TUNER_SUB_MONO or V4L2_TUNER_SUB_LANG1. When the driver does not support stereo audio it shall fall back to mono.
V4L2_TUNER_SUB_LANG1	0x0008	Modulate channel 1 and 2 as primary and secondary language of a bilingual audio signal. When the input has only one channel it is used for both languages. It is not possible to encode the primary or secondary language only. This flag does not combine with V4L2_TUNER_SUB_MONO, V4L2_TUNER_SUB_STEREO or V4L2_TUNER_SUB_SAP. If the hardware does not support the respective audio matrix, or the current video standard does not permit bilingual audio the *VIDIOC_S_MODULATOR* ioctl shall return an EINVAL error code and the driver shall fall back to mono or stereo mode.
V4L2_TUNER_SUB_LANG2	0x0004	Same effect as V4L2_TUNER_SUB_SAP.
V4L2_TUNER_SUB_SAP	0x0004	When combined with V4L2_TUNER_SUB_MONO the first channel is encoded as mono audio, the last channel as Second Audio Program. When the input has only one channel it is used for both audio tracks. When the input has three channels the mono track is a down-mix of channel 1 and 2. When combined with V4L2_TUNER_SUB_STEREO channel 1 and 2 are encoded as left and right stereo audio, channel 3 as Second Audio Program. When the input has only two channels, the first is encoded as left and right channel and the second as SAP. When the input has only one channel it is used for all audio tracks. It is not possible to encode a Second Audio Program only. This flag must combine with V4L2_TUNER_SUB_MONO or V4L2_TUNER_SUB_STEREO. If the hardware does not support the respective audio matrix, or the current video standard does not permit SAP the *VIDIOC_S_MODULATOR* ioctl shall return an EINVAL error code and driver shall fall back to mono or stereo mode.
V4L2_TUNER_SUB_RDS	0x0010	Enable the RDS encoder for a radio FM trans-

Return Value

On success 0 is returned, on error -1 and the `errno` variable is set appropriately. The generic error codes are described at the *Generic Error Codes* chapter.

EINVAL
The struct *v4l2_modulator* index is out of bounds.

ioctl VIDIOC_G_OUTPUT, VIDIOC_S_OUTPUT

Name

VIDIOC_G_OUTPUT - VIDIOC_S_OUTPUT - Query or select the current video output

Synopsis

VIDIOC_G_OUTPUT

`int ioctl(int fd, VIDIOC_G_OUTPUT, int *argp)`

VIDIOC_S_OUTPUT

`int ioctl(int fd, VIDIOC_S_OUTPUT, int *argp)`

Arguments

fd
File descriptor returned by *open()*.

argp
Pointer to an integer with output index.

Description

To query the current video output applications call the *VIDIOC_G_OUTPUT* ioctl with a pointer to an integer where the driver stores the number of the output, as in the struct *v4l2_output* `index` field. This ioctl will fail only when there are no video outputs, returning the `EINVAL` error code.

To select a video output applications store the number of the desired output in an integer and call the *VIDIOC_S_OUTPUT* ioctl with a pointer to this integer. Side effects are possible. For example outputs may support different video standards, so the driver may implicitly switch the current standard. standard. Because of these possible side effects applications must select an output before querying or negotiating any other parameters.

Information about video outputs is available using the *ioctl VIDIOC_ENUMOUTPUT* ioctl.

Return Value

On success 0 is returned, on error -1 and the `errno` variable is set appropriately. The generic error codes are described at the *Generic Error Codes* chapter.

EINVAL
: The number of the video output is out of bounds, or there are no video outputs at all.

ioctl VIDIOC_G_PARM, VIDIOC_S_PARM

Name

VIDIOC_G_PARM - VIDIOC_S_PARM - Get or set streaming parameters

Synopsis

`VIDIOC_G_PARM`

`int ioctl(int fd, VIDIOC_G_PARM, v4l2_streamparm *argp)`

`VIDIOC_S_PARM`

`int ioctl(int fd, VIDIOC_S_PARM, v4l2_streamparm *argp)`

Arguments

`fd`
: File descriptor returned by *`open()`*.

`argp`
: Pointer to struct *`v4l2_streamparm`*.

Description

Applications can request a different frame interval. The capture or output device will be reconfigured to support the requested frame interval if possible. Optionally drivers may choose to skip or repeat frames to achieve the requested frame interval.

For stateful encoders (see *Memory-to-Memory Stateful Video Encoder Interface*) this represents the frame interval that is typically embedded in the encoded video stream.

Changing the frame interval shall never change the format. Changing the format, on the other hand, may change the frame interval.

Further these ioctls can be used to determine the number of buffers used internally by a driver in read/write mode. For implications see the section discussing the *`read()`* function.

To get and set the streaming parameters applications call the *VIDIOC_G_PARM* and *VIDIOC_S_PARM* ioctl, respectively. They take a pointer to a struct *v4l2_streamparm* which contains a union holding separate parameters for input and output devices.

type **v4l2_streamparm**

Table 183: struct v4l2_streamparm

__u32	type	The buffer (stream) type, same as struct *v4l2_format* type, set by the application. See *v4l2_buf_type*.
union {	parm	
struct *v4l2_captureparm*	capture	Parameters for capture devices, used when type is V4L2_BUF_TYPE_VIDEO_CAPTURE or V4L2_BUF_TYPE_VIDEO_CAPTURE_MPLANE.
struct *v4l2_outputparm*	output	Parameters for output devices, used when type is V4L2_BUF_TYPE_VIDEO_OUTPUT or V4L2_BUF_TYPE_VIDEO_OUTPUT_MPLANE.
__u8	raw_data[200]	A place holder for future extensions.
}		

type **v4l2_captureparm**

Table 184: struct v4l2_captureparm

__u32	capability	See *Streaming Parameters Capabilities*.
__u32	capturemode	Set by drivers and applications, see *Capture Parameters Flags*.
struct *v4l2_fract*	timeperframe	This is the desired period between successive frames captured by the driver, in seconds.
This will configure the speed at which the video source (e.g. a sensor) generates video frames. If the speed is fixed, then the driver may choose to skip or repeat frames in order to achieve the requested frame rate. For stateful encoders (see *Memory-to-Memory Stateful Video Encoder Interface*) this represents the frame interval that is typically embedded in the encoded video stream. Applications store here the desired frame period, drivers return the actual frame period. Changing the video standard (also implicitly by switching the video input) may reset this parameter to the nominal frame period. To reset manually applications can just set this field to zero. Drivers support this function only when they set the V4L2_CAP_TIMEPERFRAME flag in the capability field.		
__u32	extendedmode	Custom (driver specific) streaming parameters. When unused, applications and drivers must set this field to zero. Applications using this field should check the driver name and version, see *Querying Capabilities*.
__u32	readbuffers	Applications set this field to the desired number of buffers used internally by the driver in *read()* mode. Drivers return the actual number of buffers. When an application requests zero buffers, drivers should just return the current setting rather than the minimum or an error code. For details see *Read/Write*.
__u32	reserved[4]	Reserved for future extensions. Drivers and applications must set the array to zero.

type **v4l2_outputparm**

Table 185: struct v4l2_outputparm

<table>
<tr><td>__u32</td><td>capability</td><td>See Streaming Parameters Capabilities.</td></tr>
<tr><td>__u32</td><td>outputmode</td><td>Set by drivers and applications, see Capture Parameters Flags.</td></tr>
<tr><td>struct v4l2_fract</td><td>timeperframe</td><td>This is the desired period between successive frames output by the driver, in seconds.</td></tr>
<tr><td colspan="3">The field is intended to repeat frames on the driver side in write() mode (in streaming mode timestamps can be used to throttle the output), saving I/O bandwidth.
For stateful encoders (see Memory-to-Memory Stateful Video Encoder Interface) this represents the frame interval that is typically embedded in the encoded video stream and it provides a hint to the encoder of the speed at which raw frames are queued up to the encoder.
Applications store here the desired frame period, drivers return the actual frame period.
Changing the video standard (also implicitly by switching the video output) may reset this parameter to the nominal frame period. To reset manually applications can just set this field to zero.
Drivers support this function only when they set the V4L2_CAP_TIMEPERFRAME flag in the capability field.</td></tr>
<tr><td>__u32</td><td>extendedmode</td><td>Custom (driver specific) streaming parameters. When unused, applications and drivers must set this field to zero. Applications using this field should check the driver name and version, see Querying Capabilities.</td></tr>
<tr><td>__u32</td><td>writebuffers</td><td>Applications set this field to the desired number of buffers used internally by the driver in write() mode. Drivers return the actual number of buffers. When an application requests zero buffers, drivers should just return the current setting rather than the minimum or an error code. For details see Read/Write.</td></tr>
<tr><td>__u32</td><td>reserved[4]</td><td>Reserved for future extensions. Drivers and applications must set the array to zero.</td></tr>
</table>

Table 186: Streaming Parameters Capabilities

V4L2_CAP_TIMEPERFRAME	0x1000	The frame period can be modified by setting the timeperframe field.

Table 187: Capture Parameters Flags

V4L2_MODE_HIGHQUALITY	0x0001	High quality imaging mode. High quality mode is intended for still imaging applications. The idea is to get the best possible image quality that the hardware can deliver. It is not defined how the driver writer may achieve that; it will depend on the hardware and the ingenuity of the driver writer. High quality mode is a different mode from the regular motion video capture modes. In high quality mode: • The driver may be able to capture higher resolutions than for motion capture. • The driver may support fewer pixel formats than motion capture (eg; true color). • The driver may capture and arithmetically combine multiple successive fields or frames to remove color edge artifacts and reduce the noise in the video data. • The driver may capture images in slices like a scanner in order to handle larger format images than would otherwise be possible. • An image capture operation may be significantly slower than motion capture. • Moving objects in the image might have excessive motion blur. • Capture might only work through the *read()* call.

Return Value

On success 0 is returned, on error -1 and the `errno` variable is set appropriately. The generic error codes are described at the *Generic Error Codes* chapter.

ioctl VIDIOC_G_PRIORITY, VIDIOC_S_PRIORITY

Name

VIDIOC_G_PRIORITY - VIDIOC_S_PRIORITY - Query or request the access priority associated with a file descriptor

Synopsis

VIDIOC_G_PRIORITY

`int ioctl(int fd, VIDIOC_G_PRIORITY, enum v4l2_priority *argp)`

VIDIOC_S_PRIORITY

`int ioctl(int fd, VIDIOC_S_PRIORITY, const enum v4l2_priority *argp)`

Arguments

fd
File descriptor returned by *open()*.

argp
Pointer to an enum *v4l2_priority* type.

Description

To query the current access priority applications call the *VIDIOC_G_PRIORITY* ioctl with a pointer to an enum v4l2_priority variable where the driver stores the current priority.

To request an access priority applications store the desired priority in an enum v4l2_priority variable and call *VIDIOC_S_PRIORITY* ioctl with a pointer to this variable.

type **v4l2_priority**

Table 188: enum v4l2_priority

V4L2_PRIORITY_UNSET	0	
V4L2_PRIORITY_BACKGROUND	1	Lowest priority, usually applications running in background, for example monitoring VBI transmissions. A proxy application running in user space will be necessary if multiple applications want to read from a device at this priority.
V4L2_PRIORITY_INTERACTIVE	2	
V4L2_PRIORITY_DEFAULT	2	Medium priority, usually applications started and interactively controlled by the user. For example TV viewers, Teletext browsers, or just "panel" applications to change the channel or video controls. This is the default priority unless an application requests another.
V4L2_PRIORITY_RECORD	3	Highest priority. Only one file descriptor can have this priority, it blocks any other fd from changing device properties. Usually applications which must not be interrupted, like video recording.

Return Value

On success 0 is returned, on error -1 and the `errno` variable is set appropriately. The generic error codes are described at the *Generic Error Codes* chapter.

EINVAL
The requested priority value is invalid.

EBUSY
Another application already requested higher priority.

ioctl VIDIOC_G_SELECTION, VIDIOC_S_SELECTION

Name

VIDIOC_G_SELECTION - VIDIOC_S_SELECTION - Get or set one of the selection rectangles

Synopsis

VIDIOC_G_SELECTION

`int ioctl(int fd, VIDIOC_G_SELECTION, struct v4l2_selection *argp)`

VIDIOC_S_SELECTION

`int ioctl(int fd, VIDIOC_S_SELECTION, struct v4l2_selection *argp)`

Arguments

fd
File descriptor returned by *open()*.

argp
Pointer to struct *v4l2_selection*.

Description

The ioctls are used to query and configure selection rectangles.

To query the cropping (composing) rectangle set struct *v4l2_selection* `type` field to the respective buffer type. The next step is setting the value of struct *v4l2_selection* `target` field to `V4L2_SEL_TGT_CROP` (`V4L2_SEL_TGT_COMPOSE`). Please refer to table *Common selection definitions* or *Cropping, composing and scaling – the SELECTION API* for additional targets. The `flags` and `reserved` fields of struct *v4l2_selection* are ignored and they must be filled with zeros. The driver fills the rest of the structure or returns EINVAL error code if incorrect buffer type or target was used. If cropping (composing) is not supported then the active rectangle is not mutable and it is always equal to the bounds rectangle.

Finally, the struct *v4l2_rect* `r` rectangle is filled with the current cropping (composing) coordinates. The coordinates are expressed in driver-dependent units. The only exception are rectangles for images in raw formats, whose coordinates are always expressed in pixels.

To change the cropping (composing) rectangle set the struct *v4l2_selection* `type` field to the respective buffer type. The next step is setting the value of struct *v4l2_selection* `target` to `V4L2_SEL_TGT_CROP` (`V4L2_SEL_TGT_COMPOSE`). Please refer to table *Common selection definitions* or *Cropping, composing and scaling – the SELECTION API* for additional targets. The struct *v4l2_rect* `r` rectangle need to be set to the desired active area. Field struct *v4l2_selection* `reserved` is ignored and must be filled with zeros. The driver may adjust coordinates of the requested rectangle. An application may introduce constraints to control rounding behaviour. The struct *v4l2_selection* `flags` field must be set to one of the following:

- `0` - The driver can adjust the rectangle size freely and shall choose a crop/compose rectangle as close as possible to the requested one.
- `V4L2_SEL_FLAG_GE` - The driver is not allowed to shrink the rectangle. The original rectangle must lay inside the adjusted one.
- `V4L2_SEL_FLAG_LE` - The driver is not allowed to enlarge the rectangle. The adjusted rectangle must lay inside the original one.
- `V4L2_SEL_FLAG_GE | V4L2_SEL_FLAG_LE` - The driver must choose the size exactly the same as in the requested rectangle.

Please refer to *Size adjustments with constraint flags.*.

The driver may have to adjusts the requested dimensions against hardware limits and other parts as the pipeline, i.e. the bounds given by the capture/output window or TV display. The closest possible values of horizontal and vertical offset and sizes are chosen according to following priority:

1. Satisfy constraints from struct *v4l2_selection* `flags`.
2. Adjust width, height, left, and top to hardware limits and alignments.
3. Keep center of adjusted rectangle as close as possible to the original one.
4. Keep width and height as close as possible to original ones.
5. Keep horizontal and vertical offset as close as possible to original ones.

On success the struct *v4l2_rect* `r` field contains the adjusted rectangle. When the parameters are unsuitable the application may modify the cropping (composing) or image parameters and repeat the cycle until satisfactory parameters have been negotiated. If constraints flags have to be violated at then `ERANGE` is returned. The error indicates that *there exist no rectangle* that satisfies the constraints.

Selection targets and flags are documented in *Common selection definitions*.

type **`v4l2_selection`**

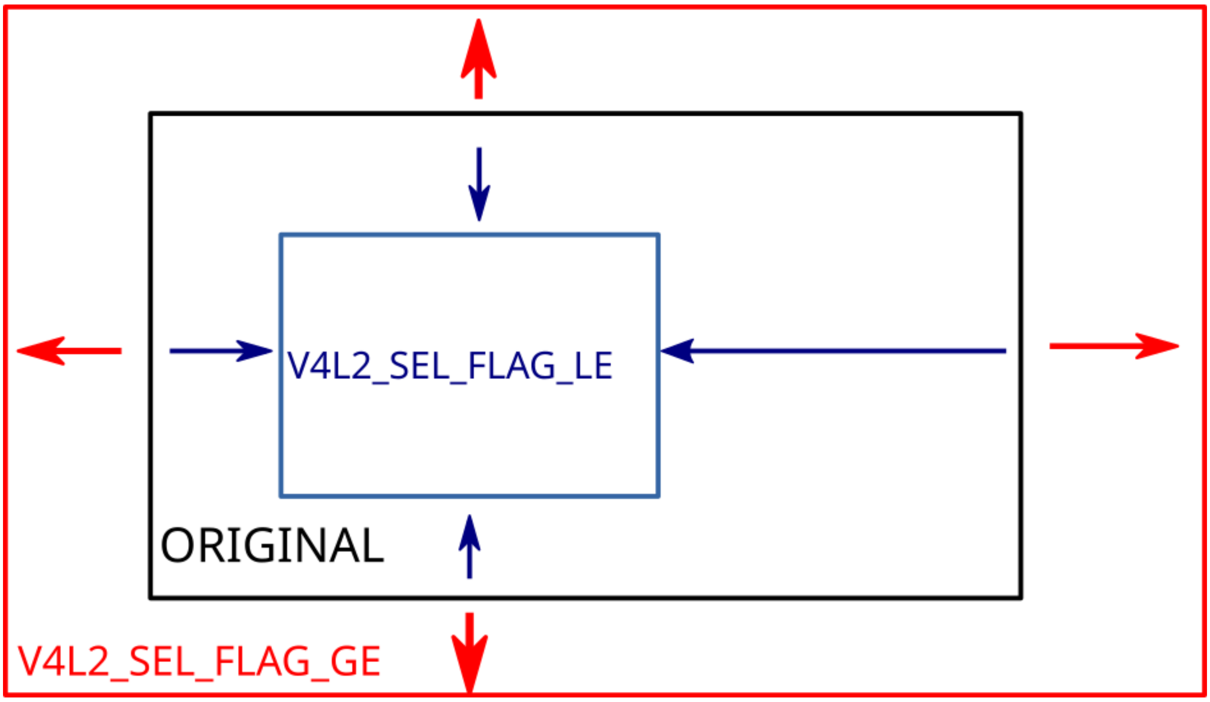

Fig. 18: Size adjustments with constraint flags.
Behaviour of rectangle adjustment for different constraint flags.

Table 189: struct v4l2_selection

__u32	`type`	Type of the buffer (from enum *v4l2_buf_type*).
__u32	`target`	Used to select between *cropping and composing rectangles*.
__u32	`flags`	Flags controlling the selection rectangle adjustments, refer to *selection flags*.
struct *v4l2_rect*	`r`	The selection rectangle.
__u32	`reserved[9]`	Reserved fields for future use. Drivers and applications must zero this array.

Note: Unfortunately in the case of multiplanar buffer types (`V4L2_BUF_TYPE_VIDEO_CAPTURE_MPLANE` and `V4L2_BUF_TYPE_VIDEO_OUTPUT_MPLANE`) this API was messed up with regards to how the *v4l2_selection* type field should be filled in. Some drivers only accepted the `_MPLANE` buffer type while other drivers only accepted a non-multiplanar buffer type (i.e. without the `_MPLANE` at the end).

Starting with kernel 4.13 both variations are allowed.

Return Value

On success 0 is returned, on error -1 and the `errno` variable is set appropriately. The generic error codes are described at the *Generic Error Codes* chapter.

EINVAL
Given buffer type `type` or the selection target `target` is not supported, or the `flags` argument is not valid.

ERANGE
It is not possible to adjust struct *v4l2_rect* `r` rectangle to satisfy all constraints given in the `flags` argument.

ENODATA
Selection is not supported for this input or output.

EBUSY
It is not possible to apply change of the selection rectangle at the moment. Usually because streaming is in progress.

ioctl VIDIOC_G_SLICED_VBI_CAP

Name

VIDIOC_G_SLICED_VBI_CAP - Query sliced VBI capabilities

Synopsis

VIDIOC_G_SLICED_VBI_CAP

```
int ioctl(int fd, VIDIOC_G_SLICED_VBI_CAP, struct
v4l2_sliced_vbi_cap *argp)
```

Arguments

fd
File descriptor returned by *open()*.

argp
Pointer to struct *v4l2_sliced_vbi_cap*.

Description

To find out which data services are supported by a sliced VBI capture or output device, applications initialize the `type` field of a struct *v4l2_sliced_vbi_cap*, clear the `reserved` array and call the *VIDIOC_G_SLICED_VBI_CAP* ioctl. The driver fills in the remaining fields or returns an `EINVAL` error code if the sliced VBI API is unsupported or `type` is invalid.

Note: The type field was added, and the ioctl changed from read-only to write-read, in Linux 2.6.19.

type **v4l2_sliced_vbi_cap**

Table 190: struct v4l2_sliced_vbi_cap

__u16	service_set	A set of all data services supported by the driver. Equal to the union of all elements of the service_lines array.		
__u16	service_lines[2][24]	Each element of this array contains a set of data services the hardware can look for or insert into a particular scan line. Data services are defined in *Sliced VBI services*. Array indices map to ITU-R line numbers[1] as follows:		
		Element	525 line systems	625 line systems
		service_lines[0][1]	1	1
		service_lines[0][23]	23	23
		service_lines[1][1]	264	314
		service_lines[1][23]	286	336
		The number of VBI lines the hardware can capture or output per frame, or the number of services it can identify on a given line may be limited. For example on PAL line 16 the hardware may be able to look for a VPS or Teletext signal, but not both at the same time. Applications can learn about these limits using the *VIDIOC_S_FMT* ioctl as described in *Sliced VBI Data Interface*.		
		Drivers must set service_lines [0][0] and service_lines[1][0] to zero.		
__u32	type	Type of the data stream, see *v4l2_buf_type*. Should be V4L2_BUF_TYPE_SLICED_VBI_CAPTURE or V4L2_BUF_TYPE_SLICED_VBI_OUTPUT.		
__u32	reserved[3]	This array is reserved for future extensions. Applications and drivers must set it to zero.		

[1] See also *Figure 4.2. ITU-R 525 line numbering (M/NTSC and M/PAL)* and *Figure 4.3. ITU-R 625 line numbering*.

Table 191: Sliced VBI services

Symbol	Value	Reference	Lines, usually	Payload
V4L2_SLICED_TELETEXT_ (Teletext System B)	0x0001	*ETS 300 706*, *ITU BT.653*	PAL/SECAM line 7-22, 320-335 (second field 7-22)	Last 42 of the 45 byte Teletext packet, that is without clock run-in and framing code, lsb first transmitted.
V4L2_SLICED_VPS	0x0400	*ETS 300 231*	PAL line 16	Byte number 3 to 15 according to Figure 9 of ETS 300 231, lsb first transmitted.
V4L2_SLICED_CAPTION_5	0x1000	*CEA 608-E*	NTSC line 21, 284 (second field 21)	Two bytes in transmission order, including parity bit, lsb first transmitted.

V4L2_SLICED_WSS_625	0x4000	*EN 300 294*, *ITU BT.1119*	PAL/SECAM line 23	`Byte 0 1`
				 `msb lsb msb lsb`
				 `Bit 7 6 5 4 3 2 1 0 x x 13 12 11 10 9`

V4L2_SLICED_VBI_525	0x1000	Set of services applicable to 525 line systems.		
V4L2_SLICED_VBI_625	0x4401	Set of services applicable to 625 line systems.		

Return Value

On success 0 is returned, on error -1 and the `errno` variable is set appropriately. The generic error codes are described at the *Generic Error Codes* chapter.

EINVAL
The value in the `type` field is wrong.

ioctl VIDIOC_G_STD, VIDIOC_S_STD, VIDIOC_SUBDEV_G_STD, VIDIOC_SUBDEV_S_STD

Name

VIDIOC_G_STD - VIDIOC_S_STD - VIDIOC_SUBDEV_G_STD - VIDIOC_SUBDEV_S_STD - Query or select the video standard of the current input

Synopsis

VIDIOC_G_STD

`int ioctl(int fd, VIDIOC_G_STD, v4l2_std_id *argp)`

VIDIOC_S_STD

`int ioctl(int fd, VIDIOC_S_STD, const v4l2_std_id *argp)`

VIDIOC_SUBDEV_G_STD

`int ioctl(int fd, VIDIOC_SUBDEV_G_STD, v4l2_std_id *argp)`

VIDIOC_SUBDEV_S_STD

`int ioctl(int fd, VIDIOC_SUBDEV_S_STD, const v4l2_std_id *argp)`

Arguments

fd
: File descriptor returned by *open()*.

argp
: Pointer to v4l2_std_id.

Description

To query and select the current video standard applications use the *VIDIOC_G_STD* and *VIDIOC_S_STD* ioctls which take a pointer to a *v4l2_std_id* type as argument. *VIDIOC_G_STD* can return a single flag or a set of flags as in struct *v4l2_standard* field id. The flags must be unambiguous such that they appear in only one enumerated struct *v4l2_standard* structure.

VIDIOC_S_STD accepts one or more flags, being a write-only ioctl it does not return the actual new standard as *VIDIOC_G_STD* does. When no flags are given or the current input does not support the requested standard the driver returns an EINVAL error code. When the standard set is ambiguous drivers may return EINVAL or choose any of the requested standards. If the current input or output does not support standard video timings (e.g. if *ioctl VIDIOC_ENUMINPUT* does not set the V4L2_IN_CAP_STD flag), then ENODATA error code is returned.

Calling VIDIOC_SUBDEV_S_STD on a subdev device node that has been registered in read-only mode is not allowed. An error is returned and the errno variable is set to -EPERM.

Return Value

On success 0 is returned, on error -1 and the errno variable is set appropriately. The generic error codes are described at the *Generic Error Codes* chapter.

EINVAL
: The *VIDIOC_S_STD* parameter was unsuitable.

ENODATA
: Standard video timings are not supported for this input or output.

EPERM
: VIDIOC_SUBDEV_S_STD has been called on a read-only subdevice.

ioctl VIDIOC_G_TUNER, VIDIOC_S_TUNER

Name

VIDIOC_G_TUNER - VIDIOC_S_TUNER - Get or set tuner attributes

Synopsis

VIDIOC_G_TUNER

`int ioctl(int fd, VIDIOC_G_TUNER, struct v4l2_tuner *argp)`

VIDIOC_S_TUNER

`int ioctl(int fd, VIDIOC_S_TUNER, const struct v4l2_tuner *argp)`

Arguments

fd
File descriptor returned by *open()*.

argp
Pointer to struct *v4l2_tuner*.

Description

To query the attributes of a tuner applications initialize the `index` field and zero out the `reserved` array of a struct *v4l2_tuner* and call the `VIDIOC_G_TUNER` ioctl with a pointer to this structure. Drivers fill the rest of the structure or return an `EINVAL` error code when the index is out of bounds. To enumerate all tuners applications shall begin at index zero, incrementing by one until the driver returns `EINVAL`.

Tuners have two writable properties, the audio mode and the radio frequency. To change the audio mode, applications initialize the `index`, `audmode` and `reserved` fields and call the `VIDIOC_S_TUNER` ioctl. This will *not* change the current tuner, which is determined by the current video input. Drivers may choose a different audio mode if the requested mode is invalid or unsupported. Since this is a write-only ioctl, it does not return the actually selected audio mode.

SDR specific tuner types are `V4L2_TUNER_SDR` and `V4L2_TUNER_RF`. For SDR devices `audmode` field must be initialized to zero. The term 'tuner' means SDR receiver in this context.

To change the radio frequency the *VIDIOC_S_FREQUENCY* ioctl is available.

type **v4l2_tuner**

Table 192: struct v4l2_tuner

__u32	index	Identifies the tuner, set by the application.
__u8	name[32]	Name of the tuner, a NUL-terminated ASCII string. This information is intended for the user.
__u32	type	Type of the tuner, see *v4l2_tuner_type*.

continues on next page

Table 192 – continued from previous page

__u32	capability	Tuner capability flags, see *Tuner and Modulator Capability Flags*. Audio flags indicate the ability to decode audio subprograms. They will *not* change, for example with the current video standard. When the structure refers to a radio tuner the V4L2_TUNER_CAP_LANG1, V4L2_TUNER_CAP_LANG2 and V4L2_TUNER_CAP_NORM flags can't be used. If multiple frequency bands are supported, then capability is the union of all capability fields of each struct *v4l2_frequency_band*.	
__u32	rangelow	The lowest tunable frequency in units of 62.5 kHz, or if the capability flag V4L2_TUNER_CAP_LOW is set, in units of 62.5 Hz, or if the capability flag V4L2_TUNER_CAP_1HZ is set, in units of 1 Hz. If multiple frequency bands are supported, then rangelow is the lowest frequency of all the frequency bands.	
__u32	rangehigh	The highest tunable frequency in units of 62.5 kHz, or if the capability flag V4L2_TUNER_CAP_LOW is set, in units of 62.5 Hz, or if the capability flag V4L2_TUNER_CAP_1HZ is set, in units of 1 Hz. If multiple frequency bands are supported, then rangehigh is the highest frequency of all the frequency bands.	
__u32	rxsubchans	Some tuners or audio decoders can determine the received audio subprograms by analyzing audio carriers, pilot tones or other indicators. To pass this information drivers set flags defined in *Tuner Audio Reception Flags* in this field. For example:	
		V4L2_TUNER_SUB_MONO	receiving mono audio
		STEREO | SAP	receiving stereo audio and a secondary audio program
		MONO | STEREO	receiving mono or stereo audio, the hardware cannot distinguish
		LANG1 | LANG2	receiving bilingual audio
		MONO | STEREO | LANG1 | LANG2	receiving mono, stereo or bilingual audio
		When the V4L2_TUNER_CAP_STEREO, _LANG1, _LANG2 or _SAP flag is cleared in the capability field, the corresponding V4L2_TUNER_SUB_ flag must not be set here. This field is valid only if this is the tuner of the current video input, or when the structure refers to a radio tuner.	
__u32	audmode	The selected audio mode, see *Tuner Audio Modes* for valid values. The audio mode does not affect audio subprogram detection, and like a *User Controls* it does not automatically change unless the requested mode is invalid or unsupported. See *Tuner Audio Matrix* for possible results when the selected and received audio programs do not match. Currently this is the only field of struct struct *v4l2_tuner* applications can change.	
__u32	signal	The signal strength if known. Ranging from 0 to 65535. Higher values indicate a better signal.	

continues on next page

Table 192 – continued from previous page

__s32	afc	Automatic frequency control. When the afc value is negative, the frequency is too low, when positive too high.
__u32	reserved[4]	Reserved for future extensions. Drivers and applications must set the array to zero.

type **v4l2_tuner_type**

Table 193: enum v4l2_tuner_type

V4L2_TUNER_RADIO	1	Tuner supports radio
V4L2_TUNER_ANALOG_TV	2	Tuner supports analog TV
V4L2_TUNER_SDR	4	Tuner controls the A/D and/or D/A block of a Software Digital Radio (SDR)
V4L2_TUNER_RF	5	Tuner controls the RF part of a Software Digital Radio (SDR)

Table 194: Tuner and Modulator Capability Flags

V4L2_TUNER_CAP_LOW	0x0001	When set, tuning frequencies are expressed in units of 62.5 Hz instead of 62.5 kHz.
V4L2_TUNER_CAP_NORM	0x0002	This is a multi-standard tuner; the video standard can or must be switched. (B/G PAL tuners for example are typically not considered multi-standard because the video standard is automatically determined from the frequency band.) The set of supported video standards is available from the struct *v4l2_input* pointing to this tuner, see the description of ioctl *ioctl VIDIOC_ENUMINPUT* for details. Only V4L2_TUNER_ANALOG_TV tuners can have this capability.
V4L2_TUNER_CAP_HWSEEK_BOUNDED	0x0004	If set, then this tuner supports the hardware seek functionality where the seek stops when it reaches the end of the frequency range.
V4L2_TUNER_CAP_HWSEEK_WRAP	0x0008	If set, then this tuner supports the hardware seek functionality where the seek wraps around when it reaches the end of the frequency range.
V4L2_TUNER_CAP_STEREO	0x0010	Stereo audio reception is supported.
V4L2_TUNER_CAP_LANG1	0x0040	Reception of the primary language of a bilingual audio program is supported. Bilingual audio is a feature of two-channel systems, transmitting the primary language monaural on the main audio carrier and a secondary language monaural on a second carrier. Only V4L2_TUNER_ANALOG_TV tuners can have this capability.

continues on next page

Table 194 – continued from previous page

V4L2_TUNER_CAP_LANG2	0x0020	Reception of the secondary language of a bilingual audio program is supported. Only V4L2_TUNER_ANALOG_TV tuners can have this capability.
V4L2_TUNER_CAP_SAP	0x0020	Reception of a secondary audio program is supported. This is a feature of the BTSC system which accompanies the NTSC video standard. Two audio carriers are available for mono or stereo transmissions of a primary language, and an independent third carrier for a monaural secondary language. Only V4L2_TUNER_ANALOG_TV tuners can have this capability. **Note:** The V4L2_TUNER_CAP_LANG2 and V4L2_TUNER_CAP_SAP flags are synonyms. V4L2_TUNER_CAP_SAP applies when the tuner supports the V4L2_STD_NTSC_M video standard.
V4L2_TUNER_CAP_RDS	0x0080	RDS capture is supported. This capability is only valid for radio tuners.
V4L2_TUNER_CAP_RDS_BLOCK_IO	0x0100	The RDS data is passed as unparsed RDS blocks.
V4L2_TUNER_CAP_RDS_CONTROLS	0x0200	The RDS data is parsed by the hardware and set via controls.
V4L2_TUNER_CAP_FREQ_BANDS	0x0400	The *ioctl VIDIOC_ENUM_FREQ_BANDS* ioctl can be used to enumerate the available frequency bands.
V4L2_TUNER_CAP_HWSEEK_PROG_LI	0x0800	The range to search when using the hardware seek functionality is programmable, see *ioctl VIDIOC_S_HW_FREQ_SEEK* for details.
V4L2_TUNER_CAP_1HZ	0x1000	When set, tuning frequencies are expressed in units of 1 Hz instead of 62.5 kHz.

Table 195: Tuner Audio Reception Flags

V4L2_TUNER_SUB_MONO	0x0001	The tuner receives a mono audio signal.
V4L2_TUNER_SUB_STEREO	0x0002	The tuner receives a stereo audio signal.
V4L2_TUNER_SUB_LANG1	0x0008	The tuner receives the primary language of a bilingual audio signal. Drivers must clear this flag when the current video standard is V4L2_STD_NTSC_M.
V4L2_TUNER_SUB_LANG2	0x0004	The tuner receives the secondary language of a bilingual audio signal (or a second audio program).
V4L2_TUNER_SUB_SAP	0x0004	The tuner receives a Second Audio Program. **Note:** The V4L2_TUNER_SUB_LANG2 and V4L2_TUNER_SUB_SAP flags are synonyms. The V4L2_TUNER_SUB_SAP flag applies when the current video standard is V4L2_STD_NTSC_M.
V4L2_TUNER_SUB_RDS	0x0010	The tuner receives an RDS channel.

Table 196: Tuner Audio Modes

V4L2_TUNER_MODE_MONO	0	Play mono audio. When the tuner receives a stereo signal this a down-mix of the left and right channel. When the tuner receives a bilingual or SAP signal this mode selects the primary language.
V4L2_TUNER_MODE_STEREO	1	Play stereo audio. When the tuner receives bilingual audio it may play different languages on the left and right channel or the primary language is played on both channels. Playing different languages in this mode is deprecated. New drivers should do this only in MODE_LANG1_LANG2. When the tuner receives no stereo signal or does not support stereo reception the driver shall fall back to MODE_MONO.
V4L2_TUNER_MODE_LANG1	3	Play the primary language, mono or stereo. Only V4L2_TUNER_ANALOG_TV tuners support this mode.
V4L2_TUNER_MODE_LANG2	2	Play the secondary language, mono. When the tuner receives no bilingual audio or SAP, or their reception is not supported the driver shall fall back to mono or stereo mode. Only V4L2_TUNER_ANALOG_TV tuners support this mode.
V4L2_TUNER_MODE_SAP	2	Play the Second Audio Program. When the tuner receives no bilingual audio or SAP, or their reception is not supported the driver shall fall back to mono or stereo mode. Only V4L2_TUNER_ANALOG_TV tuners support this mode. **Note:** The V4L2_TUNER_MODE_LANG2 and V4L2_TUNER_MODE_SAP are synonyms.
V4L2_TUNER_MODE_LANG1_LANG2	4	Play the primary language on the left channel, the secondary language on the right channel. When the tuner receives no bilingual audio or SAP, it shall fall back to MODE_LANG1 or MODE_MONO. Only V4L2_TUNER_ANALOG_TV tuners support this mode.

Table 197: Tuner Audio Matrix

Received V4L2_TUNER_SUB_	Selected V4L2_TUNER_MODE_ MONO	STEREO	LANG1	LANG2 = SAP	LANG1_LANG2[1]
MONO	Mono	Mono/Mono	Mono	Mono	Mono/Mono
MONO | SAP	Mono	Mono/Mono	Mono	SAP	Mono/SAP (preferred) or Mono/Mono
STEREO	L+R	L/R	Stereo L/R (preferred) or Mono L+R	Stereo L/R (preferred) or Mono L+R	L/R (preferred) or L+R/L+R
STEREO | SAP	L+R	L/R	Stereo L/R (preferred) or Mono L+R	SAP	L+R/SAP (preferred) or L/R or L+R/L+R
LANG1 | LANG2	Language 1	Lang1/Lang2 (deprecated[2]) or Lang1/Lang1	Language 1	Language 2	Lang1/Lang2 (preferred) or Lang1/Lang1

Return Value

On success 0 is returned, on error -1 and the `errno` variable is set appropriately. The generic error codes are described at the *Generic Error Codes* chapter.

EINVAL
: The struct *v4l2_tuner* `index` is out of bounds.

ioctl VIDIOC_LOG_STATUS

Name

VIDIOC_LOG_STATUS - Log driver status information

Synopsis

VIDIOC_LOG_STATUS

`int ioctl(int fd, VIDIOC_LOG_STATUS)`

[1] This mode has been added in Linux 2.6.17 and may not be supported by older drivers.

[2] Playback of both languages in `MODE_STEREO` is deprecated. In the future drivers should produce only the primary language in this mode. Applications should request `MODE_LANG1_LANG2` to record both languages or a stereo signal.

Arguments

fd
File descriptor returned by *open()*.

Description

As the video/audio devices become more complicated it becomes harder to debug problems. When this ioctl is called the driver will output the current device status to the kernel log. This is particular useful when dealing with problems like no sound, no video and incorrectly tuned channels. Also many modern devices autodetect video and audio standards and this ioctl will report what the device thinks what the standard is. Mismatches may give an indication where the problem is.

This ioctl is optional and not all drivers support it. It was introduced in Linux 2.6.15.

Return Value

On success 0 is returned, on error -1 and the `errno` variable is set appropriately. The generic error codes are described at the *Generic Error Codes* chapter.

ioctl VIDIOC_OVERLAY

Name

VIDIOC_OVERLAY - Start or stop video overlay

Synopsis

VIDIOC_OVERLAY

`int ioctl(int fd, VIDIOC_OVERLAY, const int *argp)`

Arguments

fd
File descriptor returned by *open()*.

argp
Pointer to an integer.

Description

This ioctl is part of the *video overlay* I/O method. Applications call *ioctl VIDIOC_OVERLAY* to start or stop the overlay. It takes a pointer to an integer which must be set to zero by the application to stop overlay, to one to start.

Drivers do not support *ioctl VIDIOC_STREAMON, VIDIOC_STREAMOFF* or *VIDIOC_STREAMOFF* with `V4L2_BUF_TYPE_VIDEO_OVERLAY`.

Return Value

On success 0 is returned, on error -1 and the `errno` variable is set appropriately. The generic error codes are described at the *Generic Error Codes* chapter.

EINVAL
: The overlay parameters have not been set up. See *Video Overlay Interface* for the necessary steps.

ioctl VIDIOC_PREPARE_BUF

Name

VIDIOC_PREPARE_BUF - Prepare a buffer for I/O

Synopsis

VIDIOC_PREPARE_BUF

`int ioctl(int fd, VIDIOC_PREPARE_BUF, struct v4l2_buffer *argp)`

Arguments

fd
: File descriptor returned by *open()*.

argp
: Pointer to struct *v4l2_buffer*.

Description

Applications can optionally call the *ioctl VIDIOC_PREPARE_BUF* ioctl to pass ownership of the buffer to the driver before actually enqueuing it, using the *VIDIOC_QBUF* ioctl, and to prepare it for future I/O. Such preparations may include cache invalidation or cleaning. Performing them in advance saves time during the actual I/O.

The struct *v4l2_buffer* structure is specified in *Buffers*.

Return Value

On success 0 is returned, on error -1 and the `errno` variable is set appropriately. The generic error codes are described at the *Generic Error Codes* chapter.

EBUSY
File I/O is in progress.

EINVAL
The buffer `type` is not supported, or the `index` is out of bounds, or no buffers have been allocated yet, or the `userptr` or `length` are invalid.

ioctl VIDIOC_QBUF, VIDIOC_DQBUF

Name

VIDIOC_QBUF - VIDIOC_DQBUF - Exchange a buffer with the driver

Synopsis

`VIDIOC_QBUF`

`int ioctl(int fd, VIDIOC_QBUF, struct v4l2_buffer *argp)`

`VIDIOC_DQBUF`

`int ioctl(int fd, VIDIOC_DQBUF, struct v4l2_buffer *argp)`

Arguments

`fd`
File descriptor returned by *`open()`*.

`argp`
Pointer to struct *`v4l2_buffer`*.

Description

Applications call the `VIDIOC_QBUF` ioctl to enqueue an empty (capturing) or filled (output) buffer in the driver's incoming queue. The semantics depend on the selected I/O method.

To enqueue a buffer applications set the `type` field of a struct *`v4l2_buffer`* to the same buffer type as was previously used with struct *`v4l2_format`* `type` and struct *`v4l2_requestbuffers`* `type`. Applications must also set the `index` field. Valid index numbers range from zero to the number of buffers allocated with *ioctl VIDIOC_REQBUFS* (struct *`v4l2_requestbuffers`* `count`) minus one. The contents of the struct *`v4l2_buffer`* returned by a *ioctl VIDIOC_QUERYBUF* ioctl will do as well. When the buffer is intended for output (`type` is `V4L2_BUF_TYPE_VIDEO_OUTPUT`, `V4L2_BUF_TYPE_VIDEO_OUTPUT_MPLANE`, or V4L2_BUF_TYPE_VBI_OUTPUT) applications must also initialize the bytesused, field and timestamp fields, see *Buffers* for details. Applications must also set flags to 0. The reserved2 and reserved fields must be set to 0. When using the *multi-planar API*, the m.planes field must contain a userspace pointer to a filled-in array of struct *v4l2_plane* and the length field must be set to the number of elements in that array.

To enqueue a *memory mapped* buffer applications set the memory field to V4L2_MEMORY_MMAP. When VIDIOC_QBUF is called with a pointer to this structure the driver sets the V4L2_BUF_FLAG_MAPPED and V4L2_BUF_FLAG_QUEUED flags and clears the V4L2_BUF_FLAG_DONE flag in the flags field, or it returns an EINVAL error code.

To enqueue a *user pointer* buffer applications set the memory field to V4L2_MEMORY_USERPTR, the m.userptr field to the address of the buffer and length to its size. When the multi-planar API is used, m.userptr and length members of the passed array of struct *v4l2_plane* have to be used instead. When VIDIOC_QBUF is called with a pointer to this structure the driver sets the V4L2_BUF_FLAG_QUEUED flag and clears the V4L2_BUF_FLAG_MAPPED and V4L2_BUF_FLAG_DONE flags in the flags field, or it returns an error code. This ioctl locks the memory pages of the buffer in physical memory, they cannot be swapped out to disk. Buffers remain locked until dequeued, until the *VIDIOC_STREAMOFF* or *ioctl VIDIOC_REQBUFS* ioctl is called, or until the device is closed.

To enqueue a *DMABUF* buffer applications set the memory field to V4L2_MEMORY_DMABUF and the m.fd field to a file descriptor associated with a DMABUF buffer. When the multi-planar API is used the m.fd fields of the passed array of struct *v4l2_plane* have to be used instead. When VIDIOC_QBUF is called with a pointer to this structure the driver sets the V4L2_BUF_FLAG_QUEUED flag and clears the V4L2_BUF_FLAG_MAPPED and V4L2_BUF_FLAG_DONE flags in the flags field, or it returns an error code. This ioctl locks the buffer. Locking a buffer means passing it to a driver for a hardware access (usually DMA). If an application accesses (reads/writes) a locked buffer then the result is undefined. Buffers remain locked until dequeued, until the *VIDIOC_STREAMOFF* or *ioctl VIDIOC_REQBUFS* ioctl is called, or until the device is closed.

The request_fd field can be used with the VIDIOC_QBUF ioctl to specify the file descriptor of a *request*, if requests are in use. Setting it means that the buffer will not be passed to the driver until the request itself is queued. Also, the driver will apply any settings associated with the request for this buffer. This field will be ignored unless the V4L2_BUF_FLAG_REQUEST_FD flag is set. 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.

Caution: It is not allowed to mix queuing requests with queuing buffers directly. EBUSY will be returned if the first buffer was queued directly and then the application tries to queue a request, or vice versa. After closing the file descriptor, calling *VIDIOC_STREAMOFF* or calling *ioctl VIDIOC_REQBUFS* the check for this will be reset.

For *memory-to-memory devices* you can specify the request_fd only for output buffers, not for capture buffers. Attempting to specify this for a capture buffer will result in an EBADR error.

Applications call the VIDIOC_DQBUF ioctl to dequeue a filled (capturing) or displayed (output) buffer from the driver's outgoing queue. They just set the type, memory and reserved fields of a struct *v4l2_buffer* as above, when VIDIOC_DQBUF is called with a pointer to this structure the driver fills the remaining fields or returns an error code. The driver may also set V4L2_BUF_FLAG_ERROR in the flags field. It indicates a non-critical (recoverable) streaming error. In such case the application may continue as normal, but should be aware that data in the dequeued buffer might be corrupted. When using the multi-planar API, the planes array must be passed in as well.

If the application sets the memory field to V4L2_MEMORY_DMABUF to dequeue a *DMABUF* buffer, the driver fills the m.fd field with a file descriptor numerically the same as the one given to VIDIOC_QBUF when the buffer was enqueued. No new file descriptor is created at dequeue time and the value is only for the application convenience. When the multi-planar API is used the m.fd fields of the passed array of struct *v4l2_plane* are filled instead.

By default VIDIOC_DQBUF blocks when no buffer is in the outgoing queue. When the O_NONBLOCK flag was given to the *open()* function, VIDIOC_DQBUF returns immediately with an EAGAIN error code when no buffer is available.

The struct *v4l2_buffer* structure is specified in *Buffers*.

Return Value

On success 0 is returned, on error -1 and the errno variable is set appropriately. The generic error codes are described at the *Generic Error Codes* chapter.

EAGAIN
: Non-blocking I/O has been selected using O_NONBLOCK and no buffer was in the outgoing queue.

EINVAL
: The buffer type is not supported, or the index is out of bounds, or no buffers have been allocated yet, or the userptr or length are invalid, or the V4L2_BUF_FLAG_REQUEST_FD flag was set but the the given request_fd was invalid, or m.fd was an invalid DMABUF file descriptor.

EIO
: VIDIOC_DQBUF failed due to an internal error. Can also indicate temporary problems like signal loss.

Note: The driver might dequeue an (empty) buffer despite returning an error, or even stop capturing. Reusing such buffer may be unsafe though and its details (e.g. index) may not be returned either. It is recommended that drivers indicate recoverable errors by setting the V4L2_BUF_FLAG_ERROR and returning 0 instead. In that case the application should be able to safely reuse the buffer and continue streaming.

EPIPE
VIDIOC_DQBUF returns this on an empty capture queue for mem2mem codecs if a buffer with the V4L2_BUF_FLAG_LAST was already dequeued and no new buffers are expected to become available.

EBADR
The V4L2_BUF_FLAG_REQUEST_FD flag was set but the device does not support requests for the given buffer type, or the V4L2_BUF_FLAG_REQUEST_FD flag was not set but the device requires that the buffer is part of a request.

EBUSY
The first buffer was queued via a request, but the application now tries to queue it directly, or vice versa (it is not permitted to mix the two APIs).

ioctl VIDIOC_QUERYBUF

Name

VIDIOC_QUERYBUF - Query the status of a buffer

Synopsis

VIDIOC_QUERYBUF

`int ioctl(int fd, VIDIOC_QUERYBUF, struct v4l2_buffer *argp)`

Arguments

fd
File descriptor returned by *open()*.

argp
Pointer to struct *v4l2_buffer*.

Description

This ioctl is part of the *streaming* I/O method. It can be used to query the status of a buffer at any time after buffers have been allocated with the *ioctl VIDIOC_REQBUFS* ioctl.

Applications set the `type` field of a struct *v4l2_buffer* to the same buffer type as was previously used with struct *v4l2_format* `type` and struct *v4l2_requestbuffers* `type`, and the `index` field. Valid index numbers range from zero to the number of buffers allocated with *ioctl VIDIOC_REQBUFS* (struct *v4l2_requestbuffers* `count`) minus one. The `reserved` and `reserved2` fields must be set to 0. When using the *multi-planar API*, the `m.planes` field must contain a userspace pointer to an array of struct *v4l2_plane* and the `length` field has to be set to the number of elements in that array. After calling *ioctl VIDIOC_QUERYBUF* with a pointer to this structure drivers return an error code or fill the rest of the structure.

In the `flags` field the `V4L2_BUF_FLAG_MAPPED`, `V4L2_BUF_FLAG_PREPARED`, `V4L2_BUF_FLAG_QUEUED` and `V4L2_BUF_FLAG_DONE` flags will be valid. The `memory` field will be set to the current I/O method. For the single-planar API, the `m.offset` contains the offset of the buffer from the start of the device memory, the `length` field its size. For the multi-planar API, fields `m.mem_offset` and `length` in the `m.planes` array elements will be used instead and the `length` field of struct *v4l2_buffer* is set to the number of filled-in array elements. The driver may or may not set the remaining fields and flags, they are meaningless in this context.

The struct *v4l2_buffer* structure is specified in *Buffers*.

Return Value

On success 0 is returned, on error -1 and the `errno` variable is set appropriately. The generic error codes are described at the *Generic Error Codes* chapter.

EINVAL
The buffer `type` is not supported, or the `index` is out of bounds.

ioctl VIDIOC_QUERYCAP

Name

VIDIOC_QUERYCAP - Query device capabilities

Synopsis

VIDIOC_QUERYCAP

```
int ioctl(int fd, VIDIOC_QUERYCAP, struct v4l2_capability *argp)
```

Arguments

fd
File descriptor returned by *open()*.

argp
Pointer to struct *v4l2_capability*.

Description

All V4L2 devices support the `VIDIOC_QUERYCAP` ioctl. It is used to identify kernel devices compatible with this specification and to obtain information about driver and hardware capabilities. The ioctl takes a pointer to a struct *v4l2_capability* which is filled by the driver. When the driver is not compatible with this specification the ioctl returns an `EINVAL` error code.

type **v4l2_capability**

Table 198: struct v4l2_capability

__u8	driver[16]	Name of the driver, a unique NUL-terminated ASCII string. For example: "bttv". Driver specific applications can use this information to verify the driver identity. It is also useful to work around known bugs, or to identify drivers in error reports. Storing strings in fixed sized arrays is bad practice but unavoidable here. Drivers and applications should take precautions to never read or write beyond the end of the array and to make sure the strings are properly NUL-terminated.
__u8	card[32]	Name of the device, a NUL-terminated UTF-8 string. For example: "Yoyodyne TV/FM". One driver may support different brands or models of video hardware. This information is intended for users, for example in a menu of available devices. Since multiple TV cards of the same brand may be installed which are supported by the same driver, this name should be combined with the character device file name (e. g. `/dev/video2`) or the `bus_info` string to avoid ambiguities.
__u8	bus_info[32]	Location of the device in the system, a NUL-terminated ASCII string. For example: "PCI:0000:05:06.0". This information is intended for users, to distinguish multiple identical devices. If no such information is available the field must simply count the devices controlled by the driver ("platform:vivid-000"). The bus_info must start with "PCI:" for PCI boards, "PCIe:" for PCI Express boards, "usb-" for USB devices, "I2C:" for i2c devices, "ISA:" for ISA devices, "parport" for parallel port devices and "platform:" for platform devices.
__u32	version	Version number of the driver. Starting with kernel 3.1, the version reported is provided by the V4L2 subsystem following the kernel numbering scheme. However, it may not always return the same version as the kernel if, for example, a stable or distribution-modified kernel uses the V4L2 stack from a newer kernel. The version number is formatted using the `KERNEL_VERSION()` macro. For example if the media stack corresponds to the V4L2 version shipped with Kernel 4.14, it would be equivalent to:

```
#define KERNEL_VERSION(a,b,c) (((a) << 16) + ((b) << 8) + (c))
__u32 version = KERNEL_VERSION(4, 14, 0);
printf ("Version: %u.%u.%u\\n",
(version >> 16) & 0xFF, (version >> 8) & 0xFF, version & 0xFF);
```

__u32	capabilities	Available capabilities of the physical device as a whole, see *Device Capabilities Flags*. The same physical device can export multiple devices in /dev (e.g. /dev/videoX, /dev/vbiY and /dev/radioZ). The `capabilities` field should contain a union of all capabilities available around the several V4L2 devices exported to userspace. For all those devices the `capabilities` field returns the same set of capabilities. This allows applications to open just one of the devices (typically the video device) and discover whether video, vbi and/or radio are also supported.
__u32	device_caps	Device capabilities of the opened device, see *Device Capabilities Flags*. Should contain the available capabilities of that specific device node. So, for example, `device_caps` of a radio device will only contain radio related capabilities and no video or vbi capabilities. This field is only set if the `capabilities` field contains the `V4L2_CAP_DEVICE_CAPS` capability. Only the `capabilities` field can have the `V4L2_CAP_DEVICE_CAPS` capability, `device_caps` will

Table 199: Device Capabilities Flags

V4L2_CAP_VIDEO_CAPTURE	0x0000000[illegible]	The device supports the single-planar API through the *Video Capture* interface.
V4L2_CAP_VIDEO_CAPTURE_MPLA	0x0000100[illegible]	The device supports the *multi-planar API* through the *Video Capture* interface.
V4L2_CAP_VIDEO_OUTPUT	0x0000000[illegible]	The device supports the single-planar API through the *Video Output* interface.
V4L2_CAP_VIDEO_OUTPUT_MPLAN	0x0000200[illegible]	The device supports the *multi-planar API* through the *Video Output* interface.
V4L2_CAP_VIDEO_M2M	0x0000800[illegible]	The device supports the single-planar API through the Video Memory-To-Memory interface.
V4L2_CAP_VIDEO_M2M_MPLANE	0x0000400[illegible]	The device supports the *multi-planar API* through the Video Memory-To-Memory interface.
V4L2_CAP_VIDEO_OVERLAY	0x0000000[illegible]	The device supports the *Video Overlay* interface. A video overlay device typically stores captured images directly in the video memory of a graphics card, with hardware clipping and scaling.
V4L2_CAP_VBI_CAPTURE	0x0000001[illegible]	The device supports the *Raw VBI Capture* interface, providing Teletext and Closed Caption data.
V4L2_CAP_VBI_OUTPUT	0x0000002[illegible]	The device supports the *Raw VBI Output* interface.
V4L2_CAP_SLICED_VBI_CAPTURE	0x0000004[illegible]	The device supports the *Sliced VBI Capture* interface.
V4L2_CAP_SLICED_VBI_OUTPUT	0x0000008[illegible]	The device supports the *Sliced VBI Output* interface.
V4L2_CAP_RDS_CAPTURE	0x0000010[illegible]	The device supports the *RDS* capture interface.
V4L2_CAP_VIDEO_OUTPUT_OVERL	0x0000020[illegible]	The device supports the *Video Output Overlay* (OSD) interface. Unlike the *Video Overlay* interface, this is a secondary function of video output devices and overlays an image onto an outgoing video signal. When the driver sets this flag, it must clear the `V4L2_CAP_VIDEO_OVERLAY` flag and vice versa.[1]
V4L2_CAP_HW_FREQ_SEEK	0x0000040[illegible]	The device supports the *ioctl VIDIOC_S_HW_FREQ_SEEK* ioctl for hardware frequency seeking.
V4L2_CAP_RDS_OUTPUT	0x0000080[illegible]	The device supports the *RDS* output interface.
V4L2_CAP_TUNER	0x0001000[illegible]	The device has some sort of tuner to receive RF-modulated video signals. For more information about tuner programming see *Tuners and Modulators*.

continues on next page

Table 199 – continued from previous page

V4L2_CAP_AUDIO	0x0002000(	The device has audio inputs or outputs. It may or may not support audio recording or playback, in PCM or compressed formats. PCM audio support must be implemented as ALSA or OSS interface. For more information on audio inputs and outputs see *Audio Inputs and Outputs*.
V4L2_CAP_RADIO	0x0004000(	This is a radio receiver.
V4L2_CAP_MODULATOR	0x0008000(	The device has some sort of modulator to emit RF-modulated video/audio signals. For more information about modulator programming see *Tuners and Modulators*.
V4L2_CAP_SDR_CAPTURE	0x0010000(	The device supports the *SDR Capture* interface.
V4L2_CAP_EXT_PIX_FORMAT	0x0020000(	The device supports the struct *v4l2_pix_format* extended fields.
V4L2_CAP_SDR_OUTPUT	0x0040000(	The device supports the *SDR Output* interface.
V4L2_CAP_META_CAPTURE	0x0080000(	The device supports the *Metadata Interface* capture interface.
V4L2_CAP_READWRITE	0x0100000(	The device supports the *read()* and/or *write()* I/O methods.
V4L2_CAP_ASYNCIO	0x0200000(	The device supports the *asynchronous* I/O methods.
V4L2_CAP_STREAMING	0x0400000(	The device supports the *streaming* I/O method.
V4L2_CAP_META_OUTPUT	0x0800000(	The device supports the *Metadata Interface* output interface.
V4L2_CAP_TOUCH	0x1000000(	This is a touch device.
V4L2_CAP_IO_MC	0x2000000(	There is only one input and/or output seen from userspace. The whole video topology configuration, including which I/O entity is routed to the input/output, is configured by userspace via the Media Controller. See *Part IV - Media Controller API*.
V4L2_CAP_DEVICE_CAPS	0x8000000(	The driver fills the device_caps field. This capability can only appear in the capabilities field and never in the device_caps field.

[1] The struct *v4l2_framebuffer* lacks an enum *v4l2_buf_type* field, therefore the type of overlay is implied by the driver capabilities.

Return Value

On success 0 is returned, on error -1 and the `errno` variable is set appropriately. The generic error codes are described at the *Generic Error Codes* chapter.

ioctls VIDIOC_QUERYCTRL, VIDIOC_QUERY_EXT_CTRL and VIDIOC_QUERYMENU

Name

VIDIOC_QUERYCTRL - VIDIOC_QUERY_EXT_CTRL - VIDIOC_QUERYMENU - Enumerate controls and menu control items

Synopsis

`int ioctl(int fd, int VIDIOC_QUERYCTRL, struct v4l2_queryctrl *argp)`

VIDIOC_QUERY_EXT_CTRL

`int ioctl(int fd, VIDIOC_QUERY_EXT_CTRL, struct v4l2_query_ext_ctrl *argp)`

VIDIOC_QUERYMENU

`int ioctl(int fd, VIDIOC_QUERYMENU, struct v4l2_querymenu *argp)`

Arguments

fd
: File descriptor returned by *open()*.

argp
: Pointer to struct `v4l2_queryctrl`, `v4l2_query_ext_ctrl` or `v4l2_querymenu` (depending on the ioctl).

Description

To query the attributes of a control applications set the `id` field of a struct *v4l2_queryctrl* and call the `VIDIOC_QUERYCTRL` ioctl with a pointer to this structure. The driver fills the rest of the structure or returns an `EINVAL` error code when the `id` is invalid.

It is possible to enumerate controls by calling `VIDIOC_QUERYCTRL` with successive `id` values starting from `V4L2_CID_BASE` up to and exclusive `V4L2_CID_LASTP1`. Drivers may return `EINVAL` if a control in this range is not supported. Further applications can enumerate private controls, which are not defined in this specification, by starting at `V4L2_CID_PRIVATE_BASE` and incrementing `id` until the driver returns `EINVAL`.

In both cases, when the driver sets the V4L2_CTRL_FLAG_DISABLED flag in the flags field this control is permanently disabled and should be ignored by the application.[1]

When the application ORs id with V4L2_CTRL_FLAG_NEXT_CTRL the driver returns the next supported non-compound control, or EINVAL if there is none. In addition, the V4L2_CTRL_FLAG_NEXT_COMPOUND flag can be specified to enumerate all compound controls (i.e. controls with type ≥ V4L2_CTRL_COMPOUND_TYPES and/or array control, in other words controls that contain more than one value). Specify both V4L2_CTRL_FLAG_NEXT_CTRL and V4L2_CTRL_FLAG_NEXT_COMPOUND in order to enumerate all controls, compound or not. Drivers which do not support these flags yet always return EINVAL.

The VIDIOC_QUERY_EXT_CTRL ioctl was introduced in order to better support controls that can use compound types, and to expose additional control information that cannot be returned in struct *v4l2_queryctrl* since that structure is full.

VIDIOC_QUERY_EXT_CTRL is used in the same way as VIDIOC_QUERYCTRL, except that the reserved array must be zeroed as well.

Additional information is required for menu controls: the names of the menu items. To query them applications set the id and index fields of struct *v4l2_querymenu* and call the VIDIOC_QUERYMENU ioctl with a pointer to this structure. The driver fills the rest of the structure or returns an EINVAL error code when the id or index is invalid. Menu items are enumerated by calling VIDIOC_QUERYMENU with successive index values from struct *v4l2_queryctrl* minimum to maximum, inclusive.

Note: It is possible for VIDIOC_QUERYMENU to return an EINVAL error code for some indices between minimum and maximum. In that case that particular menu item is not supported by this driver. Also note that the minimum value is not necessarily 0.

See also the examples in *User Controls*.

Table 200: struct v4l2_queryctrl

__u32	id	Identifies the control, set by the application. See *Control IDs* for predefined IDs. When the ID is ORed with V4L2_CTRL_FLAG_NEXT_CTRL the driver clears the flag and returns the first control with a higher ID. Drivers which do not support this flag yet always return an EINVAL error code.
__u32	type	Type of control, see *v4l2_ctrl_type*.
__u8	name[32]	Name of the control, a NUL-terminated ASCII string. This information is intended for the user.

continues on next page

[1] V4L2_CTRL_FLAG_DISABLED was intended for two purposes: Drivers can skip predefined controls not supported by the hardware (although returning EINVAL would do as well), or disable predefined and private controls after hardware detection without the trouble of reordering control arrays and indices (EINVAL cannot be used to skip private controls because it would prematurely end the enumeration).

Table 200 – continued from previous page

__s32	minimum	Minimum value, inclusive. This field gives a lower bound for the control. See enum *v4l2_ctrl_type* how the minimum value is to be used for each possible control type. Note that this a signed 32-bit value.
__s32	maximum	Maximum value, inclusive. This field gives an upper bound for the control. See enum *v4l2_ctrl_type* how the maximum value is to be used for each possible control type. Note that this a signed 32-bit value.
__s32	step	This field gives a step size for the control. See enum *v4l2_ctrl_type* how the step value is to be used for each possible control type. Note that this an unsigned 32-bit value. Generally drivers should not scale hardware control values. It may be necessary for example when the name or id imply a particular unit and the hardware actually accepts only multiples of said unit. If so, drivers must take care values are properly rounded when scaling, such that errors will not accumulate on repeated read-write cycles. This field gives the smallest change of an integer control actually affecting hardware. Often the information is needed when the user can change controls by keyboard or GUI buttons, rather than a slider. When for example a hardware register accepts values 0-511 and the driver reports 0-65535, step should be 128. Note that although signed, the step value is supposed to be always positive.
__s32	default_value	The default value of a V4L2_CTRL_TYPE_INTEGER, _BOOLEAN, _BITMASK, _MENU or _INTEGER_MENU control. Not valid for other types of controls. **Note:** Drivers reset controls to their default value only when the driver is first loaded, never afterwards.
__u32	flags	Control flags, see *Control Flags*.
__u32	reserved[2]	Reserved for future extensions. Drivers must set the array to zero.

Table 201: struct v4l2_query_ext_ctrl

__u32	id	Identifies the control, set by the application. See *Control IDs* for predefined IDs. When the ID is ORed with V4L2_CTRL_FLAG_NEXT_CTRL the driver clears the flag and returns the first non-compound control with a higher ID. When the ID is ORed with V4L2_CTRL_FLAG_NEXT_COMPOUND the driver clears the flag and returns the first compound control with a higher ID. Set both to get the first control (compound or not) with a higher ID.
__u32	type	Type of control, see *v4l2_ctrl_type*.
char	name[32]	Name of the control, a NUL-terminated ASCII string. This information is intended for the user.

continues on next page

Table 201 – continued from previous page

__s64	minimum	Minimum value, inclusive. This field gives a lower bound for the control. See enum *v4l2_ctrl_type* how the minimum value is to be used for each possible control type. Note that this a signed 64-bit value.
__s64	maximum	Maximum value, inclusive. This field gives an upper bound for the control. See enum *v4l2_ctrl_type* how the maximum value is to be used for each possible control type. Note that this a signed 64-bit value.
__u64	step	This field gives a step size for the control. See enum *v4l2_ctrl_type* how the step value is to be used for each possible control type. Note that this an unsigned 64-bit value. Generally drivers should not scale hardware control values. It may be necessary for example when the name or id imply a particular unit and the hardware actually accepts only multiples of said unit. If so, drivers must take care values are properly rounded when scaling, such that errors will not accumulate on repeated read-write cycles. This field gives the smallest change of an integer control actually affecting hardware. Often the information is needed when the user can change controls by keyboard or GUI buttons, rather than a slider. When for example a hardware register accepts values 0-511 and the driver reports 0-65535, step should be 128.
__s64	default_value	The default value of a V4L2_CTRL_TYPE_INTEGER, _INTEGER64, _BOOLEAN, _BITMASK, _MENU, _INTEGER_MENU, _U8 or _U16 control. Not valid for other types of controls. **Note:** Drivers reset controls to their default value only when the driver is first loaded, never afterwards.
__u32	flags	Control flags, see *Control Flags*.
__u32	elem_size	The size in bytes of a single element of the array. Given a char pointer p to a 3-dimensional array you can find the position of cell (z, y, x) as follows: p + ((z * dims[1] + y) * dims[0] + x) * elem_size. elem_size is always valid, also when the control isn't an array. For string controls elem_size is equal to maximum + 1.
__u32	elems	The number of elements in the N-dimensional array. If this control is not an array, then elems is 1. The elems field can never be 0.
__u32	nr_of_dims	The number of dimension in the N-dimensional array. If this control is not an array, then this field is 0.
__u32	dims[V4L2_CTRL_MAX_DIM	The size of each dimension. The first nr_of_dims elements of this array must be non-zero, all remaining elements must be zero.
__u32	reserved[32]	Reserved for future extensions. Applications and drivers must set the array to zero.

Table 202: struct v4l2_querymenu

`__u32`	`id`	Identifies the control, set by the application from the respective struct *v4l2_queryctrl* `id`.
`__u32`	`index`	Index of the menu item, starting at zero, set by the application.
union {	(anonymous)	
`__u8`	`name[`	Name of the menu item, a NUL-terminated ASCII string. This information is intended for the user. This field is valid for `V4L2_CTRL_TYPE_MENU` type controls.
`__s64`	`value`	Value of the integer menu item. This field is valid for `V4L2_CTRL_TYPE_INTEGER_MENU`

type **v4l2_ctrl_type**

Table 203: enum v4l2_ctrl_type

Type	minimum	step	maximum	Description
V4L2_CTRL_TYPE_INTEGER	any	any	any	An integer-valued control ranging from minimum to maximum inclusive. The step value indicates the increment between values.
V4L2_CTRL_TYPE_BOOLEAN	0	1	1	A boolean-valued control. Zero corresponds to "disabled", and one means "enabled".
V4L2_CTRL_TYPE_MENU	≥ 0	1	N-1	The control has a menu of N choices. The names of the menu items can be enumerated with the VIDIOC_QUERYMENU ioctl.
V4L2_CTRL_TYPE_INTEGER_MENU	≥ 0	1	N-1	The control has a menu of N choices. The values of the menu items can be enumerated with the VIDIOC_QUERYMENU ioctl. This is similar to V4L2_CTRL_TYPE_MENU except that instead of strings, the menu items are signed 64-bit integers.
V4L2_CTRL_TYPE_BITMASK	0	n/a	any	A bitmask field. The maximum value is the set of bits that can be used, all other bits are to be 0. The maximum value is interpreted as a __u32, allowing the use of bit 31 in the bitmask.
V4L2_CTRL_TYPE_BUTTON	0	0	0	A control which performs an action when set. Drivers must ignore the value passed with VIDIOC_S_CTRL and return an EACCES error code on a VIDIOC_G_CTRL attempt.
V4L2_CTRL_TYPE_INTEGER64	any	any	any	A 64-bit integer valued control. Minimum, maximum and step size cannot be queried using VIDIOC_QUERYCTRL. Only VIDIOC_QUERY_EXT_CTRL can retrieve the 64-bit min/max/step values, they should be interpreted as n/a when using VIDIOC_QUERYCTRL.

continues on next page

Table 203 – continued from previous page

Type	minimum	step	maximum	Description
V4L2_CTRL_TYPE_STRING	≥ 0	≥ 1	≥ 0	The minimum and maximum string lengths. The step size means that the string must be (minimum + N * step) characters long for N ≥ 0. These lengths do not include the terminating zero, so in order to pass a string of length 8 to *VIDIOC_S_EXT_CTRLS* you need to set the `size` field of struct *v4l2_ext_control* to 9. For *VIDIOC_G_EXT_CTRLS* you can set the `size` field to `maximum` + 1. Which character encoding is used will depend on the string control itself and should be part of the control documentation.
V4L2_CTRL_TYPE_CTRL_CLASS	n/a	n/a	n/a	This is not a control. When `VIDIOC_QUERYCTRL` is called with a control ID equal to a control class code (see *Control classes*) + 1, the ioctl returns the name of the control class and this control type. Older drivers which do not support this feature return an `EINVAL` error code.
V4L2_CTRL_TYPE_U8	any	any	any	An unsigned 8-bit valued control ranging from minimum to maximum inclusive. The step value indicates the increment between values.
V4L2_CTRL_TYPE_U16	any	any	any	An unsigned 16-bit valued control ranging from minimum to maximum inclusive. The step value indicates the increment between values.
V4L2_CTRL_TYPE_U32	any	any	any	An unsigned 32-bit valued control ranging from minimum to maximum inclusive. The step value indicates the increment between values.
V4L2_CTRL_TYPE_MPEG2_SLICI	n/a	n/a	n/a	A struct *v4l2_ctrl_mpeg2_slice_params*, containing MPEG-2 slice parameters for stateless video decoders.
V4L2_CTRL_TYPE_MPEG2_QUAN	n/a	n/a	n/a	A struct *v4l2_ctrl_mpeg2_quantization*, containing MPEG-2 quantization matrices for stateless video decoders.

continues on next page

Table 203 – continued from previous page

Type	minimum	step	maximum	Description
V4L2_CTRL_TYPE_AREA	n/a	n/a	n/a	A struct v4l2_area, containing the width and the height of a rectangular area. Units depend on the use case.
V4L2_CTRL_TYPE_H264_SPS	n/a	n/a	n/a	A struct *v4l2_ctrl_h264_sps*, containing H264 sequence parameters for stateless video decoders.
V4L2_CTRL_TYPE_H264_PPS	n/a	n/a	n/a	A struct *v4l2_ctrl_h264_pps*, containing H264 picture parameters for stateless video decoders.
V4L2_CTRL_TYPE_H264_SCALI	n/a	n/a	n/a	A struct *v4l2_ctrl_h264_scaling_matri* containing H264 scaling matrices for stateless video decoders.
V4L2_CTRL_TYPE_H264_SLICE_	n/a	n/a	n/a	A struct *v4l2_ctrl_h264_slice_params*, containing H264 slice parameters for stateless video decoders.
V4L2_CTRL_TYPE_H264_DECOD	n/a	n/a	n/a	A struct *v4l2_ctrl_h264_decode_params*, containing H264 decode parameters for stateless video decoders.
V4L2_CTRL_TYPE_HEVC_SPS	n/a	n/a	n/a	A struct *v4l2_ctrl_hevc_sps*, containing HEVC Sequence Parameter Set for stateless video decoders.
V4L2_CTRL_TYPE_HEVC_PPS	n/a	n/a	n/a	A struct *v4l2_ctrl_hevc_pps*, containing HEVC Picture Parameter Set for stateless video decoders.
V4L2_CTRL_TYPE_HEVC_SLICE_	n/a	n/a	n/a	A struct *v4l2_ctrl_hevc_slice_params*, containing HEVC slice parameters for stateless video decoders.

Table 204: Control Flags

V4L2_CTRL_FLAG_DISABLED	0x0001	This control is permanently disabled and should be ignored by the application. Any attempt to change the control will result in an EINVAL error code.
V4L2_CTRL_FLAG_GRABBED	0x0002	This control is temporarily unchangeable, for example because another application took over control of the respective resource. Such controls may be displayed specially in a user interface. Attempts to change the control may result in an EBUSY error code.
V4L2_CTRL_FLAG_READ_ONLY	0x0004	This control is permanently readable only. Any attempt to change the control will result in an EINVAL error code.

continues on next page

Table 204 – continued from previous page

V4L2_CTRL_FLAG_UPDATE	0x0008	A hint that changing this control may affect the value of other controls within the same control class. Applications should update their user interface accordingly.
V4L2_CTRL_FLAG_INACTIVE	0x0010	This control is not applicable to the current configuration and should be displayed accordingly in a user interface. For example the flag may be set on a MPEG audio level 2 bitrate control when MPEG audio encoding level 1 was selected with another control.
V4L2_CTRL_FLAG_SLIDER	0x0020	A hint that this control is best represented as a slider-like element in a user interface.
V4L2_CTRL_FLAG_WRITE_ONLY	0x0040	This control is permanently writable only. Any attempt to read the control will result in an EACCES error code error code. This flag is typically present for relative controls or action controls where writing a value will cause the device to carry out a given action (e. g. motor control) but no meaningful value can be returned.
V4L2_CTRL_FLAG_VOLATILE	0x0080	This control is volatile, which means that the value of the control changes continuously. A typical example would be the current gain value if the device is in auto-gain mode. In such a case the hardware calculates the gain value based on the lighting conditions which can change over time. **Note:** Setting a new value for a volatile control will be ignored unless *V4L2_CTRL_FLAG_EXECUTE_ON_WRITE* is also set. Setting a new value for a volatile control will *never* trigger a *V4L2_EVENT_CTRL_CH_VALUE* event.
V4L2_CTRL_FLAG_HAS_PAYLOAD	0x0100	This control has a pointer type, so its value has to be accessed using one of the pointer fields of struct *v4l2_ext_control*. This flag is set for controls that are an array, string, or have a compound type. In all cases you have to set a pointer to memory containing the payload of the control.

continues on next page

Table 204 – continued from previous page

V4L2_CTRL_FLAG_EXECUTE_ON_WRI	0x0200	The value provided to the control will be propagated to the driver even if it remains constant. This is required when the control represents an action on the hardware. For example: clearing an error flag or triggering the flash. All the controls of the type V4L2_CTRL_TYPE_BUTTON have this flag set.
V4L2_CTRL_FLAG_MODIFY_LAYOUT	0x0400	Changing this control value may modify the layout of the buffer (for video devices) or the media bus format (for sub-devices). A typical example would be the V4L2_CID_ROTATE control. Note that typically controls with this flag will also set the V4L2_CTRL_FLAG_GRABBED flag when buffers are allocated or streaming is in progress since most drivers do not support changing the format in that case.

Return Value

On success 0 is returned, on error -1 and the `errno` variable is set appropriately. The generic error codes are described at the *Generic Error Codes* chapter.

EINVAL
: The struct *v4l2_queryctrl* `id` is invalid. The struct *v4l2_querymenu* `id` is invalid or `index` is out of range (less than `minimum` or greater than `maximum`) or this particular menu item is not supported by the driver.

EACCES
: An attempt was made to read a write-only control.

ioctl VIDIOC_QUERY_DV_TIMINGS

Name

VIDIOC_QUERY_DV_TIMINGS - VIDIOC_SUBDEV_QUERY_DV_TIMINGS - Sense the DV preset received by the current input

Synopsis

VIDIOC_QUERY_DV_TIMINGS

`int ioctl(int fd, VIDIOC_QUERY_DV_TIMINGS, struct v4l2_dv_timings *argp)`

VIDIOC_SUBDEV_QUERY_DV_TIMINGS

`int ioctl(int fd, VIDIOC_SUBDEV_QUERY_DV_TIMINGS, struct v4l2_dv_timings *argp)`

Arguments

fd
File descriptor returned by *open()*.

argp
Pointer to struct *v4l2_dv_timings*.

Description

The hardware may be able to detect the current DV timings automatically, similar to sensing the video standard. To do so, applications call *ioctl VIDIOC_QUERY_DV_TIMINGS* with a pointer to a struct *v4l2_dv_timings*. Once the hardware detects the timings, it will fill in the timings structure.

Note: Drivers shall *not* switch timings automatically if new timings are detected. Instead, drivers should send the V4L2_EVENT_SOURCE_CHANGE event (if they support this) and expect that userspace will take action by calling *ioctl VIDIOC_QUERY_DV_TIMINGS*. The reason is that new timings usually mean different buffer sizes as well, and you cannot change buffer sizes on the fly. In general, applications that receive the Source Change event will have to call *ioctl VIDIOC_QUERY_DV_TIMINGS*, and if the detected timings are valid they will have to stop streaming, set the new timings, allocate new buffers and start streaming again.

If the timings could not be detected because there was no signal, then ENOLINK is returned. If a signal was detected, but it was unstable and the receiver could not lock to the signal, then ENOLCK is returned. If the receiver could lock to the signal, but the format is unsupported (e.g. because the pixelclock is out of range of the hardware capabilities), then the driver fills in whatever timings it could find and returns ERANGE. In that case the application can call *ioctl VIDIOC_DV_TIMINGS_CAP, VIDIOC_SUBDEV_DV_TIMINGS_CAP* to compare the found timings with the hardware's capabilities in order to give more precise feedback to the user.

Return Value

On success 0 is returned, on error -1 and the `errno` variable is set appropriately. The generic error codes are described at the *Generic Error Codes* chapter.

ENODATA
Digital video timings are not supported for this input or output.

ENOLINK
No timings could be detected because no signal was found.

ENOLCK
The signal was unstable and the hardware could not lock on to it.

ERANGE
Timings were found, but they are out of range of the hardware capabilities.

ioctl VIDIOC_QUERYSTD, VIDIOC_SUBDEV_QUERYSTD

Name

VIDIOC_QUERYSTD - VIDIOC_SUBDEV_QUERYSTD - Sense the video standard received by the current input

Synopsis

`VIDIOC_QUERYSTD`

`int ioctl(int fd, VIDIOC_QUERYSTD, v4l2_std_id *argp)`

`VIDIOC_SUBDEV_QUERYSTD`

`int ioctl(int fd, VIDIOC_SUBDEV_QUERYSTD, v4l2_std_id *argp)`

Arguments

`fd`
File descriptor returned by *`open()`*.

`argp`
Pointer to `v4l2_std_id`.

Description

The hardware may be able to detect the current video standard automatically. To do so, applications call *ioctl VIDIOC_QUERYSTD, VIDIOC_SUBDEV_QUERYSTD* with a pointer to a *v4l2_std_id* type. The driver stores here a set of candidates, this can be a single flag or a set of supported standards if for example the hardware can only distinguish between 50 and 60 Hz systems. If no signal was detected, then the driver will return V4L2_STD_UNKNOWN. When detection is not possible or fails, the set must contain all standards supported by the current video input or output.

Note: Drivers shall *not* switch the video standard automatically if a new video standard is detected. Instead, drivers should send the `V4L2_EVENT_SOURCE_CHANGE` event (if they support this) and expect that userspace will take action by calling *ioctl VIDIOC_QUERYSTD, VIDIOC_SUBDEV_QUERYSTD*. The reason is that a new video standard can mean different buffer sizes as well, and you cannot change buffer sizes on the fly. In general, applications that receive the Source Change event will have to call *ioctl VIDIOC_QUERYSTD, VIDIOC_SUBDEV_QUERYSTD*, and if the detected video standard is valid they will have to stop streaming, set the new standard, allocate new buffers and start streaming again.

Return Value

On success 0 is returned, on error -1 and the `errno` variable is set appropriately. The generic error codes are described at the *Generic Error Codes* chapter.

ENODATA
Standard video timings are not supported for this input or output.

ioctl VIDIOC_REQBUFS

Name

VIDIOC_REQBUFS - Initiate Memory Mapping, User Pointer I/O or DMA buffer I/O

Synopsis

`VIDIOC_REQBUFS`

```
int ioctl(int fd, VIDIOC_REQBUFS, struct v4l2_requestbuffers *argp)
```

Arguments

fd
File descriptor returned by *open()*.

argp
Pointer to struct *v4l2_requestbuffers*.

Description

This ioctl is used to initiate *memory mapped*, *user pointer* or *DMABUF* based I/O. Memory mapped buffers are located in device memory and must be allocated with this ioctl before they can be mapped into the application's address space. User buffers are allocated by applications themselves, and this ioctl is merely used to switch the driver into user pointer I/O mode and to setup some internal structures. Similarly, DMABUF buffers are allocated by applications through a device driver, and this ioctl only configures the driver into DMABUF I/O mode without performing any direct allocation.

To allocate device buffers applications initialize all fields of the struct *v4l2_requestbuffers* structure. They set the `type` field to the respective stream or buffer type, the `count` field to the desired number of buffers, `memory` must be set to the requested I/O method and the `reserved` array must be zeroed. When the ioctl is called with a pointer to this structure the driver will attempt to allocate the requested number of buffers and it stores the actual number allocated in the `count` field. It can be smaller than the number requested, even zero, when the driver runs out of free memory. A larger number is also possible when the driver requires more buffers to function correctly. For example video output requires at least two buffers, one displayed and one filled by the application.

When the I/O method is not supported the ioctl returns an `EINVAL` error code.

Applications can call *ioctl VIDIOC_REQBUFS* again to change the number of buffers. Note that if any buffers are still mapped or exported via DMABUF, then *ioctl VIDIOC_REQBUFS* can only succeed if the `V4L2_BUF_CAP_SUPPORTS_ORPHANED_BUFS` capability is set. Otherwise *ioctl VIDIOC_REQBUFS* will return the `EBUSY` error code. If `V4L2_BUF_CAP_SUPPORTS_ORPHANED_BUFS` is set, then these buffers are orphaned and will be freed when they are unmapped or when the exported DMABUF fds are closed. A `count` value of zero frees or orphans all buffers, after aborting or finishing any DMA in progress, an implicit *VIDIOC_STREAMOFF*.

type **v4l2_requestbuffers**

Table 205: struct v4l2_requestbuffers

__u32	count	The number of buffers requested or granted.
__u32	type	Type of the stream or buffers, this is the same as the struct *v4l2_format* type field. See *v4l2_buf_type* for valid values.
__u32	memory	Applications set this field to V4L2_MEMORY_MMAP, V4L2_MEMORY_DMABUF or V4L2_MEMORY_USERPTR. See v4l2_memory.
__u32	capabilities	Set by the driver. If 0, then the driver doesn't support capabilities. In that case all you know is that the driver is guaranteed to support V4L2_MEMORY_MMAP and *might* support other v4l2_memory types. It will not support any other capabilities. If you want to query the capabilities with a minimum of side-effects, then this can be called with count set to 0, memory set to V4L2_MEMORY_MMAP and type set to the buffer type. This will free any previously allocated buffers, so this is typically something that will be done at the start of the application.
__u32	reserved[1]	A place holder for future extensions. Drivers and applications must set the array to zero.

Table 206: V4L2 Buffer Capabilities Flags

V4L2_BUF_CAP_SUPPORTS_MMAP	0x00000001	This buffer type supports the V4L2_MEMORY_MMAP streaming mode.
V4L2_BUF_CAP_SUPPORTS_USERF	0x00000002	This buffer type supports the V4L2_MEMORY_USERPTR streaming mode.
V4L2_BUF_CAP_SUPPORTS_DMABU	0x00000004	This buffer type supports the V4L2_MEMORY_DMABUF streaming mode.
V4L2_BUF_CAP_SUPPORTS_REQUE	0x00000008	This buffer type supports *requests*.
V4L2_BUF_CAP_SUPPORTS_ORPHA	0x00000010	The kernel allows calling *ioctl VIDIOC_REQBUFS* while buffers are still mapped or exported via DMABUF. These orphaned buffers will be freed when they are unmapped or when the exported DMABUF fds are closed.
V4L2_BUF_CAP_SUPPORTS_M2M_H	0x00000020	Only valid for stateless decoders. If set, then userspace can set the V4L2_BUF_FLAG_M2M_HOLD_CAPTURE_BUF flag to hold off on returning the capture buffer until the OUTPUT timestamp changes.

continues on next page

Table 206 – continued from previous page

V4L2_BUF_CAP_SUPPORTS_MMAP_	0x0000004(	This capability is set by the driver to indicate that the queue supports cache and memory management hints. However, it' s only valid when the queue is used for *memory mapping* streaming I/O. See *V4L2_BUF_FLAG_NO_CACHE_INVALIDATE* and *V4L2_BUF_FLAG_NO_CACHE_CLEAN*.

Return Value

On success 0 is returned, on error -1 and the `errno` variable is set appropriately. The generic error codes are described at the *Generic Error Codes* chapter.

EINVAL
The buffer type (`type` field) or the requested I/O method (`memory`) is not supported.

ioctl VIDIOC_S_HW_FREQ_SEEK

Name

VIDIOC_S_HW_FREQ_SEEK - Perform a hardware frequency seek

Synopsis

VIDIOC_S_HW_FREQ_SEEK

```
int ioctl(int fd, VIDIOC_S_HW_FREQ_SEEK, struct v4l2_hw_freq_seek
*argp)
```

Arguments

fd
File descriptor returned by *open()*.

argp
Pointer to struct *v4l2_hw_freq_seek*.

Description

Start a hardware frequency seek from the current frequency. To do this applications initialize the `tuner`, `type`, `seek_upward`, `wrap_around`, `spacing`, `rangelow` and `rangehigh` fields, and zero out the `reserved` array of a struct *v4l2_hw_freq_seek* and call the `VIDIOC_S_HW_FREQ_SEEK` ioctl with a pointer to this structure.

The `rangelow` and `rangehigh` fields can be set to a non-zero value to tell the driver to search a specific band. If the struct *v4l2_tuner* `capability` field has the `V4L2_TUNER_CAP_HWSEEK_PROG_LIM` flag set, these values must fall within one of the bands returned by *ioctl VIDIOC_ENUM_FREQ_BANDS*. If the `V4L2_TUNER_CAP_HWSEEK_PROG_LIM` flag is not set, then these values must exactly match those of one of the bands returned by *ioctl VIDIOC_ENUM_FREQ_BANDS*. If the current frequency of the tuner does not fall within the selected band it will be clamped to fit in the band before the seek is started.

If an error is returned, then the original frequency will be restored.

This ioctl is supported if the `V4L2_CAP_HW_FREQ_SEEK` capability is set.

If this ioctl is called from a non-blocking filehandle, then `EAGAIN` error code is returned and no seek takes place.

type **v4l2_hw_freq_seek**

Table 207: struct v4l2_hw_freq_seek

__u32	tuner	The tuner index number. This is the same value as in the struct *v4l2_input* tuner field and the struct *v4l2_tuner* index field.
__u32	type	The tuner type. This is the same value as in the struct *v4l2_tuner* type field. See *v4l2_tuner_type*
__u32	seek_upward	If non-zero, seek upward from the current frequency, else seek downward.
__u32	wrap_around	If non-zero, wrap around when at the end of the frequency range, else stop seeking. The struct *v4l2_tuner* capability field will tell you what the hardware supports.
__u32	spacing	If non-zero, defines the hardware seek resolution in Hz. The driver selects the nearest value that is supported by the device. If spacing is zero a reasonable default value is used.
__u32	rangelow	If non-zero, the lowest tunable frequency of the band to search in units of 62.5 kHz, or if the struct *v4l2_tuner* capability field has the V4L2_TUNER_CAP_LOW flag set, in units of 62.5 Hz or if the struct *v4l2_tuner* capability field has the V4L2_TUNER_CAP_1HZ flag set, in units of 1 Hz. If rangelow is zero a reasonable default value is used.
__u32	rangehigh	If non-zero, the highest tunable frequency of the band to search in units of 62.5 kHz, or if the struct *v4l2_tuner* capability field has the V4L2_TUNER_CAP_LOW flag set, in units of 62.5 Hz or if the struct *v4l2_tuner* capability field has the V4L2_TUNER_CAP_1HZ flag set, in units of 1 Hz. If rangehigh is zero a reasonable default value is used.
__u32	reserved[5]	Reserved for future extensions. Applications must set the array to zero.

Return Value

On success 0 is returned, on error -1 and the errno variable is set appropriately. The generic error codes are described at the *Generic Error Codes* chapter.

EINVAL
: The tuner index is out of bounds, the wrap_around value is not supported or one of the values in the type, rangelow or rangehigh fields is wrong.

EAGAIN
: Attempted to call VIDIOC_S_HW_FREQ_SEEK with the filehandle in non-blocking mode.

ENODATA
The hardware seek found no channels.

EBUSY
Another hardware seek is already in progress.

ioctl VIDIOC_STREAMON, VIDIOC_STREAMOFF

Name

VIDIOC_STREAMON - VIDIOC_STREAMOFF - Start or stop streaming I/O

Synopsis

VIDIOC_STREAMON

`int ioctl(int fd, VIDIOC_STREAMON, const int *argp)`

VIDIOC_STREAMOFF

`int ioctl(int fd, VIDIOC_STREAMOFF, const int *argp)`

Arguments

fd
File descriptor returned by *open()*.

argp
Pointer to an integer.

Description

The `VIDIOC_STREAMON` and `VIDIOC_STREAMOFF` ioctl start and stop the capture or output process during streaming (*memory mapping*, *user pointer* or *DMABUF*) I/O.

Capture hardware is disabled and no input buffers are filled (if there are any empty buffers in the incoming queue) until `VIDIOC_STREAMON` has been called. Output hardware is disabled and no video signal is produced until `VIDIOC_STREAMON` has been called. The ioctl will succeed when at least one output buffer is in the incoming queue.

Memory-to-memory devices will not start until `VIDIOC_STREAMON` has been called for both the capture and output stream types.

If `VIDIOC_STREAMON` fails then any already queued buffers will remain queued.

The `VIDIOC_STREAMOFF` ioctl, apart of aborting or finishing any DMA in progress, unlocks any user pointer buffers locked in physical memory, and it removes all buffers from the incoming and outgoing queues. That means all images captured but not dequeued yet will be lost, likewise all images enqueued for output but not transmitted yet. I/O returns to the same state as after calling *ioctl VIDIOC_REQBUFS* and can be restarted accordingly.

If buffers have been queued with *ioctl VIDIOC_QBUF, VIDIOC_DQBUF* and `VIDIOC_STREAMOFF` is called without ever having called `VIDIOC_STREAMON`, then those queued buffers will also be removed from the incoming queue and all are returned to the same state as after calling *ioctl VIDIOC_REQBUFS* and can be restarted accordingly.

Both ioctls take a pointer to an integer, the desired buffer or stream type. This is the same as struct *v4l2_requestbuffers* `type`.

If `VIDIOC_STREAMON` is called when streaming is already in progress, or if `VIDIOC_STREAMOFF` is called when streaming is already stopped, then 0 is returned. Nothing happens in the case of `VIDIOC_STREAMON`, but `VIDIOC_STREAMOFF` will return queued buffers to their starting state as mentioned above.

Note: Applications can be preempted for unknown periods right before or after the `VIDIOC_STREAMON` or `VIDIOC_STREAMOFF` calls, there is no notion of starting or stopping "now". Buffer timestamps can be used to synchronize with other events.

Return Value

On success 0 is returned, on error -1 and the `errno` variable is set appropriately. The generic error codes are described at the *Generic Error Codes* chapter.

EINVAL
The buffer `type` is not supported, or no buffers have been allocated (memory mapping) or enqueued (output) yet.

EPIPE
The driver implements *pad-level format configuration* and the pipeline configuration is invalid.

ENOLINK
The driver implements Media Controller interface and the pipeline link configuration is invalid.

ioctl VIDIOC_SUBDEV_ENUM_FRAME_INTERVAL

Name

VIDIOC_SUBDEV_ENUM_FRAME_INTERVAL - Enumerate frame intervals

Synopsis

VIDIOC_SUBDEV_ENUM_FRAME_INTERVAL

```
int ioctl(int fd, VIDIOC_SUBDEV_ENUM_FRAME_INTERVAL, struct
v4l2_subdev_frame_interval_enum * argp)
```

Arguments

fd
File descriptor returned by *open()*.

argp
Pointer to struct *v4l2_subdev_frame_interval_enum*.

Description

This ioctl lets applications enumerate available frame intervals on a given sub-device pad. Frame intervals only makes sense for sub-devices that can control the frame period on their own. This includes, for instance, image sensors and TV tuners.

For the common use case of image sensors, the frame intervals available on the sub-device output pad depend on the frame format and size on the same pad. Applications must thus specify the desired format and size when enumerating frame intervals.

To enumerate frame intervals applications initialize the `index`, `pad`, `which`, `code`, `width` and `height` fields of struct *v4l2_subdev_frame_interval_enum* and call the *ioctl VIDIOC_SUBDEV_ENUM_FRAME_INTERVAL* ioctl with a pointer to this structure. Drivers fill the rest of the structure or return an EINVAL error code if one of the input fields is invalid. All frame intervals are enumerable by beginning at index zero and incrementing by one until `EINVAL` is returned.

Available frame intervals may depend on the current 'try' formats at other pads of the sub-device, as well as on the current active links. See *ioctl VIDIOC_SUBDEV_G_FMT, VIDIOC_SUBDEV_S_FMT* for more information about the try formats.

Sub-devices that support the frame interval enumeration ioctl should implemented it on a single pad only. Its behaviour when supported on multiple pads of the same sub-device is not defined.

type **v4l2_subdev_frame_interval_enum**

Table 208: struct v4l2_subdev_frame_interval_enum

__u32	index	Number of the format in the enumeration, set by the application.
__u32	pad	Pad number as reported by the media controller API.
__u32	code	The media bus format code, as defined in *Media Bus Formats*.
__u32	width	Frame width, in pixels.
__u32	height	Frame height, in pixels.
struct *v4l2_fract*	interval	Period, in seconds, between consecutive video frames.
__u32	which	Frame intervals to be enumerated, from enum *v4l2_subdev_format_whence*.
__u32	reserved[8]	Reserved for future extensions. Applications and drivers must set the array to zero.

Return Value

On success 0 is returned, on error -1 and the `errno` variable is set appropriately. The generic error codes are described at the *Generic Error Codes* chapter.

EINVAL
The struct *v4l2_subdev_frame_interval_enum* `pad` references a non-existing pad, one of the `code`, `width` or `height` fields are invalid for the given pad or the `index` field is out of bounds.

ioctl VIDIOC_SUBDEV_ENUM_FRAME_SIZE

Name

VIDIOC_SUBDEV_ENUM_FRAME_SIZE - Enumerate media bus frame sizes

Synopsis

VIDIOC_SUBDEV_ENUM_FRAME_SIZE

```
int ioctl(int fd, VIDIOC_SUBDEV_ENUM_FRAME_SIZE, struct
v4l2_subdev_frame_size_enum * argp)
```

Arguments

fd
File descriptor returned by *open()*.

argp
Pointer to struct *v4l2_subdev_frame_size_enum*.

Description

This ioctl allows applications to enumerate all frame sizes supported by a sub-device on the given pad for the given media bus format. Supported formats can be retrieved with the *ioctl VIDIOC_SUBDEV_ENUM_MBUS_CODE* ioctl.

To enumerate frame sizes applications initialize the `pad`, `which` , `code` and `index` fields of the struct *v4l2_subdev_mbus_code_enum* and call the *ioctl VIDIOC_SUBDEV_ENUM_FRAME_SIZE* ioctl with a pointer to the structure. Drivers fill the minimum and maximum frame sizes or return an EINVAL error code if one of the input parameters is invalid.

Sub-devices that only support discrete frame sizes (such as most sensors) will return one or more frame sizes with identical minimum and maximum values.

Not all possible sizes in given [minimum, maximum] ranges need to be supported. For instance, a scaler that uses a fixed-point scaling ratio might not be able to produce every frame size between the minimum and maximum values. Applications must use the *VIDIOC_SUBDEV_S_FMT* ioctl to try the sub-device for an exact supported frame size.

Available frame sizes may depend on the current 'try' formats at other pads of the sub-device, as well as on the current active links and the current values of V4L2 controls. See *ioctl VIDIOC_SUBDEV_G_FMT, VIDIOC_SUBDEV_S_FMT* for more information about try formats.

type **v4l2_subdev_frame_size_enum**

Table 209: struct v4l2_subdev_frame_size_enum

__u32	index	Number of the format in the enumeration, set by the application.
__u32	pad	Pad number as reported by the media controller API.
__u32	code	The media bus format code, as defined in *Media Bus Formats*.
__u32	min_width	Minimum frame width, in pixels.
__u32	max_width	Maximum frame width, in pixels.
__u32	min_height	Minimum frame height, in pixels.
__u32	max_height	Maximum frame height, in pixels.
__u32	which	Frame sizes to be enumerated, from enum *v4l2_subdev_format_whence*.
__u32	reserved[8]	Reserved for future extensions. Applications and drivers must set the array to zero.

Return Value

On success 0 is returned, on error -1 and the `errno` variable is set appropriately. The generic error codes are described at the *Generic Error Codes* chapter.

EINVAL
: The struct *v4l2_subdev_frame_size_enum* `pad` references a non-existing pad, the `code` is invalid for the given pad or the `index` field is out of bounds.

ioctl VIDIOC_SUBDEV_ENUM_MBUS_CODE

Name

VIDIOC_SUBDEV_ENUM_MBUS_CODE - Enumerate media bus formats

Synopsis

VIDIOC_SUBDEV_ENUM_MBUS_CODE

```
int ioctl(int fd, VIDIOC_SUBDEV_ENUM_MBUS_CODE, struct
v4l2_subdev_mbus_code_enum * argp)
```

Arguments

fd
: File descriptor returned by *open()*.

argp
: Pointer to struct *v4l2_subdev_mbus_code_enum*.

Description

To enumerate media bus formats available at a given sub-device pad applications initialize the `pad`, `which` and `index` fields of struct *v4l2_subdev_mbus_code_enum* and call the *ioctl VIDIOC_SUBDEV_ENUM_MBUS_CODE* ioctl with a pointer to this structure. Drivers fill the rest of the structure or return an `EINVAL` error code if either the `pad` or `index` are invalid. All media bus formats are enumerable by beginning at index zero and incrementing by one until `EINVAL` is returned.

Available media bus formats may depend on the current 'try' formats at other pads of the sub-device, as well as on the current active links. See *ioctl VIDIOC_SUBDEV_G_FMT, VIDIOC_SUBDEV_S_FMT* for more information about the try formats.

type **v4l2_subdev_mbus_code_enum**

Table 210: struct v4l2_subdev_mbus_code_enum

__u32	pad	Pad number as reported by the media controller API.
__u32	index	Number of the format in the enumeration, set by the application.
__u32	code	The media bus format code, as defined in *Media Bus Formats*.
__u32	which	Media bus format codes to be enumerated, from enum *v4l2_subdev_format_whence*.
__u32	flags	See *Subdev Media Bus Code Enumerate Flags*
__u32	reserved[7]	Reserved for future extensions. Applications and drivers must set the array to zero.

Table 211: Subdev Media Bus Code Enumerate Flags

V4L2_SUBDEV_MBUS_	0x00000001	The driver allows the application to try to change the default colorspace encoding. The application can ask to configure the colorspace of the subdevice when calling the *VIDIOC_SUBDEV_S_FMT* ioctl with *V4L2_MBUS_FRAMEFMT_SET_CSC* set. See *Media Bus Formats* on how to do this.
V4L2_SUBDEV_MBUS_	0x00000002	The driver allows the application to try to change the default transform function. The application can ask to configure the transform function of the subdevice when calling the *VIDIOC_SUBDEV_S_FMT* ioctl with *V4L2_MBUS_FRAMEFMT_SET_CSC* set. See *Media Bus Formats* on how to do this.
V4L2_SUBDEV_MBUS_	0x00000004	The driver allows the application to try to change the default Y' CbCr encoding. The application can ask to configure the Y' CbCr encoding of the subdevice when calling the *VIDIOC_SUBDEV_S_FMT* ioctl with *V4L2_MBUS_FRAMEFMT_SET_CSC* set. See *Media Bus Formats* on how to do this.
V4L2_SUBDEV_MBUS_	0x00000004	The driver allows the application to try to change the default HSV encoding. The application can ask to configure the HSV encoding of the subdevice when calling the *VIDIOC_SUBDEV_S_FMT* ioctl with *V4L2_MBUS_FRAMEFMT_SET_CSC* set. See *Media Bus Formats* on how to do this.
V4L2_SUBDEV_MBUS_	0x00000008	The driver allows the application to try to change the default quantization. The application can ask to configure the quantization of the subdevice when calling the *VIDIOC_SUBDEV_S_FMT* ioctl with *V4L2_MBUS_FRAMEFMT_SET_CSC* set. See *Media Bus Formats* on how to do this.

Return Value

On success 0 is returned, on error -1 and the `errno` variable is set appropriately. The generic error codes are described at the *Generic Error Codes* chapter.

EINVAL
: The struct *v4l2_subdev_mbus_code_enum* pad references a non-existing pad, or the `index` field is out of bounds.

ioctl VIDIOC_SUBDEV_G_CROP, VIDIOC_SUBDEV_S_CROP

Name

VIDIOC_SUBDEV_G_CROP - VIDIOC_SUBDEV_S_CROP - Get or set the crop rectangle on a subdev pad

Synopsis

VIDIOC_SUBDEV_G_CROP

`int ioctl(int fd, VIDIOC_SUBDEV_G_CROP, struct v4l2_subdev_crop *argp)`

VIDIOC_SUBDEV_S_CROP

`int ioctl(int fd, VIDIOC_SUBDEV_S_CROP, const struct v4l2_subdev_crop *argp)`

Arguments

fd
: File descriptor returned by *open()*.

argp
: Pointer to struct *v4l2_subdev_crop*.

Description

Note: This is an *Obsolete API Elements* interface and may be removed in the future. It is superseded by *the selection API*.

To retrieve the current crop rectangle applications set the `pad` field of a struct *v4l2_subdev_crop* to the desired pad number as reported by the media API and the `which` field to `V4L2_SUBDEV_FORMAT_ACTIVE`. They then call the `VIDIOC_SUBDEV_G_CROP` ioctl with a pointer to this structure. The driver fills the members of the `rect` field or returns `EINVAL` error code if the input arguments are invalid, or if cropping is not supported on the given pad.

To change the current crop rectangle applications set both the pad and which fields and all members of the rect field. They then call the VIDIOC_SUBDEV_S_CROP ioctl with a pointer to this structure. The driver verifies the requested crop rectangle, adjusts it based on the hardware capabilities and configures the device. Upon return the struct *v4l2_subdev_crop* contains the current format as would be returned by a VIDIOC_SUBDEV_G_CROP call.

Applications can query the device capabilities by setting the which to V4L2_SUBDEV_FORMAT_TRY. When set, 'try' crop rectangles are not applied to the device by the driver, but are mangled exactly as active crop rectangles and stored in the sub-device file handle. Two applications querying the same sub-device would thus not interact with each other.

If the subdev device node has been registered in read-only mode, calls to VIDIOC_SUBDEV_S_CROP are only valid if the which field is set to V4L2_SUBDEV_FORMAT_TRY, otherwise an error is returned and the errno variable is set to -EPERM.

Drivers must not return an error solely because the requested crop rectangle doesn't match the device capabilities. They must instead modify the rectangle to match what the hardware can provide. The modified format should be as close as possible to the original request.

type **v4l2_subdev_crop**

Table 212: struct v4l2_subdev_crop

__u32	pad	Pad number as reported by the media framework.
__u32	which	Crop rectangle to get or set, from enum *v4l2_subdev_format_whence*.
struct *v4l2_rect*	rect	Crop rectangle boundaries, in pixels.
__u32	reserved[8]	Reserved for future extensions. Applications and drivers must set the array to zero.

Return Value

On success 0 is returned, on error -1 and the errno variable is set appropriately. The generic error codes are described at the *Generic Error Codes* chapter.

EBUSY
The crop rectangle can't be changed because the pad is currently busy. This can be caused, for instance, by an active video stream on the pad. The ioctl must not be retried without performing another action to fix the problem first. Only returned by VIDIOC_SUBDEV_S_CROP

EINVAL
The struct *v4l2_subdev_crop* pad references a non-existing pad, the which field references a non-existing format, or cropping is not supported on the given subdev pad.

EPERM
The VIDIOC_SUBDEV_S_CROP ioctl has been called on a read-only subdevice

and the which field is set to V4L2_SUBDEV_FORMAT_ACTIVE.

ioctl VIDIOC_SUBDEV_G_FMT, VIDIOC_SUBDEV_S_FMT

Name

VIDIOC_SUBDEV_G_FMT - VIDIOC_SUBDEV_S_FMT - Get or set the data format on a subdev pad

Synopsis

VIDIOC_SUBDEV_G_FMT

`int ioctl(int fd, VIDIOC_SUBDEV_G_FMT, struct v4l2_subdev_format *argp)`

VIDIOC_SUBDEV_S_FMT

`int ioctl(int fd, VIDIOC_SUBDEV_S_FMT, struct v4l2_subdev_format *argp)`

Arguments

fd
File descriptor returned by *open()*.

argp
Pointer to struct *v4l2_subdev_format*.

Description

These ioctls are used to negotiate the frame format at specific subdev pads in the image pipeline.

To retrieve the current format applications set the pad field of a struct *v4l2_subdev_format* to the desired pad number as reported by the media API and the which field to V4L2_SUBDEV_FORMAT_ACTIVE. When they call the VIDIOC_SUBDEV_G_FMT ioctl with a pointer to this structure the driver fills the members of the format field.

To change the current format applications set both the pad and which fields and all members of the format field. When they call the VIDIOC_SUBDEV_S_FMT ioctl with a pointer to this structure the driver verifies the requested format, adjusts it based on the hardware capabilities and configures the device. Upon return the struct *v4l2_subdev_format* contains the current format as would be returned by a VIDIOC_SUBDEV_G_FMT call.

Applications can query the device capabilities by setting the which to V4L2_SUBDEV_FORMAT_TRY. When set, 'try' formats are not applied to the device by the driver, but are changed exactly as active formats and stored in the sub-device file handle. Two applications querying the same sub-device would thus not interact with each other.

For instance, to try a format at the output pad of a sub-device, applications would first set the try format at the sub-device input with the `VIDIOC_SUBDEV_S_FMT` ioctl. They would then either retrieve the default format at the output pad with the `VIDIOC_SUBDEV_G_FMT` ioctl, or set the desired output pad format with the `VIDIOC_SUBDEV_S_FMT` ioctl and check the returned value.

Try formats do not depend on active formats, but can depend on the current links configuration or sub-device controls value. For instance, a low-pass noise filter might crop pixels at the frame boundaries, modifying its output frame size.

If the subdev device node has been registered in read-only mode, calls to `VIDIOC_SUBDEV_S_FMT` are only valid if the `which` field is set to `V4L2_SUBDEV_FORMAT_TRY`, otherwise an error is returned and the errno variable is set to `-EPERM`.

Drivers must not return an error solely because the requested format doesn't match the device capabilities. They must instead modify the format to match what the hardware can provide. The modified format should be as close as possible to the original request.

type **`v4l2_subdev_format`**

Table 213: struct v4l2_subdev_format

__u32	`pad`	Pad number as reported by the media controller API.
__u32	`which`	Format to modified, from enum *v4l2_subdev_format_whence*.
struct *`v4l2_mbus_framefmt`*	`format`	Definition of an image format, see *`v4l2_mbus_framefmt`* for details.
__u32	`reserved[8]`	Reserved for future extensions. Applications and drivers must set the array to zero.

Table 214: enum v4l2_subdev_format_whence

V4L2_SUBDEV_FORMAT_TRY	0	Try formats, used for querying device capabilities.
V4L2_SUBDEV_FORMAT_ACTIVE	1	Active formats, applied to the hardware.

Return Value

On success 0 is returned, on error -1 and the `errno` variable is set appropriately. The generic error codes are described at the *Generic Error Codes* chapter.

EBUSY
: The format can't be changed because the pad is currently busy. This can be caused, for instance, by an active video stream on the pad. The ioctl must not be retried without performing another action to fix the problem first. Only returned by `VIDIOC_SUBDEV_S_FMT`

EINVAL
The struct *v4l2_subdev_format* pad references a non-existing pad, or the which field references a non-existing format.

EPERM
The VIDIOC_SUBDEV_S_FMT ioctl has been called on a read-only subdevice and the which field is set to V4L2_SUBDEV_FORMAT_ACTIVE.

On success 0 is returned, on error -1 and the errno variable is set appropriately. The generic error codes are described at the *Generic Error Codes* chapter.

ioctl VIDIOC_SUBDEV_G_FRAME_INTERVAL, VIDIOC_SUBDEV_S_FRAME_INTERVAL

Name

VIDIOC_SUBDEV_G_FRAME_INTERVAL - VIDIOC_SUBDEV_S_FRAME_INTERVAL - Get or set the frame interval on a subdev pad

Synopsis

VIDIOC_SUBDEV_G_FRAME_INTERVAL

`int ioctl(int fd, VIDIOC_SUBDEV_G_FRAME_INTERVAL, struct v4l2_subdev_frame_interval *argp)`

VIDIOC_SUBDEV_S_FRAME_INTERVAL

`int ioctl(int fd, VIDIOC_SUBDEV_S_FRAME_INTERVAL, struct v4l2_subdev_frame_interval *argp)`

Arguments

fd
File descriptor returned by *open()*.

argp
Pointer to struct *v4l2_subdev_frame_interval*.

Description

These ioctls are used to get and set the frame interval at specific subdev pads in the image pipeline. The frame interval only makes sense for sub-devices that can control the frame period on their own. This includes, for instance, image sensors and TV tuners. Sub-devices that don't support frame intervals must not implement these ioctls.

To retrieve the current frame interval applications set the pad field of a struct *v4l2_subdev_frame_interval* to the desired pad number as reported by the media controller API. When they call the VIDIOC_SUBDEV_G_FRAME_INTERVAL ioctl with a pointer to this structure the driver fills the members of the interval field.

To change the current frame interval applications set both the pad field and all members of the interval field. When they call the VIDIOC_SUBDEV_S_FRAME_INTERVAL ioctl with a pointer to this structure the driver verifies the requested interval, adjusts it based on the hardware capabilities and configures the device. Upon return the struct *v4l2_subdev_frame_interval* contains the current frame interval as would be returned by a VIDIOC_SUBDEV_G_FRAME_INTERVAL call.

Calling VIDIOC_SUBDEV_S_FRAME_INTERVAL on a subdev device node that has been registered in read-only mode is not allowed. An error is returned and the errno variable is set to -EPERM.

Drivers must not return an error solely because the requested interval doesn't match the device capabilities. They must instead modify the interval to match what the hardware can provide. The modified interval should be as close as possible to the original request.

Changing the frame interval shall never change the format. Changing the format, on the other hand, may change the frame interval.

Sub-devices that support the frame interval ioctls should implement them on a single pad only. Their behaviour when supported on multiple pads of the same sub-device is not defined.

type **v4l2_subdev_frame_interval**

Table 215: struct v4l2_subdev_frame_interval

__u32	pad	Pad number as reported by the media controller API.
struct *v4l2_fract*	interval	Period, in seconds, between consecutive video frames.
__u32	reserved[9]	Reserved for future extensions. Applications and drivers must set the array to zero.

Return Value

On success 0 is returned, on error -1 and the errno variable is set appropriately. The generic error codes are described at the *Generic Error Codes* chapter.

EBUSY
: The frame interval can't be changed because the pad is currently busy. This can be caused, for instance, by an active video stream on the pad. The ioctl must not be retried without performing another action to fix the problem first. Only returned by VIDIOC_SUBDEV_S_FRAME_INTERVAL

EINVAL
: The struct *v4l2_subdev_frame_interval* pad references a non-existing pad, or the pad doesn't support frame intervals.

EPERM
: The VIDIOC_SUBDEV_S_FRAME_INTERVAL ioctl has been called on a read-only subdevice.

ioctl VIDIOC_SUBDEV_G_SELECTION, VIDIOC_SUBDEV_S_SELECTION

Name

VIDIOC_SUBDEV_G_SELECTION - VIDIOC_SUBDEV_S_SELECTION - Get or set selection rectangles on a subdev pad

Synopsis

VIDIOC_SUBDEV_G_SELECTION

`int ioctl(int fd, VIDIOC_SUBDEV_G_SELECTION, struct v4l2_subdev_selection *argp)`

VIDIOC_SUBDEV_S_SELECTION

`int ioctl(int fd, VIDIOC_SUBDEV_S_SELECTION, struct v4l2_subdev_selection *argp)`

Arguments

fd
: File descriptor returned by *open()*.

argp
: Pointer to struct *v4l2_subdev_selection*.

Description

The selections are used to configure various image processing functionality performed by the subdevs which affect the image size. This currently includes cropping, scaling and composition.

The selection API replaces *the old subdev crop API*. All the function of the crop API, and more, are supported by the selections API.

See *Sub-device Interface* for more information on how each selection target affects the image processing pipeline inside the subdevice.

If the subdev device node has been registered in read-only mode, calls to `VIDIOC_SUBDEV_S_SELECTION` are only valid if the which field is set to `V4L2_SUBDEV_FORMAT_TRY`, otherwise an error is returned and the errno variable is set to `-EPERM`.

Types of selection targets

There are two types of selection targets: actual and bounds. The actual targets are the targets which configure the hardware. The BOUNDS target will return a rectangle that contain all possible actual rectangles.

Discovering supported features

To discover which targets are supported, the user can perform `VIDIOC_SUBDEV_G_SELECTION` on them. Any unsupported target will return `EINVAL`.

Selection targets and flags are documented in *Common selection definitions*.

type **v4l2_subdev_selection**

Table 216: struct v4l2_subdev_selection

__u32	`which`	Active or try selection, from enum *v4l2_subdev_format_whence*.
__u32	`pad`	Pad number as reported by the media framework.
__u32	`target`	Target selection rectangle. See *Common selection definitions*.
__u32	`flags`	Flags. See *Selection flags*.
struct *v4l2_rect*	`r`	Selection rectangle, in pixels.
__u32	`reserved[8]`	Reserved for future extensions. Applications and drivers must set the array to zero.

Return Value

On success 0 is returned, on error -1 and the `errno` variable is set appropriately. The generic error codes are described at the *Generic Error Codes* chapter.

EBUSY
: The selection rectangle can't be changed because the pad is currently busy. This can be caused, for instance, by an active video stream on the pad. The ioctl must not be retried without performing another action to fix the problem first. Only returned by `VIDIOC_SUBDEV_S_SELECTION`

EINVAL
: The struct *v4l2_subdev_selection* `pad` references a non-existing pad, the `which` field references a non-existing format, or the selection target is not supported on the given subdev pad.

EPERM
: The `VIDIOC_SUBDEV_S_SELECTION` ioctl has been called on a read-only subdevice and the `which` field is set to `V4L2_SUBDEV_FORMAT_ACTIVE`.

ioctl VIDIOC_SUBDEV_QUERYCAP

Name

VIDIOC_SUBDEV_QUERYCAP - Query sub-device capabilities

Synopsis

VIDIOC_SUBDEV_QUERYCAP

`int ioctl(int fd, VIDIOC_SUBDEV_QUERYCAP, struct v4l2_subdev_capability *argp)`

Arguments

fd
File descriptor returned by *open()*.

argp
Pointer to struct *v4l2_subdev_capability*.

Description

All V4L2 sub-devices support the VIDIOC_SUBDEV_QUERYCAP ioctl. It is used to identify kernel devices compatible with this specification and to obtain information about driver and hardware capabilities. The ioctl takes a pointer to a struct *v4l2_subdev_capability* which is filled by the driver. When the driver is not compatible with this specification the ioctl returns ENOTTY error code.

type **v4l2_subdev_capability**

Table 217: struct v4l2_subdev_capability

__u32	version	Version number of the driver. The version reported is provided by the V4L2 subsystem following the kernel numbering scheme. However, it may not always return the same version as the kernel if, for example, a stable or distribution-modified kernel uses the V4L2 stack from a newer kernel. The version number is formatted using the KERNEL_VERSION() macro:

```
#define KERNEL_VERSION(a,b,c) (((a) << 16) + ((b) << 8) + (c))
__u32 version = KERNEL_VERSION(0, 8, 1);
printf ("Version: %u.%u.%u\\n",
(version >> 16) & 0xFF, (version >> 8) & 0xFF, version & 0xFF);
```

__u32	capabilitie	Sub-device capabilities of the opened device, see *Sub-Device Capabilities Flags*.
__u32	reserved[14]	Reserved for future extensions. Set to 0 by the V4L2 core.

Table 218: Sub-Device Capabilities Flags

V4L2_SUBDEV_CAP_RO_SUBD	0x00000001	The sub-device device node is registered in read-only mode. Access to the sub-device ioctls that modify the device state is restricted. Refer to each individual subdevice ioctl documentation for a description of which restrictions apply to a read-only sub-device.

Return Value

On success 0 is returned, on error -1 and the `errno` variable is set appropriately. The generic error codes are described at the *Generic Error Codes* chapter.

ENOTTY
The device node is not a V4L2 sub-device.

ioctl VIDIOC_SUBSCRIBE_EVENT, VIDIOC_UNSUBSCRIBE_EVENT

Name

VIDIOC_SUBSCRIBE_EVENT - VIDIOC_UNSUBSCRIBE_EVENT - Subscribe or unsubscribe event

Synopsis

VIDIOC_SUBSCRIBE_EVENT

```
int ioctl(int fd, VIDIOC_SUBSCRIBE_EVENT, struct
v4l2_event_subscription *argp)
```

VIDIOC_UNSUBSCRIBE_EVENT

```
int ioctl(int fd, VIDIOC_UNSUBSCRIBE_EVENT, struct
v4l2_event_subscription *argp)
```

Arguments

fd
File descriptor returned by *open()*.

argp
Pointer to struct *v4l2_event_subscription*.

Description

Subscribe or unsubscribe V4L2 event. Subscribed events are dequeued by using the *ioctl VIDIOC_DQEVENT* ioctl.

type **v4l2_event_subscription**

Table 219: struct v4l2_event_subscription

__u32	type	Type of the event, see *Event Types*. **Note:** V4L2_EVENT_ALL can be used with *VIDIOC_UNSUBSCRIBE_EVENT* for unsubscribing all events at once.
__u32	id	ID of the event source. If there is no ID associated with the event source, then set this to 0. Whether or not an event needs an ID depends on the event type.
__u32	flags	Event flags, see *Event Flags*.
__u32	reserved[5]	Reserved for future extensions. Drivers and applications must set the array to zero.

Table 220: Event Flags

V4L2_EVENT_SUB_FL_SEND_INITIAL	0x0001	When this event is subscribed an initial event will be sent containing the current status. This only makes sense for events that are triggered by a status change such as V4L2_EVENT_CTRL. Other events will ignore this flag.
V4L2_EVENT_SUB_FL_ALLOW_FEEDBA	0x0002	If set, then events directly caused by an ioctl will also be sent to the filehandle that called that ioctl. For example, changing a control using *VIDIOC_S_CTRL* will cause a V4L2_EVENT_CTRL to be sent back to that same filehandle. Normally such events are suppressed to prevent feedback loops where an application changes a control to a one value and then another, and then receives an event telling it that that control has changed to the first value. Since it can't tell whether that event was caused by another application or by the *VIDIOC_S_CTRL* call it is hard to decide whether to set the control to the value in the event, or ignore it. Think carefully when you set this flag so you won't get into situations like that.

Return Value

On success 0 is returned, on error -1 and the errno variable is set appropriately. The generic error codes are described at the *Generic Error Codes* chapter.

V4L2 mmap()

Name

v4l2-mmap - Map device memory into application address space

Synopsis

```
#include <unistd.h>
#include <sys/mman.h>
```

void ***mmap**(void *start, size_t length, int prot, int flags, int fd, off_t offset)

Arguments

start
: Map the buffer to this address in the application' s address space. When the MAP_FIXED flag is specified, start must be a multiple of the pagesize and mmap will fail when the specified address cannot be used. Use of this option is discouraged; applications should just specify a NULL pointer here.

length
: Length of the memory area to map. This must be the same value as returned by the driver in the struct *v4l2_buffer* length field for the single-planar API, and the same value as returned by the driver in the struct *v4l2_plane* length field for the multi-planar API.

prot
: The prot argument describes the desired memory protection. Regardless of the device type and the direction of data exchange it should be set to PROT_READ | PROT_WRITE, permitting read and write access to image buffers. Drivers should support at least this combination of flags.

Note:

1. The Linux videobuf kernel module, which is used by some drivers supports only PROT_READ | PROT_WRITE. When the driver does not support the desired protection, the *mmap()* function fails.
2. Device memory accesses (e. g. the memory on a graphics card with video capturing hardware) may incur a performance penalty compared to main memory accesses, or reads may be significantly slower than writes or vice versa. Other I/O methods may be more efficient in such case.

flags

The `flags` parameter specifies the type of the mapped object, mapping options and whether modifications made to the mapped copy of the page are private to the process or are to be shared with other references.

`MAP_FIXED` requests that the driver selects no other address than the one specified. If the specified address cannot be used, *mmap()* will fail. If `MAP_FIXED` is specified, `start` must be a multiple of the pagesize. Use of this option is discouraged.

One of the `MAP_SHARED` or `MAP_PRIVATE` flags must be set. `MAP_SHARED` allows applications to share the mapped memory with other (e. g. child-) processes.

Note: The Linux `videobuf` module which is used by some drivers supports only `MAP_SHARED`. `MAP_PRIVATE` requests copy-on-write semantics. V4L2 applications should not set the `MAP_PRIVATE`, `MAP_DENYWRITE`, `MAP_EXECUTABLE` or `MAP_ANON` flags.

---

fd

File descriptor returned by *open()*.

offset

Offset of the buffer in device memory. This must be the same value as returned by the driver in the struct *v4l2_buffer* `m` union `offset` field for the single-planar API, and the same value as returned by the driver in the struct *v4l2_plane* `m` union `mem_offset` field for the multi-planar API.

Description

The *mmap()* function asks to map `length` bytes starting at `offset` in the memory of the device specified by `fd` into the application address space, preferably at address `start`. This latter address is a hint only, and is usually specified as 0.

Suitable length and offset parameters are queried with the *ioctl VIDIOC_QUERYBUF* ioctl. Buffers must be allocated with the *ioctl VIDIOC_REQBUFS* ioctl before they can be queried.

To unmap buffers the *munmap()* function is used.

Return Value

On success *mmap()* returns a pointer to the mapped buffer. On error `MAP_FAILED` (-1) is returned, and the `errno` variable is set appropriately. Possible error codes are:

EBADF

`fd` is not a valid file descriptor.

EACCES

`fd` is not open for reading and writing.

EINVAL
The `start` or `length` or `offset` are not suitable. (E. g. they are too large, or not aligned on a `PAGESIZE` boundary.)

The `flags` or `prot` value is not supported.

No buffers have been allocated with the *ioctl VIDIOC_REQBUFS* ioctl.

ENOMEM
Not enough physical or virtual memory was available to complete the request.

V4L2 munmap()

Name

v4l2-munmap - Unmap device memory

Synopsis

```
#include <unistd.h>
#include <sys/mman.h>
```

int **munmap**(void *start, size_t length)

Arguments

start
Address of the mapped buffer as returned by the *mmap()* function.

length
Length of the mapped buffer. This must be the same value as given to *mmap()* and returned by the driver in the struct *v4l2_buffer* `length` field for the single-planar API and in the struct *v4l2_plane* `length` field for the multi-planar API.

Description

Unmaps a previously with the *mmap()* function mapped buffer and frees it, if possible.

Return Value

On success *munmap()* returns 0, on failure -1 and the `errno` variable is set appropriately:

EINVAL
The `start` or `length` is incorrect, or no buffers have been mapped yet.

V4L2 open()

Name

v4l2-open - Open a V4L2 device

Synopsis

```
#include <fcntl.h>
```

int **open**(const char *device_name, int flags)

Arguments

device_name
Device to be opened.

flags
Open flags. Access mode must be `O_RDWR`. This is just a technicality, input devices still support only reading and output devices only writing.

When the `O_NONBLOCK` flag is given, the *read()* function and the *VIDIOC_DQBUF* ioctl will return the `EAGAIN` error code when no data is available or no buffer is in the driver outgoing queue, otherwise these functions block until data becomes available. All V4L2 drivers exchanging data with applications must support the `O_NONBLOCK` flag.

Other flags have no effect.

Description

To open a V4L2 device applications call *open()* with the desired device name. This function has no side effects; all data format parameters, current input or output, control values or other properties remain unchanged. At the first *open()* call after loading the driver they will be reset to default values, drivers are never in an undefined state.

Return Value

On success *open()* returns the new file descriptor. On error -1 is returned, and the `errno` variable is set appropriately. Possible error codes are:

EACCES
The caller has no permission to access the device.

EBUSY
The driver does not support multiple opens and the device is already in use.

ENXIO
No device corresponding to this device special file exists.

ENOMEM
Not enough kernel memory was available to complete the request.

EMFILE
The process already has the maximum number of files open.

ENFILE
The limit on the total number of files open on the system has been reached.

V4L2 poll()

Name

v4l2-poll - Wait for some event on a file descriptor

Synopsis

```
#include <sys/poll.h>
```

int **poll**(struct pollfd *ufds, unsigned int nfds, int timeout)

Arguments

Description

With the *poll()* function applications can suspend execution until the driver has captured data or is ready to accept data for output.

When streaming I/O has been negotiated this function waits until a buffer has been filled by the capture device and can be dequeued with the *VIDIOC_DQBUF* ioctl. For output devices this function waits until the device is ready to accept a new buffer to be queued up with the *VIDIOC_QBUF* ioctl for display. When buffers are already in the outgoing queue of the driver (capture) or the incoming queue isn't full (display) the function returns immediately.

On success *poll()* returns the number of file descriptors that have been selected (that is, file descriptors for which the `revents` field of the respective `struct` pollfd structure is non-zero). Capture devices set the POLLIN and POLLRDNORM flags in the revents field, output devices the POLLOUT and POLLWRNORM flags. When the function timed out it returns a value of zero, on failure it returns -1 and the errno variable is set appropriately. When the application did not call *VIDIOC_STREAMON* the *poll()* function succeeds, but sets the POLLERR flag in the revents field. When the application has called *VIDIOC_STREAMON* for a capture device but hasn't yet called *VIDIOC_QBUF*, the *poll()* function succeeds and sets the POLLERR flag in the revents field. For output devices this same situation will cause *poll()* to succeed as well, but it sets the POLLOUT and POLLWRNORM flags in the revents field.

If an event occurred (see *ioctl VIDIOC_DQEVENT*) then POLLPRI will be set in the revents field and *poll()* will return.

When use of the *read()* function has been negotiated and the driver does not capture yet, the *poll()* function starts capturing. When that fails it returns a POLLERR as above. Otherwise it waits until data has been captured and can be read. When the driver captures continuously (as opposed to, for example, still images) the function may return immediately.

When use of the *write()* function has been negotiated and the driver does not stream yet, the *poll()* function starts streaming. When that fails it returns a POLLERR as above. Otherwise it waits until the driver is ready for a non-blocking *write()* call.

If the caller is only interested in events (just POLLPRI is set in the events field), then *poll()* will *not* start streaming if the driver does not stream yet. This makes it possible to just poll for events and not for buffers.

All drivers implementing the *read()* or *write()* function or streaming I/O must also support the *poll()* function.

For more details see the *poll()* manual page.

Return Value

On success, *poll()* returns the number structures which have non-zero revents fields, or zero if the call timed out. On error -1 is returned, and the errno variable is set appropriately:

EBADF
: One or more of the ufds members specify an invalid file descriptor.

EBUSY
: The driver does not support multiple read or write streams and the device is already in use.

EFAULT
: ufds references an inaccessible memory area.

EINTR
: The call was interrupted by a signal.

EINVAL
: The nfds value exceeds the RLIMIT_NOFILE value. Use getrlimit() to obtain this value.

V4L2 read()

Name

v4l2-read - Read from a V4L2 device

Synopsis

```
#include <unistd.h>
```

ssize_t **read**(int fd, void *buf, size_t count)

Arguments

fd
File descriptor returned by *open()*.

buf
Buffer to be filled

count
Max number of bytes to read

Description

read() attempts to read up to `count` bytes from file descriptor `fd` into the buffer starting at `buf`. The layout of the data in the buffer is discussed in the respective device interface section, see ##. If `count` is zero, *read()* returns zero and has no other results. If `count` is greater than `SSIZE_MAX`, the result is unspecified. Regardless of the `count` value each *read()* call will provide at most one frame (two fields) worth of data.

By default *read()* blocks until data becomes available. When the `O_NONBLOCK` flag was given to the *open()* function it returns immediately with an `EAGAIN` error code when no data is available. The *select()* or *poll()* functions can always be used to suspend execution until data becomes available. All drivers supporting the *read()* function must also support *select()* and *poll()*.

Drivers can implement read functionality in different ways, using a single or multiple buffers and discarding the oldest or newest frames once the internal buffers are filled.

read() never returns a "snapshot" of a buffer being filled. Using a single buffer the driver will stop capturing when the application starts reading the buffer until the read is finished. Thus only the period of the vertical blanking interval is available for reading, or the capture rate must fall below the nominal frame rate of the video standard.

The behavior of *read()* when called during the active picture period or the vertical blanking separating the top and bottom field depends on the discarding policy. A driver discarding the oldest frames keeps capturing into an internal buffer, continuously overwriting the previously, not read frame, and returns the frame being received at the time of the *read()* call as soon as it is complete.

A driver discarding the newest frames stops capturing until the next *read()* call. The frame being received at *read()* time is discarded, returning the following frame instead. Again this implies a reduction of the capture rate to one half or less of the nominal frame rate. An example of this model is the video read mode of the bttv driver, initiating a DMA to user memory when *read()* is called and returning when the DMA finished.

In the multiple buffer model drivers maintain a ring of internal buffers, automatically advancing to the next free buffer. This allows continuous capturing when the application can empty the buffers fast enough. Again, the behavior when the driver runs out of free buffers depends on the discarding policy.

Applications can get and set the number of buffers used internally by the driver with the *VIDIOC_G_PARM* and *VIDIOC_S_PARM* ioctls. They are optional, however. The discarding policy is not reported and cannot be changed. For minimum requirements see *Interfaces*.

Return Value

On success, the number of bytes read is returned. It is not an error if this number is smaller than the number of bytes requested, or the amount of data required for one frame. This may happen for example because *read()* was interrupted by a signal. On error, -1 is returned, and the `errno` variable is set appropriately. In this case the next read will start at the beginning of a new frame. Possible error codes are:

EAGAIN
: Non-blocking I/O has been selected using O_NONBLOCK and no data was immediately available for reading.

EBADF
: `fd` is not a valid file descriptor or is not open for reading, or the process already has the maximum number of files open.

EBUSY
: The driver does not support multiple read streams and the device is already in use.

EFAULT
: `buf` references an inaccessible memory area.

EINTR
: The call was interrupted by a signal before any data was read.

EIO
: I/O error. This indicates some hardware problem or a failure to communicate with a remote device (USB camera etc.).

EINVAL
: The *read()* function is not supported by this driver, not on this device, or generally not on this type of device.

V4L2 select()

Name

v4l2-select - Synchronous I/O multiplexing

Synopsis

```
#include <sys/time.h>
#include <sys/types.h>
#include <unistd.h>
```

int **select**(int nfds, fd_set *readfds, fd_set *writefds, fd_set *exceptfds, struct timeval *timeout)

Arguments

nfds
: The highest-numbered file descriptor in any of the three sets, plus 1.

readfds
: File descriptions to be watched if a read() call won't block.

writefds
: File descriptions to be watched if a write() won't block.

exceptfds
: File descriptions to be watched for V4L2 events.

timeout
: Maximum time to wait.

Description

With the *select()* function applications can suspend execution until the driver has captured data or is ready to accept data for output.

When streaming I/O has been negotiated this function waits until a buffer has been filled or displayed and can be dequeued with the *VIDIOC_DQBUF* ioctl. When buffers are already in the outgoing queue of the driver the function returns immediately.

On success *select()* returns the total number of bits set in `fd_set`. When the function timed out it returns a value of zero. On failure it returns -1 and the `errno` variable is set appropriately. When the application did not call *ioctl VIDIOC_QBUF, VIDIOC_DQBUF* or *ioctl VIDIOC_STREAMON, VIDIOC_STREAMOFF* yet the *select()* function succeeds, setting the bit of the file descriptor in `readfds` or `writefds`, but subsequent *VIDIOC_DQBUF* calls will fail.[1]

[1] The Linux kernel implements *select()* like the *poll()* function, but *select()* cannot return a `POLLERR`.

When use of the *read()* function has been negotiated and the driver does not capture yet, the *select()* function starts capturing. When that fails, *select()* returns successful and a subsequent *read()* call, which also attempts to start capturing, will return an appropriate error code. When the driver captures continuously (as opposed to, for example, still images) and data is already available the *select()* function returns immediately.

When use of the *write()* function has been negotiated the *select()* function just waits until the driver is ready for a non-blocking *write()* call.

All drivers implementing the *read()* or *write()* function or streaming I/O must also support the *select()* function.

For more details see the *select()* manual page.

Return Value

On success, *select()* returns the number of descriptors contained in the three returned descriptor sets, which will be zero if the timeout expired. On error -1 is returned, and the `errno` variable is set appropriately; the sets and `timeout` are undefined. Possible error codes are:

EBADF
: One or more of the file descriptor sets specified a file descriptor that is not open.

EBUSY
: The driver does not support multiple read or write streams and the device is already in use.

EFAULT
: The `readfds`, `writefds`, `exceptfds` or `timeout` pointer references an inaccessible memory area.

EINTR
: The call was interrupted by a signal.

EINVAL
: The `nfds` argument is less than zero or greater than `FD_SETSIZE`.

V4L2 write()

Name

v4l2-write - Write to a V4L2 device

Synopsis

```
#include <unistd.h>
```

ssize_t **write**(int fd, void *buf, size_t count)

Arguments

fd
File descriptor returned by *open()*.

buf
Buffer with data to be written

count
Number of bytes at the buffer

Description

write() writes up to `count` bytes to the device referenced by the file descriptor `fd` from the buffer starting at `buf`. When the hardware outputs are not active yet, this function enables them. When `count` is zero, *write()* returns 0 without any other effect.

When the application does not provide more data in time, the previous video frame, raw VBI image, sliced VPS or WSS data is displayed again. Sliced Teletext or Closed Caption data is not repeated, the driver inserts a blank line instead.

Return Value

On success, the number of bytes written are returned. Zero indicates nothing was written. On error, -1 is returned, and the `errno` variable is set appropriately. In this case the next write will start at the beginning of a new frame. Possible error codes are:

EAGAIN
Non-blocking I/O has been selected using the *O_NONBLOCK* flag and no buffer space was available to write the data immediately.

EBADF
`fd` is not a valid file descriptor or is not open for writing.

EBUSY
The driver does not support multiple write streams and the device is already in use.

EFAULT
`buf` references an inaccessible memory area.

EINTR
The call was interrupted by a signal before any data was written.

EIO
I/O error. This indicates some hardware problem.

EINVAL
The *write()* function is not supported by this driver, not on this device, or generally not on this type of device.

8.2.8 Common definitions for V4L2 and V4L2 subdev interfaces

Common selection definitions

While the *V4L2 selection API* and *V4L2 subdev selection APIs* are very similar, there's one fundamental difference between the two. On sub-device API, the selection rectangle refers to the media bus format, and is bound to a sub-device's pad. On the V4L2 interface the selection rectangles refer to the in-memory pixel format.

This section defines the common definitions of the selection interfaces on the two APIs.

Selection targets

The precise meaning of the selection targets may be dependent on which of the two interfaces they are used.

Table 221: Selection target definitions

Target name	id	Definition	Valid for V4L2	Valid for V4L2 subdev
V4L2_SEL_TGT_CROP	0x0000	Crop rectangle. Defines the cropped area.	Yes	Yes
V4L2_SEL_TGT_CROP_DEFAULT	0x0001	Suggested cropping rectangle that covers the "whole picture". This includes only active pixels and excludes other non-active pixels such as black pixels.	Yes	Yes
V4L2_SEL_TGT_CROP_BOUNDS	0x0002	Bounds of the crop rectangle. All valid crop rectangles fit inside the crop bounds rectangle.	Yes	Yes
V4L2_SEL_TGT_NATIVE_SIZE	0x0003	The native size of the device, e.g. a sensor's pixel array. `left` and `top` fields are zero for this target.	Yes	Yes
V4L2_SEL_TGT_COMPOSE	0x0100	Compose rectangle. Used to configure scaling and composition.	Yes	Yes
V4L2_SEL_TGT_COMPOSE_DEFAU	0x0101	Suggested composition rectangle that covers the "whole picture".	Yes	No
V4L2_SEL_TGT_COMPOSE_BOUNDS	0x0102	Bounds of the compose rectangle. All valid compose rectangles fit inside the compose bounds rectangle.	Yes	Yes
V4L2_SEL_TGT_COMPOSE_PADDE	0x0103	The active area and all padding pixels that are inserted or modified by hardware.	Yes	No

Selection flags

Table 222: Selection flag definitions

Flag name	id	Definition	Valid for V4L2	Valid for V4L2 subdev
V4L2_SEL_FLAG_GE	(1 << 0)	Suggest the driver it should choose greater or equal rectangle (in size) than was requested. Albeit the driver may choose a lesser size, it will only do so due to hardware limitations. Without this flag (and V4L2_SEL_FLAG_LE) the behaviour is to choose the closest possible rectangle.	Yes	Yes
V4L2_SEL_FLAG_LE	(1 << 1)	Suggest the driver it should choose lesser or equal rectangle (in size) than was requested. Albeit the driver may choose a greater size, it will only do so due to hardware limitations.	Yes	Yes
V4L2_SEL_FLAG_KEEP_CONF	(1 << 2)	The configuration must not be propagated to any further processing steps. If this flag is not given, the configuration is propagated inside the subdevice to all further processing steps.	No	Yes

8.2.9 Video For Linux Two Header File

videodev2.h

```
/* SPDX-License-Identifier: ((GPL-2.0+ WITH Linux-syscall-note) OR␣
↪BSD-3-Clause) */
/*
 *  Video for Linux Two header file
 *
 *  Copyright (C) 1999-2012 the contributors
 *
 *  This program is free software; you can redistribute it and/or␣
↪modify
 *  it under the terms of the GNU General Public License as␣
↪published by
 *  the Free Software Foundation; either version 2 of the License,␣
↪or
 *  (at your option) any later version.
 *
 *  This program is distributed in the hope that it will be useful,
```

```
 *  but WITHOUT ANY WARRANTY; without even the implied warranty of
 *  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 *  GNU General Public License for more details.
 *
 *  Alternatively you can redistribute this file under the terms of␣
↪the
 *  BSD license as stated below:
 *
 *  Redistribution and use in source and binary forms, with or␣
↪without
 *  modification, are permitted provided that the following␣
↪conditions
 *  are met:
 *  1. Redistributions of source code must retain the above␣
↪copyright
 *     notice, this list of conditions and the following disclaimer.
 *  2. Redistributions in binary form must reproduce the above␣
↪copyright
 *     notice, this list of conditions and the following disclaimer␣
↪in
 *     the documentation and/or other materials provided with the
 *     distribution.
 *  3. The names of its contributors may not be used to endorse or␣
↪promote
 *     products derived from this software without specific prior␣
↪written
 *     permission.
 *
 *  THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND␣
↪CONTRIBUTORS
 *  "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT␣
↪NOT
 *  LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND␣
↪FITNESS FOR
 *  A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE␣
↪COPYRIGHT
 *  OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,␣
↪INCIDENTAL,
 *  SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT␣
↪NOT LIMITED
 *  TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE,␣
↪DATA, OR
 *  PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY␣
↪THEORY OF
 *  LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT␣
↪(INCLUDING
 *  NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF␣
↪THIS
 *  SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
 *
 *      Header file for v4l or V4L2 drivers and applications
```

```
 * with public API.
 * All kernel-specific stuff were moved to media/v4l2-dev.h, so
 * no #if __KERNEL tests are allowed here
 *
 *      See https://linuxtv.org for more info
 *
 *      Author: Bill Dirks <bill@thedirks.org>
 *              Justin Schoeman
 *              Hans Verkuil <hverkuil@xs4all.nl>
 *              et al.
 */
#ifndef _UAPI__LINUX_VIDEODEV2_H
#define _UAPI__LINUX_VIDEODEV2_H

#ifndef __KERNEL__
#include <sys/time.h>
#endif
#include <linux/compiler.h>
#include <linux/ioctl.h>
#include <linux/types.h>
#include <linux/v4l2-common.h>
#include <linux/v4l2-controls.h>

/*
 * Common stuff for both V4L1 and V4L2
 * Moved from videodev.h
 */
#define VIDEO_MAX_FRAME               32
#define VIDEO_MAX_PLANES               8

/*
 *      M I S C E L L A N E O U S
 */

/*  Four-character-code (FOURCC) */
#define v4l2_fourcc(a, b, c, d)\
        ((__u32)(a) | ((__u32)(b) << 8) | ((__u32)(c) << 16) | ((__
↪u32)(d) << 24))
#define v4l2_fourcc_be(a, b, c, d)      (v4l2_fourcc(a, b, c, d) |␣
↪(1U << 31))

/*
 *      E N U M S
 */
enum v4l2_field {
        V4L2_FIELD_ANY           = 0, /* driver can choose from␣
↪none,
                                         top, bottom, interlaced
                                         depending on whatever it␣
↪thinks
                                         is approximate ... */
```

```
        V4L2_FIELD_NONE          = 1, /* this device has no fields .
↪.. */
        V4L2_FIELD_TOP           = 2, /* top field only */
        V4L2_FIELD_BOTTOM        = 3, /* bottom field only */
        V4L2_FIELD_INTERLACED    = 4, /* both fields interlaced */
        V4L2_FIELD_SEQ_TB        = 5, /* both fields sequential␣
↪into one
                                        buffer, top-bottom order */
        V4L2_FIELD_SEQ_BT        = 6, /* same as above + bottom-top␣
↪order */
        V4L2_FIELD_ALTERNATE     = 7, /* both fields alternating␣
↪into
                                        separate buffers */
        V4L2_FIELD_INTERLACED_TB = 8, /* both fields interlaced,␣
↪top field
                                        first and the top field is
                                        transmitted first */
        V4L2_FIELD_INTERLACED_BT = 9, /* both fields interlaced,␣
↪top field
                                        first and the bottom field␣
↪is
                                        transmitted first */
};
#define V4L2_FIELD_HAS_TOP(field)       \
        ((field) == V4L2_FIELD_TOP      ||\
         (field) == V4L2_FIELD_INTERLACED ||\
         (field) == V4L2_FIELD_INTERLACED_TB ||\
         (field) == V4L2_FIELD_INTERLACED_BT ||\
         (field) == V4L2_FIELD_SEQ_TB   ||\
         (field) == V4L2_FIELD_SEQ_BT)
#define V4L2_FIELD_HAS_BOTTOM(field)    \
        ((field) == V4L2_FIELD_BOTTOM   ||\
         (field) == V4L2_FIELD_INTERLACED ||\
         (field) == V4L2_FIELD_INTERLACED_TB ||\
         (field) == V4L2_FIELD_INTERLACED_BT ||\
         (field) == V4L2_FIELD_SEQ_TB   ||\
         (field) == V4L2_FIELD_SEQ_BT)
#define V4L2_FIELD_HAS_BOTH(field)      \
        ((field) == V4L2_FIELD_INTERLACED ||\
         (field) == V4L2_FIELD_INTERLACED_TB ||\
         (field) == V4L2_FIELD_INTERLACED_BT ||\
         (field) == V4L2_FIELD_SEQ_TB ||\
         (field) == V4L2_FIELD_SEQ_BT)
#define V4L2_FIELD_HAS_T_OR_B(field)    \
        ((field) == V4L2_FIELD_BOTTOM ||\
         (field) == V4L2_FIELD_TOP ||\
         (field) == V4L2_FIELD_ALTERNATE)
#define V4L2_FIELD_IS_INTERLACED(field) \
        ((field) == V4L2_FIELD_INTERLACED ||\
         (field) == V4L2_FIELD_INTERLACED_TB ||\
         (field) == V4L2_FIELD_INTERLACED_BT)
```

```
#define V4L2_FIELD_IS_SEQUENTIAL(field) \
        ((field) == V4L2_FIELD_SEQ_TB ||\
         (field) == V4L2_FIELD_SEQ_BT)

enum v4l2_buf_type {
        V4L2_BUF_TYPE_VIDEO_CAPTURE        = 1,
        V4L2_BUF_TYPE_VIDEO_OUTPUT         = 2,
        V4L2_BUF_TYPE_VIDEO_OVERLAY        = 3,
        V4L2_BUF_TYPE_VBI_CAPTURE          = 4,
        V4L2_BUF_TYPE_VBI_OUTPUT           = 5,
        V4L2_BUF_TYPE_SLICED_VBI_CAPTURE   = 6,
        V4L2_BUF_TYPE_SLICED_VBI_OUTPUT    = 7,
        V4L2_BUF_TYPE_VIDEO_OUTPUT_OVERLAY = 8,
        V4L2_BUF_TYPE_VIDEO_CAPTURE_MPLANE = 9,
        V4L2_BUF_TYPE_VIDEO_OUTPUT_MPLANE  = 10,
        V4L2_BUF_TYPE_SDR_CAPTURE          = 11,
        V4L2_BUF_TYPE_SDR_OUTPUT           = 12,
        V4L2_BUF_TYPE_META_CAPTURE         = 13,
        V4L2_BUF_TYPE_META_OUTPUT          = 14,
        /* Deprecated, do not use */
        V4L2_BUF_TYPE_PRIVATE              = 0x80,
};

#define V4L2_TYPE_IS_MULTIPLANAR(type)                  \
        ((type) == V4L2_BUF_TYPE_VIDEO_CAPTURE_MPLANE   \
         || (type) == V4L2_BUF_TYPE_VIDEO_OUTPUT_MPLANE)

#define V4L2_TYPE_IS_OUTPUT(type)                               \
        ((type) == V4L2_BUF_TYPE_VIDEO_OUTPUT                   \
         || (type) == V4L2_BUF_TYPE_VIDEO_OUTPUT_MPLANE         \
         || (type) == V4L2_BUF_TYPE_VIDEO_OVERLAY               \
         || (type) == V4L2_BUF_TYPE_VIDEO_OUTPUT_OVERLAY        \
         || (type) == V4L2_BUF_TYPE_VBI_OUTPUT                  \
         || (type) == V4L2_BUF_TYPE_SLICED_VBI_OUTPUT           \
         || (type) == V4L2_BUF_TYPE_SDR_OUTPUT                  \
         || (type) == V4L2_BUF_TYPE_META_OUTPUT)

#define V4L2_TYPE_IS_CAPTURE(type) (!V4L2_TYPE_IS_OUTPUT(type))

enum v4l2_tuner_type {
        V4L2_TUNER_RADIO             = 1,
        V4L2_TUNER_ANALOG_TV         = 2,
        V4L2_TUNER_DIGITAL_TV        = 3,
        V4L2_TUNER_SDR               = 4,
        V4L2_TUNER_RF                = 5,
};

/* Deprecated, do not use */
#define V4L2_TUNER_ADC  V4L2_TUNER_SDR

enum v4l2_memory {
```

```
        V4L2_MEMORY_MMAP             = 1,
        V4L2_MEMORY_USERPTR          = 2,
        V4L2_MEMORY_OVERLAY          = 3,
        V4L2_MEMORY_DMABUF           = 4,
};

/* see also http://vektor.theorem.ca/graphics/ycbcr/ */
enum v4l2_colorspace {
        /*
         * Default colorspace, i.e. let the driver figure it out.
         * Can only be used with video capture.
         */
        V4L2_COLORSPACE_DEFAULT       = 0,

        /* SMPTE 170M: used for broadcast NTSC/PAL SDTV */
        V4L2_COLORSPACE_SMPTE170M     = 1,

        /* Obsolete pre-1998 SMPTE 240M HDTV standard, superseded␣
↪by Rec 709 */
        V4L2_COLORSPACE_SMPTE240M     = 2,

        /* Rec.709: used for HDTV */
        V4L2_COLORSPACE_REC709        = 3,

        /*
         * Deprecated, do not use. No driver will ever return this.␣
↪This was
         * based on a misunderstanding of the bt878 datasheet.
         */
        V4L2_COLORSPACE_BT878         = 4,

        /*
         * NTSC 1953 colorspace. This only makes sense when dealing␣
↪with
         * really, really old NTSC recordings. Superseded by SMPTE␣
↪170M.
         */
        V4L2_COLORSPACE_470_SYSTEM_M  = 5,

        /*
         * EBU Tech 3213 PAL/SECAM colorspace. This only makes␣
↪sense when
         * dealing with really old PAL/SECAM recordings. Superseded␣
↪by
         * SMPTE 170M.
         */
        V4L2_COLORSPACE_470_SYSTEM_BG = 6,

        /*
         * Effectively shorthand for V4L2_COLORSPACE_SRGB, V4L2_
↪YCBCR_ENC_601
```

```
         * and V4L2_QUANTIZATION_FULL_RANGE. To be used for␣
↪(Motion-)JPEG.
         */
        V4L2_COLORSPACE_JPEG          = 7,

        /* For RGB colorspaces such as produces by most webcams. */
        V4L2_COLORSPACE_SRGB          = 8,

        /* opRGB colorspace */
        V4L2_COLORSPACE_OPRGB         = 9,

        /* BT.2020 colorspace, used for UHDTV. */
        V4L2_COLORSPACE_BT2020        = 10,

        /* Raw colorspace: for RAW unprocessed images */
        V4L2_COLORSPACE_RAW           = 11,

        /* DCI-P3 colorspace, used by cinema projectors */
        V4L2_COLORSPACE_DCI_P3        = 12,
};

/*
 * Determine how COLORSPACE_DEFAULT should map to a proper␣
↪colorspace.
 * This depends on whether this is a SDTV image (use SMPTE 170M), an
 * HDTV image (use Rec. 709), or something else (use sRGB).
 */
#define V4L2_MAP_COLORSPACE_DEFAULT(is_sdtv, is_hdtv) \
        ((is_sdtv) ? V4L2_COLORSPACE_SMPTE170M : \
         ((is_hdtv) ? V4L2_COLORSPACE_REC709 : V4L2_COLORSPACE_
↪SRGB))

enum v4l2_xfer_func {
        /*
         * Mapping of V4L2_XFER_FUNC_DEFAULT to actual transfer␣
↪functions
         * for the various colorspaces:
         *
         * V4L2_COLORSPACE_SMPTE170M, V4L2_COLORSPACE_470_SYSTEM_M,
         * V4L2_COLORSPACE_470_SYSTEM_BG, V4L2_COLORSPACE_REC709 and
         * V4L2_COLORSPACE_BT2020: V4L2_XFER_FUNC_709
         *
         * V4L2_COLORSPACE_SRGB, V4L2_COLORSPACE_JPEG: V4L2_XFER_
↪FUNC_SRGB
         *
         * V4L2_COLORSPACE_OPRGB: V4L2_XFER_FUNC_OPRGB
         *
         * V4L2_COLORSPACE_SMPTE240M: V4L2_XFER_FUNC_SMPTE240M
         *
         * V4L2_COLORSPACE_RAW: V4L2_XFER_FUNC_NONE
         *
```

```
         * V4L2_COLORSPACE_DCI_P3: V4L2_XFER_FUNC_DCI_P3
         */
        V4L2_XFER_FUNC_DEFAULT     = 0,
        V4L2_XFER_FUNC_709         = 1,
        V4L2_XFER_FUNC_SRGB        = 2,
        V4L2_XFER_FUNC_OPRGB       = 3,
        V4L2_XFER_FUNC_SMPTE240M   = 4,
        V4L2_XFER_FUNC_NONE        = 5,
        V4L2_XFER_FUNC_DCI_P3      = 6,
        V4L2_XFER_FUNC_SMPTE2084   = 7,
};

/*
 * Determine how XFER_FUNC_DEFAULT should map to a proper transfer␣
↪function.
 * This depends on the colorspace.
 */
#define V4L2_MAP_XFER_FUNC_DEFAULT(colsp) \
        ((colsp) == V4L2_COLORSPACE_OPRGB ? V4L2_XFER_FUNC_OPRGB : \
         ((colsp) == V4L2_COLORSPACE_SMPTE240M ? V4L2_XFER_FUNC_
↪SMPTE240M : \
          ((colsp) == V4L2_COLORSPACE_DCI_P3 ? V4L2_XFER_FUNC_DCI_
↪P3 : \
           ((colsp) == V4L2_COLORSPACE_RAW ? V4L2_XFER_FUNC_NONE : \
            ((colsp) == V4L2_COLORSPACE_SRGB || (colsp) == V4L2_
↪COLORSPACE_JPEG ? \
             V4L2_XFER_FUNC_SRGB : V4L2_XFER_FUNC_709)))))

enum v4l2_ycbcr_encoding {
        /*
         * Mapping of V4L2_YCBCR_ENC_DEFAULT to actual encodings␣
↪for the
         * various colorspaces:
         *
         * V4L2_COLORSPACE_SMPTE170M, V4L2_COLORSPACE_470_SYSTEM_M,
         * V4L2_COLORSPACE_470_SYSTEM_BG, V4L2_COLORSPACE_SRGB,
         * V4L2_COLORSPACE_OPRGB and V4L2_COLORSPACE_JPEG: V4L2_
↪YCBCR_ENC_601
         *
         * V4L2_COLORSPACE_REC709 and V4L2_COLORSPACE_DCI_P3: V4L2_
↪YCBCR_ENC_709
         *
         * V4L2_COLORSPACE_BT2020: V4L2_YCBCR_ENC_BT2020
         *
         * V4L2_COLORSPACE_SMPTE240M: V4L2_YCBCR_ENC_SMPTE240M
         */
        V4L2_YCBCR_ENC_DEFAULT        = 0,

        /* ITU-R 601 -- SDTV */
        V4L2_YCBCR_ENC_601            = 1,
```

```
        /* Rec. 709 -- HDTV */
        V4L2_YCBCR_ENC_709            = 2,

        /* ITU-R 601/EN 61966-2-4 Extended Gamut -- SDTV */
        V4L2_YCBCR_ENC_XV601          = 3,

        /* Rec. 709/EN 61966-2-4 Extended Gamut -- HDTV */
        V4L2_YCBCR_ENC_XV709          = 4,

#ifndef __KERNEL__
        /*
         * sYCC (Y'CbCr encoding of sRGB), identical to ENC_601. It␣
↪was added
         * originally due to a misunderstanding of the sYCC␣
↪standard. It should
         * not be used, instead use V4L2_YCBCR_ENC_601.
         */
        V4L2_YCBCR_ENC_SYCC           = 5,
#endif

        /* BT.2020 Non-constant Luminance Y'CbCr */
        V4L2_YCBCR_ENC_BT2020         = 6,

        /* BT.2020 Constant Luminance Y'CbcCrc */
        V4L2_YCBCR_ENC_BT2020_CONST_LUM = 7,

        /* SMPTE 240M -- Obsolete HDTV */
        V4L2_YCBCR_ENC_SMPTE240M      = 8,
};

/*
 * enum v4l2_hsv_encoding values should not collide with the ones␣
↪from
 * enum v4l2_ycbcr_encoding.
 */
enum v4l2_hsv_encoding {

        /* Hue mapped to 0 - 179 */
        V4L2_HSV_ENC_180              = 128,

        /* Hue mapped to 0-255 */
        V4L2_HSV_ENC_256              = 129,
};

/*
 * Determine how YCBCR_ENC_DEFAULT should map to a proper Y'CbCr␣
↪encoding.
 * This depends on the colorspace.
 */
#define V4L2_MAP_YCBCR_ENC_DEFAULT(colsp) \
        (((colsp) == V4L2_COLORSPACE_REC709 || \
```

```
          (colsp) == V4L2_COLORSPACE_DCI_P3) ? V4L2_YCBCR_ENC_709 :␣
↪\
         ((colsp) == V4L2_COLORSPACE_BT2020 ? V4L2_YCBCR_ENC_BT2020␣
↪: \
          ((colsp) == V4L2_COLORSPACE_SMPTE240M ? V4L2_YCBCR_ENC_
↪SMPTE240M : \
           V4L2_YCBCR_ENC_601)))

enum v4l2_quantization {
        /*
         * The default for R'G'B' quantization is always full range.
         * For Y'CbCr the quantization is always limited range,␣
↪except
         * for COLORSPACE_JPEG: this is full range.
         */
        V4L2_QUANTIZATION_DEFAULT     = 0,
        V4L2_QUANTIZATION_FULL_RANGE  = 1,
        V4L2_QUANTIZATION_LIM_RANGE   = 2,
};

/*
 * Determine how QUANTIZATION_DEFAULT should map to a proper␣
↪quantization.
 * This depends on whether the image is RGB or not, the colorspace.
 * The Y'CbCr encoding is not used anymore, but is still there for␣
↪backwards
 * compatibility.
 */
#define V4L2_MAP_QUANTIZATION_DEFAULT(is_rgb_or_hsv, colsp, ycbcr_
↪enc) \
        (((is_rgb_or_hsv) || (colsp) == V4L2_COLORSPACE_JPEG) ? \
         V4L2_QUANTIZATION_FULL_RANGE : V4L2_QUANTIZATION_LIM_RANGE)

/*
 * Deprecated names for opRGB colorspace (IEC 61966-2-5)
 *
 * WARNING: Please don't use these deprecated defines in your code,␣
↪as
 * there is a chance we have to remove them in the future.
 */
#ifndef __KERNEL__
#define V4L2_COLORSPACE_ADOBERGB V4L2_COLORSPACE_OPRGB
#define V4L2_XFER_FUNC_ADOBERGB  V4L2_XFER_FUNC_OPRGB
#endif

enum v4l2_priority {
        V4L2_PRIORITY_UNSET       = 0,  /* not initialized */
        V4L2_PRIORITY_BACKGROUND  = 1,
        V4L2_PRIORITY_INTERACTIVE = 2,
        V4L2_PRIORITY_RECORD      = 3,
        V4L2_PRIORITY_DEFAULT     = V4L2_PRIORITY_INTERACTIVE,
```

```
};

struct v4l2_rect {
        __s32   left;
        __s32   top;
        __u32   width;
        __u32   height;
};

struct v4l2_fract {
        __u32   numerator;
        __u32   denominator;
};

struct v4l2_area {
        __u32   width;
        __u32   height;
};

/**
  * struct v4l2_capability - Describes V4L2 device caps returned by␣
 ↪VIDIOC_QUERYCAP
  *
  * @driver:       name of the driver module (e.g. "bttv")
  * @card:         name of the card (e.g. "Hauppauge WinTV")
  * @bus_info:     name of the bus (e.g. "PCI:" + pci_name(pci_dev)␣
 ↪)
  * @version:      KERNEL_VERSION
  * @capabilities: capabilities of the physical device as a whole
  * @device_caps:  capabilities accessed via this particular device␣
 ↪(node)
  * @reserved:     reserved fields for future extensions
  */
struct v4l2_capability {
        __u8    driver[16];
        __u8    card[32];
        __u8    bus_info[32];
        __u32   version;
        __u32   capabilities;
        __u32   device_caps;
        __u32   reserved[3];
};

/* Values for 'capabilities' field */
#define V4L2_CAP_VIDEO_CAPTURE          0x00000001  /* Is a video␣
 ↪capture device */
#define V4L2_CAP_VIDEO_OUTPUT           0x00000002  /* Is a video␣
 ↪output device */
#define V4L2_CAP_VIDEO_OVERLAY          0x00000004  /* Can do video␣
 ↪overlay */
#define V4L2_CAP_VBI_CAPTURE            0x00000010  /* Is a raw VBI␣
```

```
↪capture device */
#define V4L2_CAP_VBI_OUTPUT             0x00000020  /* Is a raw VBI␣
↪output device */
#define V4L2_CAP_SLICED_VBI_CAPTURE     0x00000040  /* Is a sliced␣
↪VBI capture device */
#define V4L2_CAP_SLICED_VBI_OUTPUT      0x00000080  /* Is a sliced␣
↪VBI output device */
#define V4L2_CAP_RDS_CAPTURE            0x00000100  /* RDS data␣
↪capture */
#define V4L2_CAP_VIDEO_OUTPUT_OVERLAY   0x00000200  /* Can do video␣
↪output overlay */
#define V4L2_CAP_HW_FREQ_SEEK           0x00000400  /* Can do␣
↪hardware frequency seek  */
#define V4L2_CAP_RDS_OUTPUT             0x00000800  /* Is an RDS␣
↪encoder */

/* Is a video capture device that supports multiplanar formats */
#define V4L2_CAP_VIDEO_CAPTURE_MPLANE   0x00001000
/* Is a video output device that supports multiplanar formats */
#define V4L2_CAP_VIDEO_OUTPUT_MPLANE    0x00002000
/* Is a video mem-to-mem device that supports multiplanar formats */
#define V4L2_CAP_VIDEO_M2M_MPLANE       0x00004000
/* Is a video mem-to-mem device */
#define V4L2_CAP_VIDEO_M2M              0x00008000

#define V4L2_CAP_TUNER                  0x00010000  /* has a tuner␣
↪*/
#define V4L2_CAP_AUDIO                  0x00020000  /* has audio␣
↪support */
#define V4L2_CAP_RADIO                  0x00040000  /* is a radio␣
↪device */
#define V4L2_CAP_MODULATOR              0x00080000  /* has a␣
↪modulator */

#define V4L2_CAP_SDR_CAPTURE            0x00100000  /* Is a SDR␣
↪capture device */
#define V4L2_CAP_EXT_PIX_FORMAT         0x00200000  /* Supports the␣
↪extended pixel format */
#define V4L2_CAP_SDR_OUTPUT             0x00400000  /* Is a SDR␣
↪output device */
#define V4L2_CAP_META_CAPTURE           0x00800000  /* Is a␣
↪metadata capture device */

#define V4L2_CAP_READWRITE              0x01000000  /* read/write␣
↪systemcalls */
#define V4L2_CAP_ASYNCIO                0x02000000  /* async I/O */
#define V4L2_CAP_STREAMING              0x04000000  /* streaming I/
↪O ioctls */
#define V4L2_CAP_META_OUTPUT            0x08000000  /* Is a␣
↪metadata output device */
```

```
#define V4L2_CAP_TOUCH                  0x10000000  /* Is a touch␣
↪device */

#define V4L2_CAP_IO_MC                  0x20000000  /* Is input/
↪output controlled by the media controller */

#define V4L2_CAP_DEVICE_CAPS            0x80000000  /* sets device␣
↪capabilities field */

/*
 *      V I D E O   I M A G E   F O R M A T
 */
struct v4l2_pix_format {
        __u32                   width;
        __u32                   height;
        __u32                   pixelformat;
        __u32                   field;          /* enum v4l2_field␣
↪*/
        __u32                   bytesperline;   /* for padding,␣
↪zero if unused */
        __u32                   sizeimage;
        __u32                   colorspace;     /* enum v4l2_
↪colorspace */
        __u32                   priv;           /* private data,␣
↪depends on pixelformat */
        __u32                   flags;          /* format flags␣
↪(V4L2_PIX_FMT_FLAG_*) */
        union {
                /* enum v4l2_ycbcr_encoding */
                __u32                   ycbcr_enc;
                /* enum v4l2_hsv_encoding */
                __u32                   hsv_enc;
        };
        __u32                   quantization;   /* enum v4l2_
↪quantization */
        __u32                   xfer_func;      /* enum v4l2_xfer_
↪func */
};

/*      Pixel format         FOURCC                          depth ␣
↪Description  */

/* RGB formats */
#define V4L2_PIX_FMT_RGB332  v4l2_fourcc('R', 'G', 'B', '1') /*  8 ␣
↪RGB-3-3-2     */
#define V4L2_PIX_FMT_RGB444  v4l2_fourcc('R', '4', '4', '4') /* 16 ␣
↪xxxxrrrr ggggbbbb */
#define V4L2_PIX_FMT_ARGB444 v4l2_fourcc('A', 'R', '1', '2') /* 16 ␣
↪aaaarrrr ggggbbbb */
#define V4L2_PIX_FMT_XRGB444 v4l2_fourcc('X', 'R', '1', '2') /* 16 ␣
↪xxxxrrrr ggggbbbb */
```

```
#define V4L2_PIX_FMT_RGBA444 v4l2_fourcc('R', 'A', '1', '2') /* 16 ␣
↪rrrrgggg bbbbaaaa */
#define V4L2_PIX_FMT_RGBX444 v4l2_fourcc('R', 'X', '1', '2') /* 16 ␣
↪rrrrgggg bbbbxxxx */
#define V4L2_PIX_FMT_ABGR444 v4l2_fourcc('A', 'B', '1', '2') /* 16 ␣
↪aaaabbbb ggggrrrr */
#define V4L2_PIX_FMT_XBGR444 v4l2_fourcc('X', 'B', '1', '2') /* 16 ␣
↪xxxxbbbb ggggrrrr */

/*
 * Originally this had 'BA12' as fourcc, but this clashed with the␣
↪older
 * V4L2_PIX_FMT_SGRBG12 which inexplicably used that same fourcc.
 * So use 'GA12' instead for V4L2_PIX_FMT_BGRA444.
 */
#define V4L2_PIX_FMT_BGRA444 v4l2_fourcc('G', 'A', '1', '2') /* 16 ␣
↪bbbbgggg rrrraaaa */
#define V4L2_PIX_FMT_BGRX444 v4l2_fourcc('B', 'X', '1', '2') /* 16 ␣
↪bbbbgggg rrrrxxxx */
#define V4L2_PIX_FMT_RGB555  v4l2_fourcc('R', 'G', 'B', 'O') /* 16 ␣
↪RGB-5-5-5     */
#define V4L2_PIX_FMT_ARGB555 v4l2_fourcc('A', 'R', '1', '5') /* 16 ␣
↪ARGB-1-5-5-5  */
#define V4L2_PIX_FMT_XRGB555 v4l2_fourcc('X', 'R', '1', '5') /* 16 ␣
↪XRGB-1-5-5-5  */
#define V4L2_PIX_FMT_RGBA555 v4l2_fourcc('R', 'A', '1', '5') /* 16 ␣
↪RGBA-5-5-5-1  */
#define V4L2_PIX_FMT_RGBX555 v4l2_fourcc('R', 'X', '1', '5') /* 16 ␣
↪RGBX-5-5-5-1  */
#define V4L2_PIX_FMT_ABGR555 v4l2_fourcc('A', 'B', '1', '5') /* 16 ␣
↪ABGR-1-5-5-5  */
#define V4L2_PIX_FMT_XBGR555 v4l2_fourcc('X', 'B', '1', '5') /* 16 ␣
↪XBGR-1-5-5-5  */
#define V4L2_PIX_FMT_BGRA555 v4l2_fourcc('B', 'A', '1', '5') /* 16 ␣
↪BGRA-5-5-5-1  */
#define V4L2_PIX_FMT_BGRX555 v4l2_fourcc('B', 'X', '1', '5') /* 16 ␣
↪BGRX-5-5-5-1  */
#define V4L2_PIX_FMT_RGB565  v4l2_fourcc('R', 'G', 'B', 'P') /* 16 ␣
↪RGB-5-6-5     */
#define V4L2_PIX_FMT_RGB555X v4l2_fourcc('R', 'G', 'B', 'Q') /* 16 ␣
↪RGB-5-5-5 BE  */
#define V4L2_PIX_FMT_ARGB555X v4l2_fourcc_be('A', 'R', '1', '5') /*␣
↪16  ARGB-5-5-5 BE */
#define V4L2_PIX_FMT_XRGB555X v4l2_fourcc_be('X', 'R', '1', '5') /*␣
↪16  XRGB-5-5-5 BE */
#define V4L2_PIX_FMT_RGB565X v4l2_fourcc('R', 'G', 'B', 'R') /* 16 ␣
↪RGB-5-6-5 BE  */
#define V4L2_PIX_FMT_BGR666  v4l2_fourcc('B', 'G', 'R', 'H') /* 18 ␣
↪BGR-6-6-6     */
#define V4L2_PIX_FMT_BGR24   v4l2_fourcc('B', 'G', 'R', '3') /* 24 ␣
↪BGR-8-8-8     */
```

```
#define V4L2_PIX_FMT_RGB24   v4l2_fourcc('R', 'G', 'B', '3') /* 24 ␣
↪RGB-8-8-8     */
#define V4L2_PIX_FMT_BGR32   v4l2_fourcc('B', 'G', 'R', '4') /* 32 ␣
↪BGR-8-8-8-8   */
#define V4L2_PIX_FMT_ABGR32  v4l2_fourcc('A', 'R', '2', '4') /* 32 ␣
↪BGRA-8-8-8-8  */
#define V4L2_PIX_FMT_XBGR32  v4l2_fourcc('X', 'R', '2', '4') /* 32 ␣
↪BGRX-8-8-8-8  */
#define V4L2_PIX_FMT_BGRA32  v4l2_fourcc('R', 'A', '2', '4') /* 32 ␣
↪ABGR-8-8-8-8  */
#define V4L2_PIX_FMT_BGRX32  v4l2_fourcc('R', 'X', '2', '4') /* 32 ␣
↪XBGR-8-8-8-8  */
#define V4L2_PIX_FMT_RGB32   v4l2_fourcc('R', 'G', 'B', '4') /* 32 ␣
↪RGB-8-8-8-8   */
#define V4L2_PIX_FMT_RGBA32  v4l2_fourcc('A', 'B', '2', '4') /* 32 ␣
↪RGBA-8-8-8-8  */
#define V4L2_PIX_FMT_RGBX32  v4l2_fourcc('X', 'B', '2', '4') /* 32 ␣
↪RGBX-8-8-8-8  */
#define V4L2_PIX_FMT_ARGB32  v4l2_fourcc('B', 'A', '2', '4') /* 32 ␣
↪ARGB-8-8-8-8  */
#define V4L2_PIX_FMT_XRGB32  v4l2_fourcc('B', 'X', '2', '4') /* 32 ␣
↪XRGB-8-8-8-8  */

/* Grey formats */
#define V4L2_PIX_FMT_GREY    v4l2_fourcc('G', 'R', 'E', 'Y') /*  8 ␣
↪Greyscale     */
#define V4L2_PIX_FMT_Y4      v4l2_fourcc('Y', '0', '4', ' ') /*  4 ␣
↪Greyscale     */
#define V4L2_PIX_FMT_Y6      v4l2_fourcc('Y', '0', '6', ' ') /*  6 ␣
↪Greyscale     */
#define V4L2_PIX_FMT_Y10     v4l2_fourcc('Y', '1', '0', ' ') /* 10 ␣
↪Greyscale     */
#define V4L2_PIX_FMT_Y12     v4l2_fourcc('Y', '1', '2', ' ') /* 12 ␣
↪Greyscale     */
#define V4L2_PIX_FMT_Y14     v4l2_fourcc('Y', '1', '4', ' ') /* 14 ␣
↪Greyscale     */
#define V4L2_PIX_FMT_Y16     v4l2_fourcc('Y', '1', '6', ' ') /* 16 ␣
↪Greyscale     */
#define V4L2_PIX_FMT_Y16_BE  v4l2_fourcc_be('Y', '1', '6', ' ') /*␣
↪16  Greyscale BE  */

/* Grey bit-packed formats */
#define V4L2_PIX_FMT_Y10BPACK    v4l2_fourcc('Y', '1', '0', 'B') /*␣
↪10  Greyscale bit-packed */
#define V4L2_PIX_FMT_Y10P    v4l2_fourcc('Y', '1', '0', 'P') /* 10 ␣
↪Greyscale, MIPI RAW10 packed */

/* Palette formats */
#define V4L2_PIX_FMT_PAL8    v4l2_fourcc('P', 'A', 'L', '8') /*  8 ␣
↪8-bit palette */
```

```
/* Chrominance formats */
#define V4L2_PIX_FMT_UV8      v4l2_fourcc('U', 'V', '8', ' ') /*  8 ␣
↪UV 4:4 */

/* Luminance+Chrominance formats */
#define V4L2_PIX_FMT_YUYV     v4l2_fourcc('Y', 'U', 'Y', 'V') /* 16 ␣
↪YUV 4:2:2     */
#define V4L2_PIX_FMT_YYUV     v4l2_fourcc('Y', 'Y', 'U', 'V') /* 16 ␣
↪YUV 4:2:2     */
#define V4L2_PIX_FMT_YVYU     v4l2_fourcc('Y', 'V', 'Y', 'U') /* 16␣
↪YVU 4:2:2 */
#define V4L2_PIX_FMT_UYVY     v4l2_fourcc('U', 'Y', 'V', 'Y') /* 16 ␣
↪YUV 4:2:2     */
#define V4L2_PIX_FMT_VYUY     v4l2_fourcc('V', 'Y', 'U', 'Y') /* 16 ␣
↪YUV 4:2:2     */
#define V4L2_PIX_FMT_Y41P     v4l2_fourcc('Y', '4', '1', 'P') /* 12 ␣
↪YUV 4:1:1     */
#define V4L2_PIX_FMT_YUV444   v4l2_fourcc('Y', '4', '4', '4') /* 16 ␣
↪xxxxyyyy uuuuvvvv */
#define V4L2_PIX_FMT_YUV555   v4l2_fourcc('Y', 'U', 'V', 'O') /* 16 ␣
↪YUV-5-5-5     */
#define V4L2_PIX_FMT_YUV565   v4l2_fourcc('Y', 'U', 'V', 'P') /* 16 ␣
↪YUV-5-6-5     */
#define V4L2_PIX_FMT_YUV32    v4l2_fourcc('Y', 'U', 'V', '4') /* 32 ␣
↪YUV-8-8-8-8   */
#define V4L2_PIX_FMT_AYUV32   v4l2_fourcc('A', 'Y', 'U', 'V') /* 32 ␣
↪AYUV-8-8-8-8  */
#define V4L2_PIX_FMT_XYUV32   v4l2_fourcc('X', 'Y', 'U', 'V') /* 32 ␣
↪XYUV-8-8-8-8  */
#define V4L2_PIX_FMT_VUYA32   v4l2_fourcc('V', 'U', 'Y', 'A') /* 32 ␣
↪VUYA-8-8-8-8  */
#define V4L2_PIX_FMT_VUYX32   v4l2_fourcc('V', 'U', 'Y', 'X') /* 32 ␣
↪VUYX-8-8-8-8  */
#define V4L2_PIX_FMT_HI240    v4l2_fourcc('H', 'I', '2', '4') /*  8 ␣
↪8-bit color   */
#define V4L2_PIX_FMT_HM12     v4l2_fourcc('H', 'M', '1', '2') /*  8 ␣
↪YUV 4:2:0 16x16 macroblocks */
#define V4L2_PIX_FMT_M420     v4l2_fourcc('M', '4', '2', '0') /* 12 ␣
↪YUV 4:2:0 2 lines y, 1 line uv interleaved */

/* two planes -- one Y, one Cr + Cb interleaved  */
#define V4L2_PIX_FMT_NV12     v4l2_fourcc('N', 'V', '1', '2') /* 12 ␣
↪Y/CbCr 4:2:0  */
#define V4L2_PIX_FMT_NV21     v4l2_fourcc('N', 'V', '2', '1') /* 12 ␣
↪Y/CrCb 4:2:0  */
#define V4L2_PIX_FMT_NV16     v4l2_fourcc('N', 'V', '1', '6') /* 16 ␣
↪Y/CbCr 4:2:2  */
#define V4L2_PIX_FMT_NV61     v4l2_fourcc('N', 'V', '6', '1') /* 16 ␣
↪Y/CrCb 4:2:2  */
#define V4L2_PIX_FMT_NV24     v4l2_fourcc('N', 'V', '2', '4') /* 24 ␣
↪Y/CbCr 4:4:4  */
```

```
#define V4L2_PIX_FMT_NV42    v4l2_fourcc('N', 'V', '4', '2') /* 24 ␣
↪Y/CrCb 4:4:4  */

/* two non contiguous planes - one Y, one Cr + Cb interleaved  */
#define V4L2_PIX_FMT_NV12M   v4l2_fourcc('N', 'M', '1', '2') /* 12 ␣
↪Y/CbCr 4:2:0  */
#define V4L2_PIX_FMT_NV21M   v4l2_fourcc('N', 'M', '2', '1') /* 21 ␣
↪Y/CrCb 4:2:0  */
#define V4L2_PIX_FMT_NV16M   v4l2_fourcc('N', 'M', '1', '6') /* 16 ␣
↪Y/CbCr 4:2:2  */
#define V4L2_PIX_FMT_NV61M   v4l2_fourcc('N', 'M', '6', '1') /* 16 ␣
↪Y/CrCb 4:2:2  */
#define V4L2_PIX_FMT_NV12MT  v4l2_fourcc('T', 'M', '1', '2') /* 12 ␣
↪Y/CbCr 4:2:0 64x32 macroblocks */
#define V4L2_PIX_FMT_NV12MT_16X16 v4l2_fourcc('V', 'M', '1', '2') /
↪* 12  Y/CbCr 4:2:0 16x16 macroblocks */

/* three planes - Y Cb, Cr */
#define V4L2_PIX_FMT_YUV410  v4l2_fourcc('Y', 'U', 'V', '9') /*  9 ␣
↪YUV 4:1:0     */
#define V4L2_PIX_FMT_YVU410  v4l2_fourcc('Y', 'V', 'U', '9') /*  9 ␣
↪YVU 4:1:0     */
#define V4L2_PIX_FMT_YUV411P v4l2_fourcc('4', '1', '1', 'P') /* 12 ␣
↪YVU411 planar */
#define V4L2_PIX_FMT_YUV420  v4l2_fourcc('Y', 'U', '1', '2') /* 12 ␣
↪YUV 4:2:0     */
#define V4L2_PIX_FMT_YVU420  v4l2_fourcc('Y', 'V', '1', '2') /* 12 ␣
↪YVU 4:2:0     */
#define V4L2_PIX_FMT_YUV422P v4l2_fourcc('4', '2', '2', 'P') /* 16 ␣
↪YVU422 planar */

/* three non contiguous planes - Y, Cb, Cr */
#define V4L2_PIX_FMT_YUV420M v4l2_fourcc('Y', 'M', '1', '2') /* 12 ␣
↪YUV420 planar */
#define V4L2_PIX_FMT_YVU420M v4l2_fourcc('Y', 'M', '2', '1') /* 12 ␣
↪YVU420 planar */
#define V4L2_PIX_FMT_YUV422M v4l2_fourcc('Y', 'M', '1', '6') /* 16 ␣
↪YUV422 planar */
#define V4L2_PIX_FMT_YVU422M v4l2_fourcc('Y', 'M', '6', '1') /* 16 ␣
↪YVU422 planar */
#define V4L2_PIX_FMT_YUV444M v4l2_fourcc('Y', 'M', '2', '4') /* 24 ␣
↪YUV444 planar */
#define V4L2_PIX_FMT_YVU444M v4l2_fourcc('Y', 'M', '4', '2') /* 24 ␣
↪YVU444 planar */

/* Bayer formats - see http://www.siliconimaging.com/RGB%20Bayer.
↪htm */
#define V4L2_PIX_FMT_SBGGR8  v4l2_fourcc('B', 'A', '8', '1') /*  8 ␣
↪BGBG.. GRGR.. */
#define V4L2_PIX_FMT_SGBRG8  v4l2_fourcc('G', 'B', 'R', 'G') /*  8 ␣
↪GBGB.. RGRG.. */
```

```
#define V4L2_PIX_FMT_SGRBG8  v4l2_fourcc('G', 'R', 'B', 'G') /*  8 ␣
↪GRGR.. BGBG.. */
#define V4L2_PIX_FMT_SRGGB8  v4l2_fourcc('R', 'G', 'G', 'B') /*  8 ␣
↪RGRG.. GBGB.. */
#define V4L2_PIX_FMT_SBGGR10 v4l2_fourcc('B', 'G', '1', '0') /* 10 ␣
↪BGBG.. GRGR.. */
#define V4L2_PIX_FMT_SGBRG10 v4l2_fourcc('G', 'B', '1', '0') /* 10 ␣
↪GBGB.. RGRG.. */
#define V4L2_PIX_FMT_SGRBG10 v4l2_fourcc('B', 'A', '1', '0') /* 10 ␣
↪GRGR.. BGBG.. */
#define V4L2_PIX_FMT_SRGGB10 v4l2_fourcc('R', 'G', '1', '0') /* 10 ␣
↪RGRG.. GBGB.. */
        /* 10bit raw bayer packed, 5 bytes for every 4 pixels */
#define V4L2_PIX_FMT_SBGGR10P v4l2_fourcc('p', 'B', 'A', 'A')
#define V4L2_PIX_FMT_SGBRG10P v4l2_fourcc('p', 'G', 'A', 'A')
#define V4L2_PIX_FMT_SGRBG10P v4l2_fourcc('p', 'g', 'A', 'A')
#define V4L2_PIX_FMT_SRGGB10P v4l2_fourcc('p', 'R', 'A', 'A')
        /* 10bit raw bayer a-law compressed to 8 bits */
#define V4L2_PIX_FMT_SBGGR10ALAW8 v4l2_fourcc('a', 'B', 'A', '8')
#define V4L2_PIX_FMT_SGBRG10ALAW8 v4l2_fourcc('a', 'G', 'A', '8')
#define V4L2_PIX_FMT_SGRBG10ALAW8 v4l2_fourcc('a', 'g', 'A', '8')
#define V4L2_PIX_FMT_SRGGB10ALAW8 v4l2_fourcc('a', 'R', 'A', '8')
        /* 10bit raw bayer DPCM compressed to 8 bits */
#define V4L2_PIX_FMT_SBGGR10DPCM8 v4l2_fourcc('b', 'B', 'A', '8')
#define V4L2_PIX_FMT_SGBRG10DPCM8 v4l2_fourcc('b', 'G', 'A', '8')
#define V4L2_PIX_FMT_SGRBG10DPCM8 v4l2_fourcc('B', 'D', '1', '0')
#define V4L2_PIX_FMT_SRGGB10DPCM8 v4l2_fourcc('b', 'R', 'A', '8')
#define V4L2_PIX_FMT_SBGGR12 v4l2_fourcc('B', 'G', '1', '2') /* 12 ␣
↪BGBG.. GRGR.. */
#define V4L2_PIX_FMT_SGBRG12 v4l2_fourcc('G', 'B', '1', '2') /* 12 ␣
↪GBGB.. RGRG.. */
#define V4L2_PIX_FMT_SGRBG12 v4l2_fourcc('B', 'A', '1', '2') /* 12 ␣
↪GRGR.. BGBG.. */
#define V4L2_PIX_FMT_SRGGB12 v4l2_fourcc('R', 'G', '1', '2') /* 12 ␣
↪RGRG.. GBGB.. */
        /* 12bit raw bayer packed, 6 bytes for every 4 pixels */
#define V4L2_PIX_FMT_SBGGR12P v4l2_fourcc('p', 'B', 'C', 'C')
#define V4L2_PIX_FMT_SGBRG12P v4l2_fourcc('p', 'G', 'C', 'C')
#define V4L2_PIX_FMT_SGRBG12P v4l2_fourcc('p', 'g', 'C', 'C')
#define V4L2_PIX_FMT_SRGGB12P v4l2_fourcc('p', 'R', 'C', 'C')
#define V4L2_PIX_FMT_SBGGR14 v4l2_fourcc('B', 'G', '1', '4') /* 14 ␣
↪BGBG.. GRGR.. */
#define V4L2_PIX_FMT_SGBRG14 v4l2_fourcc('G', 'B', '1', '4') /* 14 ␣
↪GBGB.. RGRG.. */
#define V4L2_PIX_FMT_SGRBG14 v4l2_fourcc('G', 'R', '1', '4') /* 14 ␣
↪GRGR.. BGBG.. */
#define V4L2_PIX_FMT_SRGGB14 v4l2_fourcc('R', 'G', '1', '4') /* 14 ␣
↪RGRG.. GBGB.. */
        /* 14bit raw bayer packed, 7 bytes for every 4 pixels */
#define V4L2_PIX_FMT_SBGGR14P v4l2_fourcc('p', 'B', 'E', 'E')
#define V4L2_PIX_FMT_SGBRG14P v4l2_fourcc('p', 'G', 'E', 'E')
```

```
#define V4L2_PIX_FMT_SGRBG14P v4l2_fourcc('p', 'g', 'E', 'E')
#define V4L2_PIX_FMT_SRGGB14P v4l2_fourcc('p', 'R', 'E', 'E')
#define V4L2_PIX_FMT_SBGGR16 v4l2_fourcc('B', 'Y', 'R', '2') /* 16 ␣
↪BGBG.. GRGR.. */
#define V4L2_PIX_FMT_SGBRG16 v4l2_fourcc('G', 'B', '1', '6') /* 16 ␣
↪GBGB.. RGRG.. */
#define V4L2_PIX_FMT_SGRBG16 v4l2_fourcc('G', 'R', '1', '6') /* 16 ␣
↪GRGR.. BGBG.. */
#define V4L2_PIX_FMT_SRGGB16 v4l2_fourcc('R', 'G', '1', '6') /* 16 ␣
↪RGRG.. GBGB.. */

/* HSV formats */
#define V4L2_PIX_FMT_HSV24 v4l2_fourcc('H', 'S', 'V', '3')
#define V4L2_PIX_FMT_HSV32 v4l2_fourcc('H', 'S', 'V', '4')

/* compressed formats */
#define V4L2_PIX_FMT_MJPEG    v4l2_fourcc('M', 'J', 'P', 'G') /*␣
↪Motion-JPEG   */
#define V4L2_PIX_FMT_JPEG     v4l2_fourcc('J', 'P', 'E', 'G') /*␣
↪JFIF JPEG     */
#define V4L2_PIX_FMT_DV       v4l2_fourcc('d', 'v', 's', 'd') /*␣
↪1394          */
#define V4L2_PIX_FMT_MPEG     v4l2_fourcc('M', 'P', 'E', 'G') /*␣
↪MPEG-1/2/4 Multiplexed */
#define V4L2_PIX_FMT_H264     v4l2_fourcc('H', '2', '6', '4') /*␣
↪H264 with start codes */
#define V4L2_PIX_FMT_H264_NO_SC v4l2_fourcc('A', 'V', 'C', '1') /*␣
↪H264 without start codes */
#define V4L2_PIX_FMT_H264_MVC v4l2_fourcc('M', '2', '6', '4') /*␣
↪H264 MVC */
#define V4L2_PIX_FMT_H263     v4l2_fourcc('H', '2', '6', '3') /*␣
↪H263          */
#define V4L2_PIX_FMT_MPEG1    v4l2_fourcc('M', 'P', 'G', '1') /*␣
↪MPEG-1 ES     */
#define V4L2_PIX_FMT_MPEG2    v4l2_fourcc('M', 'P', 'G', '2') /*␣
↪MPEG-2 ES     */
#define V4L2_PIX_FMT_MPEG2_SLICE v4l2_fourcc('M', 'G', '2', 'S') /*␣
↪MPEG-2 parsed slice data */
#define V4L2_PIX_FMT_MPEG4    v4l2_fourcc('M', 'P', 'G', '4') /*␣
↪MPEG-4 part 2 ES */
#define V4L2_PIX_FMT_XVID     v4l2_fourcc('X', 'V', 'I', 'D') /*␣
↪Xvid           */
#define V4L2_PIX_FMT_VC1_ANNEX_G v4l2_fourcc('V', 'C', '1', 'G') /*␣
↪SMPTE 421M Annex G compliant stream */
#define V4L2_PIX_FMT_VC1_ANNEX_L v4l2_fourcc('V', 'C', '1', 'L') /*␣
↪SMPTE 421M Annex L compliant stream */
#define V4L2_PIX_FMT_VP8      v4l2_fourcc('V', 'P', '8', '0') /*␣
↪VP8 */
#define V4L2_PIX_FMT_VP9      v4l2_fourcc('V', 'P', '9', '0') /*␣
↪VP9 */
#define V4L2_PIX_FMT_HEVC     v4l2_fourcc('H', 'E', 'V', 'C') /*␣
```

```
↪HEVC aka H.265 */
#define V4L2_PIX_FMT_FWHT     v4l2_fourcc('F', 'W', 'H', 'T') /*␣
↪Fast Walsh Hadamard Transform (vicodec) */
#define V4L2_PIX_FMT_FWHT_STATELESS     v4l2_fourcc('S', 'F', 'W',␣
↪'H') /* Stateless FWHT (vicodec) */

/*  Vendor-specific formats   */
#define V4L2_PIX_FMT_CPIA1    v4l2_fourcc('C', 'P', 'I', 'A') /*␣
↪cpia1 YUV */
#define V4L2_PIX_FMT_WNVA     v4l2_fourcc('W', 'N', 'V', 'A') /*␣
↪Winnov hw compress */
#define V4L2_PIX_FMT_SN9C10X  v4l2_fourcc('S', '9', '1', '0') /*␣
↪SN9C10x compression */
#define V4L2_PIX_FMT_SN9C20X_I420 v4l2_fourcc('S', '9', '2', '0') /
↪* SN9C20x YUV 4:2:0 */
#define V4L2_PIX_FMT_PWC1     v4l2_fourcc('P', 'W', 'C', '1') /*␣
↪pwc older webcam */
#define V4L2_PIX_FMT_PWC2     v4l2_fourcc('P', 'W', 'C', '2') /*␣
↪pwc newer webcam */
#define V4L2_PIX_FMT_ET61X251 v4l2_fourcc('E', '6', '2', '5') /*␣
↪ET61X251 compression */
#define V4L2_PIX_FMT_SPCA501  v4l2_fourcc('S', '5', '0', '1') /*␣
↪YUYV per line */
#define V4L2_PIX_FMT_SPCA505  v4l2_fourcc('S', '5', '0', '5') /*␣
↪YYUV per line */
#define V4L2_PIX_FMT_SPCA508  v4l2_fourcc('S', '5', '0', '8') /*␣
↪YUVY per line */
#define V4L2_PIX_FMT_SPCA561  v4l2_fourcc('S', '5', '6', '1') /*␣
↪compressed GBRG bayer */
#define V4L2_PIX_FMT_PAC207   v4l2_fourcc('P', '2', '0', '7') /*␣
↪compressed BGGR bayer */
#define V4L2_PIX_FMT_MR97310A v4l2_fourcc('M', '3', '1', '0') /*␣
↪compressed BGGR bayer */
#define V4L2_PIX_FMT_JL2005BCD v4l2_fourcc('J', 'L', '2', '0') /*␣
↪compressed RGGB bayer */
#define V4L2_PIX_FMT_SN9C2028 v4l2_fourcc('S', 'O', 'N', 'X') /*␣
↪compressed GBRG bayer */
#define V4L2_PIX_FMT_SQ905C   v4l2_fourcc('9', '0', '5', 'C') /*␣
↪compressed RGGB bayer */
#define V4L2_PIX_FMT_PJPG     v4l2_fourcc('P', 'J', 'P', 'G') /*␣
↪Pixart 73xx JPEG */
#define V4L2_PIX_FMT_OV511    v4l2_fourcc('O', '5', '1', '1') /*␣
↪ov511 JPEG */
#define V4L2_PIX_FMT_OV518    v4l2_fourcc('O', '5', '1', '8') /*␣
↪ov518 JPEG */
#define V4L2_PIX_FMT_STV0680  v4l2_fourcc('S', '6', '8', '0') /*␣
↪stv0680 bayer */
#define V4L2_PIX_FMT_TM6000   v4l2_fourcc('T', 'M', '6', '0') /*␣
↪tm5600/tm60x0 */
#define V4L2_PIX_FMT_CIT_YYVYUY v4l2_fourcc('C', 'I', 'T', 'V') /*␣
↪one line of Y then 1 line of VYUY */
```

```
#define V4L2_PIX_FMT_KONICA420  v4l2_fourcc('K', 'O', 'N', 'I') /*␣
↪YUV420 planar in blocks of 256 pixels */
#define V4L2_PIX_FMT_JPGL       v4l2_fourcc('J', 'P', 'G', 'L') /*␣
↪JPEG-Lite */
#define V4L2_PIX_FMT_SE401      v4l2_fourcc('S', '4', '0', '1') /*␣
↪se401 janggu compressed rgb */
#define V4L2_PIX_FMT_S5C_UYVY_JPG v4l2_fourcc('S', '5', 'C', 'I') /
↪* S5C73M3 interleaved UYVY/JPEG */
#define V4L2_PIX_FMT_Y8I      v4l2_fourcc('Y', '8', 'I', ' ') /*␣
↪Greyscale 8-bit L/R interleaved */
#define V4L2_PIX_FMT_Y12I     v4l2_fourcc('Y', '1', '2', 'I') /*␣
↪Greyscale 12-bit L/R interleaved */
#define V4L2_PIX_FMT_Z16      v4l2_fourcc('Z', '1', '6', ' ') /*␣
↪Depth data 16-bit */
#define V4L2_PIX_FMT_MT21C    v4l2_fourcc('M', 'T', '2', '1') /*␣
↪Mediatek compressed block mode  */
#define V4L2_PIX_FMT_INZI     v4l2_fourcc('I', 'N', 'Z', 'I') /*␣
↪Intel Planar Greyscale 10-bit and Depth 16-bit */
#define V4L2_PIX_FMT_SUNXI_TILED_NV12 v4l2_fourcc('S', 'T', '1',␣
↪'2') /* Sunxi Tiled NV12 Format */
#define V4L2_PIX_FMT_CNF4     v4l2_fourcc('C', 'N', 'F', '4') /*␣
↪Intel 4-bit packed depth confidence information */

/* 10bit raw bayer packed, 32 bytes for every 25 pixels, last LSB 6␣
↪bits unused */
#define V4L2_PIX_FMT_IPU3_SBGGR10       v4l2_fourcc('i', 'p', '3',␣
↪'b') /* IPU3 packed 10-bit BGGR bayer */
#define V4L2_PIX_FMT_IPU3_SGBRG10       v4l2_fourcc('i', 'p', '3',␣
↪'g') /* IPU3 packed 10-bit GBRG bayer */
#define V4L2_PIX_FMT_IPU3_SGRBG10       v4l2_fourcc('i', 'p', '3',␣
↪'G') /* IPU3 packed 10-bit GRBG bayer */
#define V4L2_PIX_FMT_IPU3_SRGGB10       v4l2_fourcc('i', 'p', '3',␣
↪'r') /* IPU3 packed 10-bit RGGB bayer */

/* SDR formats - used only for Software Defined Radio devices */
#define V4L2_SDR_FMT_CU8          v4l2_fourcc('C', 'U', '0', '8') /
↪* IQ u8 */
#define V4L2_SDR_FMT_CU16LE       v4l2_fourcc('C', 'U', '1', '6') /
↪* IQ u16le */
#define V4L2_SDR_FMT_CS8          v4l2_fourcc('C', 'S', '0', '8') /
↪* complex s8 */
#define V4L2_SDR_FMT_CS14LE       v4l2_fourcc('C', 'S', '1', '4') /
↪* complex s14le */
#define V4L2_SDR_FMT_RU12LE       v4l2_fourcc('R', 'U', '1', '2') /
↪* real u12le */
#define V4L2_SDR_FMT_PCU16BE      v4l2_fourcc('P', 'C', '1', '6') /
↪* planar complex u16be */
#define V4L2_SDR_FMT_PCU18BE      v4l2_fourcc('P', 'C', '1', '8') /
↪* planar complex u18be */
#define V4L2_SDR_FMT_PCU20BE      v4l2_fourcc('P', 'C', '2', '0') /
↪* planar complex u20be */
```

```
/* Touch formats - used for Touch devices */
#define V4L2_TCH_FMT_DELTA_TD16 v4l2_fourcc('T', 'D', '1', '6') /*␣
↪16-bit signed deltas */
#define V4L2_TCH_FMT_DELTA_TD08 v4l2_fourcc('T', 'D', '0', '8') /*␣
↪8-bit signed deltas */
#define V4L2_TCH_FMT_TU16       v4l2_fourcc('T', 'U', '1', '6') /*␣
↪16-bit unsigned touch data */
#define V4L2_TCH_FMT_TU08       v4l2_fourcc('T', 'U', '0', '8') /*␣
↪8-bit unsigned touch data */

/* Meta-data formats */
#define V4L2_META_FMT_VSP1_HGO    v4l2_fourcc('V', 'S', 'P', 'H') /
↪* R-Car VSP1 1-D Histogram */
#define V4L2_META_FMT_VSP1_HGT    v4l2_fourcc('V', 'S', 'P', 'T') /
↪* R-Car VSP1 2-D Histogram */
#define V4L2_META_FMT_UVC         v4l2_fourcc('U', 'V', 'C', 'H') /
↪* UVC Payload Header metadata */
#define V4L2_META_FMT_D4XX        v4l2_fourcc('D', '4', 'X', 'X') /
↪* D4XX Payload Header metadata */
#define V4L2_META_FMT_VIVID       v4l2_fourcc('V', 'I', 'V', 'D') /
↪* Vivid Metadata */

/* priv field value to indicates that subsequent fields are valid.␣
↪*/
#define V4L2_PIX_FMT_PRIV_MAGIC         0xfeedcafe

/* Flags */
#define V4L2_PIX_FMT_FLAG_PREMUL_ALPHA  0x00000001
#define V4L2_PIX_FMT_FLAG_SET_CSC       0x00000002

/*
 *      F O R M A T   E N U M E R A T I O N
 */
struct v4l2_fmtdesc {
        __u32               index;             /* Format number      ␣
↪ */
        __u32               type;              /* enum v4l2_buf_
↪type */
        __u32               flags;
        __u8                description[32];   /* Description␣
↪string */
        __u32               pixelformat;       /* Format fourcc      ␣
↪ */
        __u32               mbus_code;          /* Media bus code    ␣
↪ */
        __u32               reserved[3];
};

#define V4L2_FMT_FLAG_COMPRESSED                0x0001
#define V4L2_FMT_FLAG_EMULATED                  0x0002
```

```
#define V4L2_FMT_FLAG_CONTINUOUS_BYTESTREAM     0x0004
#define V4L2_FMT_FLAG_DYN_RESOLUTION            0x0008
#define V4L2_FMT_FLAG_ENC_CAP_FRAME_INTERVAL    0x0010
#define V4L2_FMT_FLAG_CSC_COLORSPACE            0x0020
#define V4L2_FMT_FLAG_CSC_XFER_FUNC             0x0040
#define V4L2_FMT_FLAG_CSC_YCBCR_ENC             0x0080
#define V4L2_FMT_FLAG_CSC_HSV_ENC               V4L2_FMT_FLAG_CSC_
↪YCBCR_ENC
#define V4L2_FMT_FLAG_CSC_QUANTIZATION          0x0100

        /* Frame Size and frame rate enumeration */
/*
 *      F R A M E   S I Z E   E N U M E R A T I O N
 */
enum v4l2_frmsizetypes {
        V4L2_FRMSIZE_TYPE_DISCRETE      = 1,
        V4L2_FRMSIZE_TYPE_CONTINUOUS    = 2,
        V4L2_FRMSIZE_TYPE_STEPWISE      = 3,
};

struct v4l2_frmsize_discrete {
        __u32                   width;          /* Frame width␣
↪[pixel] */
        __u32                   height;         /* Frame height␣
↪[pixel] */
};

struct v4l2_frmsize_stepwise {
        __u32                   min_width;      /* Minimum frame␣
↪width [pixel] */
        __u32                   max_width;      /* Maximum frame␣
↪width [pixel] */
        __u32                   step_width;     /* Frame width step␣
↪size [pixel] */
        __u32                   min_height;     /* Minimum frame␣
↪height [pixel] */
        __u32                   max_height;     /* Maximum frame␣
↪height [pixel] */
        __u32                   step_height;    /* Frame height␣
↪step size [pixel] */
};

struct v4l2_frmsizeenum {
        __u32                   index;          /* Frame size␣
↪number */
        __u32                   pixel_format;   /* Pixel format */
        __u32                   type;           /* Frame size type␣
↪the device supports. */

        union {                                 /* Frame size */
                struct v4l2_frmsize_discrete    discrete;
```

```
                struct v4l2_frmsize_stepwise    stepwise;
        };

        __u32   reserved[2];                    /* Reserved space␣
↪for future use */
};

/*
 *      F R A M E   R A T E   E N U M E R A T I O N
 */
enum v4l2_frmivaltypes {
        V4L2_FRMIVAL_TYPE_DISCRETE      = 1,
        V4L2_FRMIVAL_TYPE_CONTINUOUS    = 2,
        V4L2_FRMIVAL_TYPE_STEPWISE      = 3,
};

struct v4l2_frmival_stepwise {
        struct v4l2_fract       min;            /* Minimum frame␣
↪interval [s] */
        struct v4l2_fract       max;            /* Maximum frame␣
↪interval [s] */
        struct v4l2_fract       step;           /* Frame interval␣
↪step size [s] */
};

struct v4l2_frmivalenum {
        __u32                   index;          /* Frame format␣
↪index */
        __u32                   pixel_format;   /* Pixel format */
        __u32                   width;          /* Frame width */
        __u32                   height;         /* Frame height */
        __u32                   type;           /* Frame interval␣
↪type the device supports. */

        union {                                 /* Frame interval */
                struct v4l2_fract               discrete;
                struct v4l2_frmival_stepwise    stepwise;
        };

        __u32   reserved[2];                    /* Reserved space␣
↪for future use */
};

/*
 *      T I M E C O D E
 */
struct v4l2_timecode {
        __u32   type;
        __u32   flags;
        __u8    frames;
        __u8    seconds;
```

```
        __u8    minutes;
        __u8    hours;
        __u8    userbits[4];
};

/*  Type  */
#define V4L2_TC_TYPE_24FPS              1
#define V4L2_TC_TYPE_25FPS              2
#define V4L2_TC_TYPE_30FPS              3
#define V4L2_TC_TYPE_50FPS              4
#define V4L2_TC_TYPE_60FPS              5

/*  Flags  */
#define V4L2_TC_FLAG_DROPFRAME          0x0001 /* "drop-frame" mode␣
↪*/
#define V4L2_TC_FLAG_COLORFRAME         0x0002
#define V4L2_TC_USERBITS_field          0x000C
#define V4L2_TC_USERBITS_USERDEFINED    0x0000
#define V4L2_TC_USERBITS_8BITCHARS      0x0008
/* The above is based on SMPTE timecodes */

struct v4l2_jpegcompression {
        int quality;

        int  APPn;              /* Number of APP segment to be␣
↪written,
                                 * must be 0..15 */
        int  APP_len;           /* Length of data in JPEG APPn␣
↪segment */
        char APP_data[60];      /* Data in the JPEG APPn segment. */

        int  COM_len;           /* Length of data in JPEG COM␣
↪segment */
        char COM_data[60];      /* Data in JPEG COM segment */

        __u32 jpeg_markers;     /* Which markers should go into the␣
↪JPEG
                                 * output. Unless you exactly know␣
↪what
                                 * you do, leave them untouched.
                                 * Including less markers will make␣
↪the
                                 * resulting code smaller, but␣
↪there will
                                 * be fewer applications which can␣
↪read it.
                                 * The presence of the APP and COM␣
↪marker
                                 * is influenced by APP_len and COM_
↪len
                                 * ONLY, not by this property! */
```

```
#define V4L2_JPEG_MARKER_DHT (1<<3)    /* Define Huffman Tables */
#define V4L2_JPEG_MARKER_DQT (1<<4)    /* Define Quantization␣
↪Tables */
#define V4L2_JPEG_MARKER_DRI (1<<5)    /* Define Restart Interval */
#define V4L2_JPEG_MARKER_COM (1<<6)    /* Comment segment */
#define V4L2_JPEG_MARKER_APP (1<<7)    /* App segment, driver will
                                        * always use APP0 */
};

/*
 *      M E M O R Y - M A P P I N G   B U F F E R S
 */

#ifdef __KERNEL__
/*
 * This corresponds to the user space version of timeval
 * for 64-bit time_t. sparc64 is different from everyone
 * else, using the microseconds in the wrong half of the
 * second 64-bit word.
 */
struct __kernel_v4l2_timeval {
        long long       tv_sec;
#if defined(__sparc__) && defined(__arch64__)
        int             tv_usec;
        int             __pad;
#else
        long long       tv_usec;
#endif
};
#endif

struct v4l2_requestbuffers {
        __u32                   count;
        __u32                   type;           /* enum v4l2_buf_
↪type */
        __u32                   memory;         /* enum v4l2_memory␣
↪*/
        __u32                   capabilities;
        __u32                   reserved[1];
};

/* capabilities for struct v4l2_requestbuffers and v4l2_create_
↪buffers */
#define V4L2_BUF_CAP_SUPPORTS_MMAP                      (1 << 0)
#define V4L2_BUF_CAP_SUPPORTS_USERPTR                   (1 << 1)
#define V4L2_BUF_CAP_SUPPORTS_DMABUF                    (1 << 2)
#define V4L2_BUF_CAP_SUPPORTS_REQUESTS                  (1 << 3)
#define V4L2_BUF_CAP_SUPPORTS_ORPHANED_BUFS             (1 << 4)
#define V4L2_BUF_CAP_SUPPORTS_M2M_HOLD_CAPTURE_BUF      (1 << 5)
#define V4L2_BUF_CAP_SUPPORTS_MMAP_CACHE_HINTS          (1 << 6)
```

```
/**
 * struct v4l2_plane - plane info for multi-planar buffers
 * @bytesused:          number of bytes occupied by data in the␣
↪plane (payload)
 * @length:             size of this plane (NOT the payload) in␣
↪bytes
 * @mem_offset:         when memory in the associated struct v4l2_
↪buffer is
 *                      V4L2_MEMORY_MMAP, equals the offset from␣
↪the start of
 *                      the device memory for this plane (or is a␣
↪"cookie" that
 *                      should be passed to mmap() called on the␣
↪video node)
 * @userptr:            when memory is V4L2_MEMORY_USERPTR, a␣
↪userspace pointer
 *                      pointing to this plane
 * @fd:                 when memory is V4L2_MEMORY_DMABUF, a␣
↪userspace file
 *                      descriptor associated with this plane
 * @data_offset:        offset in the plane to the start of data;␣
↪usually 0,
 *                      unless there is a header in front of the␣
↪data
 *
 * Multi-planar buffers consist of one or more planes, e.g. an␣
↪YCbCr buffer
 * with two planes can have one plane for Y, and another for␣
↪interleaved CbCr
 * components. Each plane can reside in a separate memory buffer,␣
↪or even in
 * a completely separate memory node (e.g. in embedded devices).
 */
struct v4l2_plane {
        __u32                   bytesused;
        __u32                   length;
        union {
                __u32           mem_offset;
                unsigned long   userptr;
                __s32           fd;
        } m;
        __u32                   data_offset;
        __u32                   reserved[11];
};

/**
 * struct v4l2_buffer - video buffer info
 * @index:      id number of the buffer
 * @type:       enum v4l2_buf_type; buffer type (type == *_MPLANE␣
↪for
```

```
 *               multiplanar buffers);
 * @bytesused:   number of bytes occupied by data in the buffer
↪(payload);
 *               unused (set to 0) for multiplanar buffers
 * @flags:       buffer informational flags
 * @field:       enum v4l2_field; field order of the image in the
↪buffer
 * @timestamp:   frame timestamp
 * @timecode:    frame timecode
 * @sequence:    sequence count of this frame
 * @memory:      enum v4l2_memory; the method, in which the actual
↪video data is
 *               passed
 * @offset:      for non-multiplanar buffers with memory == V4L2_
↪MEMORY_MMAP;
 *               offset from the start of the device memory for this
↪plane,
 *               (or a "cookie" that should be passed to mmap() as
↪offset)
 * @userptr:     for non-multiplanar buffers with memory == V4L2_
↪MEMORY_USERPTR;
 *               a userspace pointer pointing to this buffer
 * @fd:          for non-multiplanar buffers with memory == V4L2_
↪MEMORY_DMABUF;
 *               a userspace file descriptor associated with this
↪buffer
 * @planes:      for multiplanar buffers; userspace pointer to the
↪array of plane
 *               info structs for this buffer
 * @length:      size in bytes of the buffer (NOT its payload) for
↪single-plane
 *               buffers (when type != *_MPLANE); number of elements
↪in the
 *               planes array for multi-plane buffers
 * @request_fd: fd of the request that this buffer should use
 *
 * Contains data exchanged by application and driver using one of
↪the Streaming
 * I/O methods.
 */
struct v4l2_buffer {
        __u32                   index;
        __u32                   type;
        __u32                   bytesused;
        __u32                   flags;
        __u32                   field;
#ifdef __KERNEL__
        struct __kernel_v4l2_timeval timestamp;
#else
        struct timeval          timestamp;
#endif
```

```
        struct v4l2_timecode    timecode;
        __u32                   sequence;

        /* memory location */
        __u32                   memory;
        union {
                __u32           offset;
                unsigned long   userptr;
                struct v4l2_plane *planes;
                __s32           fd;
        } m;
        __u32                   length;
        __u32                   reserved2;
        union {
                __s32           request_fd;
                __u32           reserved;
        };
};

#ifndef __KERNEL__
/**
 * v4l2_timeval_to_ns - Convert timeval to nanoseconds
 * @ts:         pointer to the timeval variable to be converted
 *
 * Returns the scalar nanosecond representation of the timeval
 * parameter.
 */
static inline __u64 v4l2_timeval_to_ns(const struct timeval *tv)
{
        return (__u64)tv->tv_sec * 1000000000ULL + tv->tv_usec *␣
↪1000;
}
#endif

/*  Flags for 'flags' field */
/* Buffer is mapped (flag) */
#define V4L2_BUF_FLAG_MAPPED                    0x00000001
/* Buffer is queued for processing */
#define V4L2_BUF_FLAG_QUEUED                    0x00000002
/* Buffer is ready */
#define V4L2_BUF_FLAG_DONE                      0x00000004
/* Image is a keyframe (I-frame) */
#define V4L2_BUF_FLAG_KEYFRAME                  0x00000008
/* Image is a P-frame */
#define V4L2_BUF_FLAG_PFRAME                    0x00000010
/* Image is a B-frame */
#define V4L2_BUF_FLAG_BFRAME                    0x00000020
/* Buffer is ready, but the data contained within is corrupted. */
#define V4L2_BUF_FLAG_ERROR                     0x00000040
/* Buffer is added to an unqueued request */
#define V4L2_BUF_FLAG_IN_REQUEST                0x00000080
```

```
/* timecode field is valid */
#define V4L2_BUF_FLAG_TIMECODE                  0x00000100
/* Don't return the capture buffer until OUTPUT timestamp changes */
#define V4L2_BUF_FLAG_M2M_HOLD_CAPTURE_BUF      0x00000200
/* Buffer is prepared for queuing */
#define V4L2_BUF_FLAG_PREPARED                  0x00000400
/* Cache handling flags */
#define V4L2_BUF_FLAG_NO_CACHE_INVALIDATE       0x00000800
#define V4L2_BUF_FLAG_NO_CACHE_CLEAN            0x00001000
/* Timestamp type */
#define V4L2_BUF_FLAG_TIMESTAMP_MASK            0x0000e000
#define V4L2_BUF_FLAG_TIMESTAMP_UNKNOWN         0x00000000
#define V4L2_BUF_FLAG_TIMESTAMP_MONOTONIC       0x00002000
#define V4L2_BUF_FLAG_TIMESTAMP_COPY            0x00004000
/* Timestamp sources. */
#define V4L2_BUF_FLAG_TSTAMP_SRC_MASK           0x00070000
#define V4L2_BUF_FLAG_TSTAMP_SRC_EOF            0x00000000
#define V4L2_BUF_FLAG_TSTAMP_SRC_SOE            0x00010000
/* mem2mem encoder/decoder */
#define V4L2_BUF_FLAG_LAST                      0x00100000
/* request_fd is valid */
#define V4L2_BUF_FLAG_REQUEST_FD                0x00800000

/**
 * struct v4l2_exportbuffer - export of video buffer as DMABUF file␣
↪descriptor
 *
 * @index:      id number of the buffer
 * @type:       enum v4l2_buf_type; buffer type (type == *_MPLANE␣
↪for
 *              multiplanar buffers);
 * @plane:      index of the plane to be exported, 0 for single␣
↪plane queues
 * @flags:      flags for newly created file, currently only O_
↪CLOEXEC is
 *              supported, refer to manual of open syscall for more␣
↪details
 * @fd:         file descriptor associated with DMABUF (set by␣
↪driver)
 *
 * Contains data used for exporting a video buffer as DMABUF file␣
↪descriptor.
 * The buffer is identified by a 'cookie' returned by VIDIOC_
↪QUERYBUF
 * (identical to the cookie used to mmap() the buffer to userspace).
↪ All
 * reserved fields must be set to zero. The field reserved0 is␣
↪expected to
 * become a structure 'type' allowing an alternative layout of the␣
↪structure
 * content. Therefore this field should not be used for any other␣
```

```
↪extensions.
 */
struct v4l2_exportbuffer {
        __u32           type; /* enum v4l2_buf_type */
        __u32           index;
        __u32           plane;
        __u32           flags;
        __s32           fd;
        __u32           reserved[11];
};

/*
 *      O V E R L A Y   P R E V I E W
 */
struct v4l2_framebuffer {
        __u32                   capability;
        __u32                   flags;
/* FIXME: in theory we should pass something like PCI device +␣
↪memory
 * region + offset instead of some physical address */
        void                    *base;
        struct {
                __u32           width;
                __u32           height;
                __u32           pixelformat;
                __u32           field;          /* enum v4l2_field␣
↪*/
                __u32           bytesperline;   /* for padding,␣
↪zero if unused */
                __u32           sizeimage;
                __u32           colorspace;     /* enum v4l2_
↪colorspace */
                __u32           priv;           /* reserved field,␣
↪set to 0 */
        } fmt;
};
/*  Flags for the 'capability' field. Read only */
#define V4L2_FBUF_CAP_EXTERNOVERLAY     0x0001
#define V4L2_FBUF_CAP_CHROMAKEY         0x0002
#define V4L2_FBUF_CAP_LIST_CLIPPING     0x0004
#define V4L2_FBUF_CAP_BITMAP_CLIPPING   0x0008
#define V4L2_FBUF_CAP_LOCAL_ALPHA       0x0010
#define V4L2_FBUF_CAP_GLOBAL_ALPHA      0x0020
#define V4L2_FBUF_CAP_LOCAL_INV_ALPHA   0x0040
#define V4L2_FBUF_CAP_SRC_CHROMAKEY     0x0080
/*  Flags for the 'flags' field. */
#define V4L2_FBUF_FLAG_PRIMARY          0x0001
#define V4L2_FBUF_FLAG_OVERLAY          0x0002
#define V4L2_FBUF_FLAG_CHROMAKEY        0x0004
#define V4L2_FBUF_FLAG_LOCAL_ALPHA      0x0008
#define V4L2_FBUF_FLAG_GLOBAL_ALPHA     0x0010
```

```
#define V4L2_FBUF_FLAG_LOCAL_INV_ALPHA  0x0020
#define V4L2_FBUF_FLAG_SRC_CHROMAKEY    0x0040

struct v4l2_clip {
        struct v4l2_rect        c;
        struct v4l2_clip        __user *next;
};

struct v4l2_window {
        struct v4l2_rect        w;
        __u32                   field;   /* enum v4l2_field */
        __u32                   chromakey;
        struct v4l2_clip        __user *clips;
        __u32                   clipcount;
        void                    __user *bitmap;
        __u8                    global_alpha;
};

/*
 *      C A P T U R E   P A R A M E T E R S
 */
struct v4l2_captureparm {
        __u32              capability;    /*  Supported modes */
        __u32              capturemode;   /*  Current mode */
        struct v4l2_fract  timeperframe;  /*  Time per frame in␣
↪seconds */
        __u32              extendedmode;  /*  Driver-specific␣
↪extensions */
        __u32              readbuffers;   /*  # of buffers for read␣
↪*/
        __u32              reserved[4];
};

/*  Flags for 'capability' and 'capturemode' fields */
#define V4L2_MODE_HIGHQUALITY   0x0001  /*  High quality imaging␣
↪mode */
#define V4L2_CAP_TIMEPERFRAME   0x1000  /*  timeperframe field is␣
↪supported */

struct v4l2_outputparm {
        __u32              capability;   /*  Supported modes */
        __u32              outputmode;   /*  Current mode */
        struct v4l2_fract  timeperframe; /*  Time per frame in␣
↪seconds */
        __u32              extendedmode; /*  Driver-specific␣
↪extensions */
        __u32              writebuffers; /*  # of buffers for write␣
↪*/
        __u32              reserved[4];
};
```

```
/*
 *      I N P U T   I M A G E   C R O P P I N G
 */
struct v4l2_cropcap {
        __u32                   type;   /* enum v4l2_buf_type */
        struct v4l2_rect        bounds;
        struct v4l2_rect        defrect;
        struct v4l2_fract       pixelaspect;
};

struct v4l2_crop {
        __u32                   type;   /* enum v4l2_buf_type */
        struct v4l2_rect        c;
};

/**
 * struct v4l2_selection - selection info
 * @type:       buffer type (do not use *_MPLANE types)
 * @target:     Selection target, used to choose one of possible␣
↪rectangles;
 *              defined in v4l2-common.h; V4L2_SEL_TGT_* .
 * @flags:      constraints flags, defined in v4l2-common.h; V4L2_
↪SEL_FLAG_*.
 * @r:          coordinates of selection window
 * @reserved:   for future use, rounds structure size to 64 bytes,␣
↪set to zero
 *
 * Hardware may use multiple helper windows to process a video␣
↪stream.
 * The structure is used to exchange this selection areas between
 * an application and a driver.
 */
struct v4l2_selection {
        __u32                   type;
        __u32                   target;
        __u32                   flags;
        struct v4l2_rect        r;
        __u32                   reserved[9];
};

/*
 *      A N A L O G   V I D E O   S T A N D A R D
 */

typedef __u64 v4l2_std_id;

/*
 * Attention: Keep the V4L2_STD_* bit definitions in sync with
 * include/dt-bindings/display/sdtv-standards.h SDTV_STD_* bit␣
↪definitions.
 */
```

```
/* one bit for each */
#define V4L2_STD_PAL_B          ((v4l2_std_id)0x00000001)
#define V4L2_STD_PAL_B1         ((v4l2_std_id)0x00000002)
#define V4L2_STD_PAL_G          ((v4l2_std_id)0x00000004)
#define V4L2_STD_PAL_H          ((v4l2_std_id)0x00000008)
#define V4L2_STD_PAL_I          ((v4l2_std_id)0x00000010)
#define V4L2_STD_PAL_D          ((v4l2_std_id)0x00000020)
#define V4L2_STD_PAL_D1         ((v4l2_std_id)0x00000040)
#define V4L2_STD_PAL_K          ((v4l2_std_id)0x00000080)

#define V4L2_STD_PAL_M          ((v4l2_std_id)0x00000100)
#define V4L2_STD_PAL_N          ((v4l2_std_id)0x00000200)
#define V4L2_STD_PAL_Nc         ((v4l2_std_id)0x00000400)
#define V4L2_STD_PAL_60         ((v4l2_std_id)0x00000800)

#define V4L2_STD_NTSC_M         ((v4l2_std_id)0x00001000)       /*␣
↪BTSC */
#define V4L2_STD_NTSC_M_JP      ((v4l2_std_id)0x00002000)       /*␣
↪EIA-J */
#define V4L2_STD_NTSC_443       ((v4l2_std_id)0x00004000)
#define V4L2_STD_NTSC_M_KR      ((v4l2_std_id)0x00008000)       /*␣
↪FM A2 */

#define V4L2_STD_SECAM_B        ((v4l2_std_id)0x00010000)
#define V4L2_STD_SECAM_D        ((v4l2_std_id)0x00020000)
#define V4L2_STD_SECAM_G        ((v4l2_std_id)0x00040000)
#define V4L2_STD_SECAM_H        ((v4l2_std_id)0x00080000)
#define V4L2_STD_SECAM_K        ((v4l2_std_id)0x00100000)
#define V4L2_STD_SECAM_K1       ((v4l2_std_id)0x00200000)
#define V4L2_STD_SECAM_L        ((v4l2_std_id)0x00400000)
#define V4L2_STD_SECAM_LC       ((v4l2_std_id)0x00800000)

/* ATSC/HDTV */
#define V4L2_STD_ATSC_8_VSB     ((v4l2_std_id)0x01000000)
#define V4L2_STD_ATSC_16_VSB    ((v4l2_std_id)0x02000000)

/* FIXME:
   Although std_id is 64 bits, there is an issue on PPC32␣
↪architecture that
   makes switch(__u64) to break. So, there's a hack on v4l2-common.
↪c rounding
   this value to 32 bits.
   As, currently, the max value is for V4L2_STD_ATSC_16_VSB (30␣
↪bits wide),
   it should work fine. However, if needed to add more than two␣
↪standards,
   v4l2-common.c should be fixed.
 */

/*
 * Some macros to merge video standards in order to make live␣
```

```
↪easier for the
 * drivers and V4L2 applications
 */

/*
 * "Common" NTSC/M - It should be noticed that V4L2_STD_NTSC_443 is
 * Missing here.
 */
#define V4L2_STD_NTSC           (V4L2_STD_NTSC_M        |\
                                 V4L2_STD_NTSC_M_JP     |\
                                 V4L2_STD_NTSC_M_KR)
/* Secam macros */
#define V4L2_STD_SECAM_DK       (V4L2_STD_SECAM_D       |\
                                 V4L2_STD_SECAM_K       |\
                                 V4L2_STD_SECAM_K1)
/* All Secam Standards */
#define V4L2_STD_SECAM          (V4L2_STD_SECAM_B       |\
                                 V4L2_STD_SECAM_G       |\
                                 V4L2_STD_SECAM_H       |\
                                 V4L2_STD_SECAM_DK      |\
                                 V4L2_STD_SECAM_L       |\
                                 V4L2_STD_SECAM_LC)
/* PAL macros */
#define V4L2_STD_PAL_BG         (V4L2_STD_PAL_B         |\
                                 V4L2_STD_PAL_B1        |\
                                 V4L2_STD_PAL_G)
#define V4L2_STD_PAL_DK         (V4L2_STD_PAL_D         |\
                                 V4L2_STD_PAL_D1        |\
                                 V4L2_STD_PAL_K)
/*
 * "Common" PAL - This macro is there to be compatible with the old
 * V4L1 concept of "PAL": /BGDKHI.
 * Several PAL standards are missing here: /M, /N and /Nc
 */
#define V4L2_STD_PAL            (V4L2_STD_PAL_BG        |\
                                 V4L2_STD_PAL_DK        |\
                                 V4L2_STD_PAL_H         |\
                                 V4L2_STD_PAL_I)
/* Chroma "agnostic" standards */
#define V4L2_STD_B              (V4L2_STD_PAL_B         |\
                                 V4L2_STD_PAL_B1        |\
                                 V4L2_STD_SECAM_B)
#define V4L2_STD_G              (V4L2_STD_PAL_G         |\
                                 V4L2_STD_SECAM_G)
#define V4L2_STD_H              (V4L2_STD_PAL_H         |\
                                 V4L2_STD_SECAM_H)
#define V4L2_STD_L              (V4L2_STD_SECAM_L       |\
                                 V4L2_STD_SECAM_LC)
#define V4L2_STD_GH             (V4L2_STD_G             |\
                                 V4L2_STD_H)
#define V4L2_STD_DK             (V4L2_STD_PAL_DK        |\
```

```
                                 V4L2_STD_SECAM_DK)
#define V4L2_STD_BG             (V4L2_STD_B             |\
                                 V4L2_STD_G)
#define V4L2_STD_MN             (V4L2_STD_PAL_M         |\
                                 V4L2_STD_PAL_N         |\
                                 V4L2_STD_PAL_Nc        |\
                                 V4L2_STD_NTSC)

/* Standards where MTS/BTSC stereo could be found */
#define V4L2_STD_MTS            (V4L2_STD_NTSC_M        |\
                                 V4L2_STD_PAL_M         |\
                                 V4L2_STD_PAL_N         |\
                                 V4L2_STD_PAL_Nc)

/* Standards for Countries with 60Hz Line frequency */
#define V4L2_STD_525_60         (V4L2_STD_PAL_M         |\
                                 V4L2_STD_PAL_60        |\
                                 V4L2_STD_NTSC          |\
                                 V4L2_STD_NTSC_443)
/* Standards for Countries with 50Hz Line frequency */
#define V4L2_STD_625_50         (V4L2_STD_PAL           |\
                                 V4L2_STD_PAL_N         |\
                                 V4L2_STD_PAL_Nc        |\
                                 V4L2_STD_SECAM)

#define V4L2_STD_ATSC           (V4L2_STD_ATSC_8_VSB    |\
                                 V4L2_STD_ATSC_16_VSB)
/* Macros with none and all analog standards */
#define V4L2_STD_UNKNOWN        0
#define V4L2_STD_ALL            (V4L2_STD_525_60        |\
                                 V4L2_STD_625_50)

struct v4l2_standard {
        __u32                index;
        v4l2_std_id          id;
        __u8                 name[24];
        struct v4l2_fract    frameperiod; /* Frames, not fields */
        __u32                framelines;
        __u32                reserved[4];
};

/*
 *      D V     B T     T I M I N G S
 */

/** struct v4l2_bt_timings - BT.656/BT.1120 timing data
 * @width:      total width of the active video in pixels
 * @height:     total height of the active video in lines
 * @interlaced: Interlaced or progressive
 * @polarities: Positive or negative polarities
 * @pixelclock: Pixel clock in HZ. Ex. 74.25MHz->74250000
```

```
 * @hfrontporch:Horizontal front porch in pixels
 * @hsync:      Horizontal Sync length in pixels
 * @hbackporch: Horizontal back porch in pixels
 * @vfrontporch:Vertical front porch in lines
 * @vsync:      Vertical Sync length in lines
 * @vbackporch: Vertical back porch in lines
 * @il_vfrontporch:Vertical front porch for the even field
 *              (aka field 2) of interlaced field formats
 * @il_vsync:   Vertical Sync length for the even field
 *              (aka field 2) of interlaced field formats
 * @il_vbackporch:Vertical back porch for the even field
 *              (aka field 2) of interlaced field formats
 * @standards:  Standards the timing belongs to
 * @flags:      Flags
 * @picture_aspect: The picture aspect ratio (hor/vert).
 * @cea861_vic: VIC code as per the CEA-861 standard.
 * @hdmi_vic:   VIC code as per the HDMI standard.
 * @reserved:   Reserved fields, must be zeroed.
 *
 * A note regarding vertical interlaced timings: height refers to␣
↪the total
 * height of the active video frame (= two fields). The blanking␣
↪timings refer
 * to the blanking of each field. So the height of the total frame␣
↪is
 * calculated as follows:
 *
 * tot_height = height + vfrontporch + vsync + vbackporch +
 *                       il_vfrontporch + il_vsync + il_vbackporch
 *
 * The active height of each field is height / 2.
 */
struct v4l2_bt_timings {
        __u32   width;
        __u32   height;
        __u32   interlaced;
        __u32   polarities;
        __u64   pixelclock;
        __u32   hfrontporch;
        __u32   hsync;
        __u32   hbackporch;
        __u32   vfrontporch;
        __u32   vsync;
        __u32   vbackporch;
        __u32   il_vfrontporch;
        __u32   il_vsync;
        __u32   il_vbackporch;
        __u32   standards;
        __u32   flags;
        struct v4l2_fract picture_aspect;
        __u8    cea861_vic;
```

```
        __u8    hdmi_vic;
        __u8    reserved[46];
} __attribute__ ((packed));

/* Interlaced or progressive format */
#define V4L2_DV_PROGRESSIVE     0
#define V4L2_DV_INTERLACED      1

/* Polarities. If bit is not set, it is assumed to be negative␣
↪polarity */
#define V4L2_DV_VSYNC_POS_POL   0x00000001
#define V4L2_DV_HSYNC_POS_POL   0x00000002

/* Timings standards */
#define V4L2_DV_BT_STD_CEA861   (1 << 0)  /* CEA-861 Digital TV␣
↪Profile */
#define V4L2_DV_BT_STD_DMT      (1 << 1)  /* VESA Discrete Monitor␣
↪Timings */
#define V4L2_DV_BT_STD_CVT      (1 << 2)  /* VESA Coordinated Video␣
↪Timings */
#define V4L2_DV_BT_STD_GTF      (1 << 3)  /* VESA Generalized␣
↪Timings Formula */
#define V4L2_DV_BT_STD_SDI      (1 << 4)  /* SDI Timings */

/* Flags */

/*
 * CVT/GTF specific: timing uses reduced blanking (CVT) or the␣
↪'Secondary
 * GTF' curve (GTF). In both cases the horizontal and/or vertical␣
↪blanking
 * intervals are reduced, allowing a higher resolution over the same
 * bandwidth. This is a read-only flag.
 */
#define V4L2_DV_FL_REDUCED_BLANKING             (1 << 0)
/*
 * CEA-861 specific: set for CEA-861 formats with a framerate of a␣
↪multiple
 * of six. These formats can be optionally played at 1 / 1.001␣
↪speed.
 * This is a read-only flag.
 */
#define V4L2_DV_FL_CAN_REDUCE_FPS               (1 << 1)
/*
 * CEA-861 specific: only valid for video transmitters, the flag is␣
↪cleared
 * by receivers.
 * If the framerate of the format is a multiple of six, then the␣
↪pixelclock
 * used to set up the transmitter is divided by 1.001 to make it␣
↪compatible
```

```
 * with 60 Hz based standards such as NTSC and PAL-M that use a␣
↪framerate of
 * 29.97 Hz. Otherwise this flag is cleared. If the transmitter␣
↪can't generate
 * such frequencies, then the flag will also be cleared.
 */
#define V4L2_DV_FL_REDUCED_FPS                  (1 << 2)
/*
 * Specific to interlaced formats: if set, then field 1 is really␣
↪one half-line
 * longer and field 2 is really one half-line shorter, so each␣
↪field has
 * exactly the same number of half-lines. Whether half-lines can be␣
↪detected
 * or used depends on the hardware.
 */
#define V4L2_DV_FL_HALF_LINE                    (1 << 3)
/*
 * If set, then this is a Consumer Electronics (CE) video format.␣
↪Such formats
 * differ from other formats (commonly called IT formats) in that␣
↪if RGB
 * encoding is used then by default the RGB values use limited␣
↪range (i.e.
 * use the range 16-235) as opposed to 0-255. All formats defined␣
↪in CEA-861
 * except for the 640x480 format are CE formats.
 */
#define V4L2_DV_FL_IS_CE_VIDEO                  (1 << 4)
/* Some formats like SMPTE-125M have an interlaced signal with a odd
 * total height. For these formats, if this flag is set, the first
 * field has the extra line. If not, it is the second field.
 */
#define V4L2_DV_FL_FIRST_FIELD_EXTRA_LINE       (1 << 5)
/*
 * If set, then the picture_aspect field is valid. Otherwise assume␣
↪that the
 * pixels are square, so the picture aspect ratio is the same as␣
↪the width to
 * height ratio.
 */
#define V4L2_DV_FL_HAS_PICTURE_ASPECT           (1 << 6)
/*
 * If set, then the cea861_vic field is valid and contains the Video
 * Identification Code as per the CEA-861 standard.
 */
#define V4L2_DV_FL_HAS_CEA861_VIC               (1 << 7)
/*
 * If set, then the hdmi_vic field is valid and contains the Video
 * Identification Code as per the HDMI standard (HDMI Vendor␣
↪Specific
```

```
 * InfoFrame).
 */
#define V4L2_DV_FL_HAS_HDMI_VIC                 (1 << 8)
/*
 * CEA-861 specific: only valid for video receivers.
 * If set, then HW can detect the difference between regular FPS and
 * 1000/1001 FPS. Note: This flag is only valid for HDMI VIC codes␣
↪with
 * the V4L2_DV_FL_CAN_REDUCE_FPS flag set.
 */
#define V4L2_DV_FL_CAN_DETECT_REDUCED_FPS       (1 << 9)

/* A few useful defines to calculate the total blanking and frame␣
↪sizes */
#define V4L2_DV_BT_BLANKING_WIDTH(bt) \
        ((bt)->hfrontporch + (bt)->hsync + (bt)->hbackporch)
#define V4L2_DV_BT_FRAME_WIDTH(bt) \
        ((bt)->width + V4L2_DV_BT_BLANKING_WIDTH(bt))
#define V4L2_DV_BT_BLANKING_HEIGHT(bt) \
        ((bt)->vfrontporch + (bt)->vsync + (bt)->vbackporch + \
         ((bt)->interlaced ? \
          ((bt)->il_vfrontporch + (bt)->il_vsync + (bt)->il_
↪vbackporch) : 0))
#define V4L2_DV_BT_FRAME_HEIGHT(bt) \
        ((bt)->height + V4L2_DV_BT_BLANKING_HEIGHT(bt))

/** struct v4l2_dv_timings - DV timings
 * @type:       the type of the timings
 * @bt: BT656/1120 timings
 */
struct v4l2_dv_timings {
        __u32 type;
        union {
                struct v4l2_bt_timings  bt;
                __u32   reserved[32];
        };
} __attribute__ ((packed));

/* Values for the type field */
#define V4L2_DV_BT_656_1120     0       /* BT.656/1120 timing type␣
↪*/

/** struct v4l2_enum_dv_timings - DV timings enumeration
 * @index:      enumeration index
 * @pad:        the pad number for which to enumerate timings (used␣
↪with
 *              v4l-subdev nodes only)
 * @reserved:   must be zeroed
 * @timings:    the timings for the given index
 */
struct v4l2_enum_dv_timings {
```

```
        __u32 index;
        __u32 pad;
        __u32 reserved[2];
        struct v4l2_dv_timings timings;
};

/** struct v4l2_bt_timings_cap - BT.656/BT.1120 timing capabilities
 * @min_width:          width in pixels
 * @max_width:          width in pixels
 * @min_height:         height in lines
 * @max_height:         height in lines
 * @min_pixelclock:     Pixel clock in HZ. Ex. 74.25MHz->74250000
 * @max_pixelclock:     Pixel clock in HZ. Ex. 74.25MHz->74250000
 * @standards:          Supported standards
 * @capabilities:       Supported capabilities
 * @reserved:           Must be zeroed
 */
struct v4l2_bt_timings_cap {
        __u32   min_width;
        __u32   max_width;
        __u32   min_height;
        __u32   max_height;
        __u64   min_pixelclock;
        __u64   max_pixelclock;
        __u32   standards;
        __u32   capabilities;
        __u32   reserved[16];
} __attribute__ ((packed));

/* Supports interlaced formats */
#define V4L2_DV_BT_CAP_INTERLACED       (1 << 0)
/* Supports progressive formats */
#define V4L2_DV_BT_CAP_PROGRESSIVE      (1 << 1)
/* Supports CVT/GTF reduced blanking */
#define V4L2_DV_BT_CAP_REDUCED_BLANKING (1 << 2)
/* Supports custom formats */
#define V4L2_DV_BT_CAP_CUSTOM           (1 << 3)

/** struct v4l2_dv_timings_cap - DV timings capabilities
 * @type:       the type of the timings (same as in struct v4l2_dv_
↪timings)
 * @pad:        the pad number for which to query capabilities␣
↪(used with
 *              v4l-subdev nodes only)
 * @bt:         the BT656/1120 timings capabilities
 */
struct v4l2_dv_timings_cap {
        __u32 type;
        __u32 pad;
        __u32 reserved[2];
        union {
```

```
                struct v4l2_bt_timings_cap bt;
                __u32 raw_data[32];
        };
};

/*
 *      V I D E O   I N P U T S
 */
struct v4l2_input {
        __u32        index;             /*  Which input */
        __u8         name[32];          /*  Label */
        __u32        type;              /*  Type of input */
        __u32        audioset;          /*  Associated audios␣
↪(bitfield) */
        __u32        tuner;             /*  Tuner index */
        v4l2_std_id  std;
        __u32        status;
        __u32        capabilities;
        __u32        reserved[3];
};

/*  Values for the 'type' field */
#define V4L2_INPUT_TYPE_TUNER           1
#define V4L2_INPUT_TYPE_CAMERA          2
#define V4L2_INPUT_TYPE_TOUCH           3

/* field 'status' - general */
#define V4L2_IN_ST_NO_POWER    0x00000001  /* Attached device is␣
↪off */
#define V4L2_IN_ST_NO_SIGNAL   0x00000002
#define V4L2_IN_ST_NO_COLOR    0x00000004

/* field 'status' - sensor orientation */
/* If sensor is mounted upside down set both bits */
#define V4L2_IN_ST_HFLIP       0x00000010 /* Frames are flipped␣
↪horizontally */
#define V4L2_IN_ST_VFLIP       0x00000020 /* Frames are flipped␣
↪vertically */

/* field 'status' - analog */
#define V4L2_IN_ST_NO_H_LOCK   0x00000100  /* No horizontal sync␣
↪lock */
#define V4L2_IN_ST_COLOR_KILL  0x00000200  /* Color killer is␣
↪active */
#define V4L2_IN_ST_NO_V_LOCK   0x00000400  /* No vertical sync lock␣
↪*/
#define V4L2_IN_ST_NO_STD_LOCK 0x00000800  /* No standard format␣
↪lock */

/* field 'status' - digital */
#define V4L2_IN_ST_NO_SYNC     0x00010000  /* No synchronization␣
```

```
↪lock */
#define V4L2_IN_ST_NO_EQU       0x00020000  /* No equalizer lock */
#define V4L2_IN_ST_NO_CARRIER   0x00040000  /* Carrier recovery␣
↪failed */

/* field 'status' - VCR and set-top box */
#define V4L2_IN_ST_MACROVISION 0x01000000  /* Macrovision detected␣
↪*/
#define V4L2_IN_ST_NO_ACCESS   0x02000000  /* Conditional access␣
↪denied */
#define V4L2_IN_ST_VTR         0x04000000  /* VTR time constant */

/* capabilities flags */
#define V4L2_IN_CAP_DV_TIMINGS          0x00000002 /* Supports S_DV_
↪TIMINGS */
#define V4L2_IN_CAP_CUSTOM_TIMINGS      V4L2_IN_CAP_DV_TIMINGS /*␣
↪For compatibility */
#define V4L2_IN_CAP_STD                 0x00000004 /* Supports S_
↪STD */
#define V4L2_IN_CAP_NATIVE_SIZE         0x00000008 /* Supports␣
↪setting native size */

/*
 *      V I D E O   O U T P U T S
 */
struct v4l2_output {
        __u32        index;             /*  Which output */
        __u8         name[32];          /*  Label */
        __u32        type;              /*  Type of output */
        __u32        audioset;          /*  Associated audios␣
↪(bitfield) */
        __u32        modulator;         /*  Associated modulator */
        v4l2_std_id  std;
        __u32        capabilities;
        __u32        reserved[3];
};
/*  Values for the 'type' field */
#define V4L2_OUTPUT_TYPE_MODULATOR              1
#define V4L2_OUTPUT_TYPE_ANALOG                 2
#define V4L2_OUTPUT_TYPE_ANALOGVGAOVERLAY       3

/* capabilities flags */
#define V4L2_OUT_CAP_DV_TIMINGS         0x00000002 /* Supports S_DV_
↪TIMINGS */
#define V4L2_OUT_CAP_CUSTOM_TIMINGS     V4L2_OUT_CAP_DV_TIMINGS /*␣
↪For compatibility */
#define V4L2_OUT_CAP_STD                0x00000004 /* Supports S_
↪STD */
#define V4L2_OUT_CAP_NATIVE_SIZE        0x00000008 /* Supports␣
↪setting native size */
```

```
/*
 *      C O N T R O L S
 */
struct v4l2_control {
        __u32                id;
        __s32                value;
};

struct v4l2_ext_control {
        __u32 id;
        __u32 size;
        __u32 reserved2[1];
        union {
                __s32 value;
                __s64 value64;
                char __user *string;
                __u8 __user *p_u8;
                __u16 __user *p_u16;
                __u32 __user *p_u32;
                struct v4l2_area __user *p_area;
                void __user *ptr;
        };
} __attribute__ ((packed));

struct v4l2_ext_controls {
        union {
#ifndef __KERNEL__
                __u32 ctrl_class;
#endif
                __u32 which;
        };
        __u32 count;
        __u32 error_idx;
        __s32 request_fd;
        __u32 reserved[1];
        struct v4l2_ext_control *controls;
};

#define V4L2_CTRL_ID_MASK         (0x0fffffff)
#ifndef __KERNEL__
#define V4L2_CTRL_ID2CLASS(id)    ((id) & 0x0fff0000UL)
#endif
#define V4L2_CTRL_ID2WHICH(id)    ((id) & 0x0fff0000UL)
#define V4L2_CTRL_DRIVER_PRIV(id) (((id) & 0xffff) >= 0x1000)
#define V4L2_CTRL_MAX_DIMS        (4)
#define V4L2_CTRL_WHICH_CUR_VAL   0
#define V4L2_CTRL_WHICH_DEF_VAL   0x0f000000
#define V4L2_CTRL_WHICH_REQUEST_VAL 0x0f010000

enum v4l2_ctrl_type {
        V4L2_CTRL_TYPE_INTEGER       = 1,
```

```
        V4L2_CTRL_TYPE_BOOLEAN       = 2,
        V4L2_CTRL_TYPE_MENU          = 3,
        V4L2_CTRL_TYPE_BUTTON        = 4,
        V4L2_CTRL_TYPE_INTEGER64     = 5,
        V4L2_CTRL_TYPE_CTRL_CLASS    = 6,
        V4L2_CTRL_TYPE_STRING        = 7,
        V4L2_CTRL_TYPE_BITMASK       = 8,
        V4L2_CTRL_TYPE_INTEGER_MENU  = 9,

        /* Compound types are >= 0x0100 */
        V4L2_CTRL_COMPOUND_TYPES     = 0x0100,
        V4L2_CTRL_TYPE_U8            = 0x0100,
        V4L2_CTRL_TYPE_U16           = 0x0101,
        V4L2_CTRL_TYPE_U32           = 0x0102,
        V4L2_CTRL_TYPE_AREA          = 0x0106,
};

/*  Used in the VIDIOC_QUERYCTRL ioctl for querying controls */
struct v4l2_queryctrl {
        __u32                id;
        __u32                type;      /* enum v4l2_ctrl_type */
        __u8                 name[32];  /* Whatever */
        __s32                minimum;   /* Note signedness */
        __s32                maximum;
        __s32                step;
        __s32                default_value;
        __u32                flags;
        __u32                reserved[2];
};

/*  Used in the VIDIOC_QUERY_EXT_CTRL ioctl for querying extended␣
↪controls */
struct v4l2_query_ext_ctrl {
        __u32                id;
        __u32                type;
        char                 name[32];
        __s64                minimum;
        __s64                maximum;
        __u64                step;
        __s64                default_value;
        __u32                flags;
        __u32                elem_size;
        __u32                elems;
        __u32                nr_of_dims;
        __u32                dims[V4L2_CTRL_MAX_DIMS];
        __u32                reserved[32];
};

/*  Used in the VIDIOC_QUERYMENU ioctl for querying menu items */
struct v4l2_querymenu {
        __u32           id;
```

```
        __u32           index;
        union {
                __u8    name[32];       /* Whatever */
                __s64   value;
        };
        __u32           reserved;
} __attribute__ ((packed));

/*  Control flags  */
#define V4L2_CTRL_FLAG_DISABLED         0x0001
#define V4L2_CTRL_FLAG_GRABBED          0x0002
#define V4L2_CTRL_FLAG_READ_ONLY        0x0004
#define V4L2_CTRL_FLAG_UPDATE           0x0008
#define V4L2_CTRL_FLAG_INACTIVE         0x0010
#define V4L2_CTRL_FLAG_SLIDER           0x0020
#define V4L2_CTRL_FLAG_WRITE_ONLY       0x0040
#define V4L2_CTRL_FLAG_VOLATILE         0x0080
#define V4L2_CTRL_FLAG_HAS_PAYLOAD      0x0100
#define V4L2_CTRL_FLAG_EXECUTE_ON_WRITE 0x0200
#define V4L2_CTRL_FLAG_MODIFY_LAYOUT    0x0400

/*  Query flags, to be ORed with the control ID */
#define V4L2_CTRL_FLAG_NEXT_CTRL        0x80000000
#define V4L2_CTRL_FLAG_NEXT_COMPOUND    0x40000000

/*  User-class control IDs defined by V4L2 */
#define V4L2_CID_MAX_CTRLS              1024
/*  IDs reserved for driver specific controls */
#define V4L2_CID_PRIVATE_BASE           0x08000000

/*
 *      T U N I N G
 */
struct v4l2_tuner {
        __u32                   index;
        __u8                    name[32];
        __u32                   type;   /* enum v4l2_tuner_type */
        __u32                   capability;
        __u32                   rangelow;
        __u32                   rangehigh;
        __u32                   rxsubchans;
        __u32                   audmode;
        __s32                   signal;
        __s32                   afc;
        __u32                   reserved[4];
};

struct v4l2_modulator {
        __u32                   index;
        __u8                    name[32];
        __u32                   capability;
```

```
        __u32                   rangelow;
        __u32                   rangehigh;
        __u32                   txsubchans;
        __u32                   type;   /* enum v4l2_tuner_type */
        __u32                   reserved[3];
};

/*  Flags for the 'capability' field */
#define V4L2_TUNER_CAP_LOW              0x0001
#define V4L2_TUNER_CAP_NORM             0x0002
#define V4L2_TUNER_CAP_HWSEEK_BOUNDED   0x0004
#define V4L2_TUNER_CAP_HWSEEK_WRAP      0x0008
#define V4L2_TUNER_CAP_STEREO           0x0010
#define V4L2_TUNER_CAP_LANG2            0x0020
#define V4L2_TUNER_CAP_SAP              0x0020
#define V4L2_TUNER_CAP_LANG1            0x0040
#define V4L2_TUNER_CAP_RDS              0x0080
#define V4L2_TUNER_CAP_RDS_BLOCK_IO     0x0100
#define V4L2_TUNER_CAP_RDS_CONTROLS     0x0200
#define V4L2_TUNER_CAP_FREQ_BANDS       0x0400
#define V4L2_TUNER_CAP_HWSEEK_PROG_LIM  0x0800
#define V4L2_TUNER_CAP_1HZ              0x1000

/*  Flags for the 'rxsubchans' field */
#define V4L2_TUNER_SUB_MONO             0x0001
#define V4L2_TUNER_SUB_STEREO           0x0002
#define V4L2_TUNER_SUB_LANG2            0x0004
#define V4L2_TUNER_SUB_SAP              0x0004
#define V4L2_TUNER_SUB_LANG1            0x0008
#define V4L2_TUNER_SUB_RDS              0x0010

/*  Values for the 'audmode' field */
#define V4L2_TUNER_MODE_MONO            0x0000
#define V4L2_TUNER_MODE_STEREO          0x0001
#define V4L2_TUNER_MODE_LANG2           0x0002
#define V4L2_TUNER_MODE_SAP             0x0002
#define V4L2_TUNER_MODE_LANG1           0x0003
#define V4L2_TUNER_MODE_LANG1_LANG2     0x0004

struct v4l2_frequency {
        __u32   tuner;
        __u32   type;   /* enum v4l2_tuner_type */
        __u32   frequency;
        __u32   reserved[8];
};

#define V4L2_BAND_MODULATION_VSB        (1 << 1)
#define V4L2_BAND_MODULATION_FM         (1 << 2)
#define V4L2_BAND_MODULATION_AM         (1 << 3)

struct v4l2_frequency_band {
```

```
        __u32   tuner;
        __u32   type;   /* enum v4l2_tuner_type */
        __u32   index;
        __u32   capability;
        __u32   rangelow;
        __u32   rangehigh;
        __u32   modulation;
        __u32   reserved[9];
};

struct v4l2_hw_freq_seek {
        __u32   tuner;
        __u32   type;   /* enum v4l2_tuner_type */
        __u32   seek_upward;
        __u32   wrap_around;
        __u32   spacing;
        __u32   rangelow;
        __u32   rangehigh;
        __u32   reserved[5];
};

/*
 *      R D S
 */

struct v4l2_rds_data {
        __u8    lsb;
        __u8    msb;
        __u8    block;
} __attribute__ ((packed));

#define V4L2_RDS_BLOCK_MSK       0x7
#define V4L2_RDS_BLOCK_A         0
#define V4L2_RDS_BLOCK_B         1
#define V4L2_RDS_BLOCK_C         2
#define V4L2_RDS_BLOCK_D         3
#define V4L2_RDS_BLOCK_C_ALT     4
#define V4L2_RDS_BLOCK_INVALID   7

#define V4L2_RDS_BLOCK_CORRECTED 0x40
#define V4L2_RDS_BLOCK_ERROR     0x80

/*
 *      A U D I O
 */
struct v4l2_audio {
        __u32   index;
        __u8    name[32];
        __u32   capability;
        __u32   mode;
        __u32   reserved[2];
```

```
};

/*  Flags for the 'capability' field */
#define V4L2_AUDCAP_STEREO              0x00001
#define V4L2_AUDCAP_AVL                 0x00002

/*  Flags for the 'mode' field */
#define V4L2_AUDMODE_AVL                0x00001

struct v4l2_audioout {
        __u32   index;
        __u8    name[32];
        __u32   capability;
        __u32   mode;
        __u32   reserved[2];
};

/*
 *      M P E G   S E R V I C E S
 */
#if 1
#define V4L2_ENC_IDX_FRAME_I    (0)
#define V4L2_ENC_IDX_FRAME_P    (1)
#define V4L2_ENC_IDX_FRAME_B    (2)
#define V4L2_ENC_IDX_FRAME_MASK (0xf)

struct v4l2_enc_idx_entry {
        __u64 offset;
        __u64 pts;
        __u32 length;
        __u32 flags;
        __u32 reserved[2];
};

#define V4L2_ENC_IDX_ENTRIES (64)
struct v4l2_enc_idx {
        __u32 entries;
        __u32 entries_cap;
        __u32 reserved[4];
        struct v4l2_enc_idx_entry entry[V4L2_ENC_IDX_ENTRIES];
};

#define V4L2_ENC_CMD_START      (0)
#define V4L2_ENC_CMD_STOP       (1)
#define V4L2_ENC_CMD_PAUSE      (2)
#define V4L2_ENC_CMD_RESUME     (3)

/* Flags for V4L2_ENC_CMD_STOP */
#define V4L2_ENC_CMD_STOP_AT_GOP_END    (1 << 0)

struct v4l2_encoder_cmd {
```

```
        __u32 cmd;
        __u32 flags;
        union {
                struct {
                        __u32 data[8];
                } raw;
        };
};

/* Decoder commands */
#define V4L2_DEC_CMD_START       (0)
#define V4L2_DEC_CMD_STOP        (1)
#define V4L2_DEC_CMD_PAUSE       (2)
#define V4L2_DEC_CMD_RESUME      (3)
#define V4L2_DEC_CMD_FLUSH       (4)

/* Flags for V4L2_DEC_CMD_START */
#define V4L2_DEC_CMD_START_MUTE_AUDIO   (1 << 0)

/* Flags for V4L2_DEC_CMD_PAUSE */
#define V4L2_DEC_CMD_PAUSE_TO_BLACK     (1 << 0)

/* Flags for V4L2_DEC_CMD_STOP */
#define V4L2_DEC_CMD_STOP_TO_BLACK      (1 << 0)
#define V4L2_DEC_CMD_STOP_IMMEDIATELY   (1 << 1)

/* Play format requirements (returned by the driver): */

/* The decoder has no special format requirements */
#define V4L2_DEC_START_FMT_NONE         (0)
/* The decoder requires full GOPs */
#define V4L2_DEC_START_FMT_GOP          (1)

/* The structure must be zeroed before use by the application
   This ensures it can be extended safely in the future. */
struct v4l2_decoder_cmd {
        __u32 cmd;
        __u32 flags;
        union {
                struct {
                        __u64 pts;
                } stop;

                struct {
                        /* 0 or 1000 specifies normal speed,
                           1 specifies forward single stepping,
                           -1 specifies backward single stepping,
                           >1: playback at speed/1000 of the normal␣
↪speed,
                           <-1: reverse playback at (-speed/1000)␣
↪of the normal speed. */
```

```
                        __s32 speed;
                        __u32 format;
                } start;

                struct {
                        __u32 data[16];
                } raw;
        };
};
#endif

/*
 *      D A T A   S E R V I C E S   ( V B I )
 *
 *      Data services API by Michael Schimek
 */

/* Raw VBI */
struct v4l2_vbi_format {
        __u32   sampling_rate;          /* in 1 Hz */
        __u32   offset;
        __u32   samples_per_line;
        __u32   sample_format;          /* V4L2_PIX_FMT_* */
        __s32   start[2];
        __u32   count[2];
        __u32   flags;                  /* V4L2_VBI_* */
        __u32   reserved[2];            /* must be zero */
};

/*  VBI flags  */
#define V4L2_VBI_UNSYNC         (1 << 0)
#define V4L2_VBI_INTERLACED     (1 << 1)

/* ITU-R start lines for each field */
#define V4L2_VBI_ITU_525_F1_START (1)
#define V4L2_VBI_ITU_525_F2_START (264)
#define V4L2_VBI_ITU_625_F1_START (1)
#define V4L2_VBI_ITU_625_F2_START (314)

/* Sliced VBI
 *
 *    This implements is a proposal V4L2 API to allow SLICED VBI
 * required for some hardware encoders. It should change without
 * notice in the definitive implementation.
 */

struct v4l2_sliced_vbi_format {
        __u16   service_set;
        /* service_lines[0][...] specifies lines 0-23 (1-23 used)␣
↪of the first field
           service_lines[1][...] specifies lines 0-23 (1-23 used)␣
```

```
↪of the second field
                                (equals frame lines 313-336 for␣
↪625 line video
                                 standards, 263-286 for 525 line␣
↪standards) */
        __u16   service_lines[2][24];
        __u32   io_size;
        __u32   reserved[2];            /* must be zero */
};

/* Teletext World System Teletext
   (WST), defined on ITU-R BT.653-2 */
#define V4L2_SLICED_TELETEXT_B          (0x0001)
/* Video Program System, defined on ETS 300 231*/
#define V4L2_SLICED_VPS                 (0x0400)
/* Closed Caption, defined on EIA-608 */
#define V4L2_SLICED_CAPTION_525         (0x1000)
/* Wide Screen System, defined on ITU-R BT1119.1 */
#define V4L2_SLICED_WSS_625             (0x4000)

#define V4L2_SLICED_VBI_525             (V4L2_SLICED_CAPTION_525)
#define V4L2_SLICED_VBI_625             (V4L2_SLICED_TELETEXT_B |␣
↪V4L2_SLICED_VPS | V4L2_SLICED_WSS_625)

struct v4l2_sliced_vbi_cap {
        __u16   service_set;
        /* service_lines[0][...] specifies lines 0-23 (1-23 used)␣
↪of the first field
           service_lines[1][...] specifies lines 0-23 (1-23 used)␣
↪of the second field
                                (equals frame lines 313-336 for␣
↪625 line video
                                 standards, 263-286 for 525 line␣
↪standards) */
        __u16   service_lines[2][24];
        __u32   type;           /* enum v4l2_buf_type */
        __u32   reserved[3];    /* must be 0 */
};

struct v4l2_sliced_vbi_data {
        __u32   id;
        __u32   field;          /* 0: first field, 1: second field␣
↪*/
        __u32   line;           /* 1-23 */
        __u32   reserved;       /* must be 0 */
        __u8    data[48];
};

/*
 * Sliced VBI data inserted into MPEG Streams
 */
```

```
/*
 * V4L2_MPEG_STREAM_VBI_FMT_IVTV:
 *
 * Structure of payload contained in an MPEG 2 Private Stream 1 PES␣
↪Packet in an
 * MPEG-2 Program Pack that contains V4L2_MPEG_STREAM_VBI_FMT_IVTV␣
↪Sliced VBI
 * data
 *
 * Note, the MPEG-2 Program Pack and Private Stream 1 PES packet␣
↪header
 * definitions are not included here.  See the MPEG-2␣
↪specifications for details
 * on these headers.
 */

/* Line type IDs */
#define V4L2_MPEG_VBI_IVTV_TELETEXT_B     (1)
#define V4L2_MPEG_VBI_IVTV_CAPTION_525    (4)
#define V4L2_MPEG_VBI_IVTV_WSS_625        (5)
#define V4L2_MPEG_VBI_IVTV_VPS            (7)

struct v4l2_mpeg_vbi_itv0_line {
        __u8 id;        /* One of V4L2_MPEG_VBI_IVTV_* above */
        __u8 data[42];  /* Sliced VBI data for the line */
} __attribute__ ((packed));

struct v4l2_mpeg_vbi_itv0 {
        __le32 linemask[2]; /* Bitmasks of VBI service lines␣
↪present */
        struct v4l2_mpeg_vbi_itv0_line line[35];
} __attribute__ ((packed));

struct v4l2_mpeg_vbi_ITV0 {
        struct v4l2_mpeg_vbi_itv0_line line[36];
} __attribute__ ((packed));

#define V4L2_MPEG_VBI_IVTV_MAGIC0       "itv0"
#define V4L2_MPEG_VBI_IVTV_MAGIC1       "ITV0"

struct v4l2_mpeg_vbi_fmt_ivtv {
        __u8 magic[4];
        union {
                struct v4l2_mpeg_vbi_itv0 itv0;
                struct v4l2_mpeg_vbi_ITV0 ITV0;
        };
} __attribute__ ((packed));

/*
 *      A G G R E G A T E   S T R U C T U R E S
```

```
 */

/**
 * struct v4l2_plane_pix_format - additional, per-plane format␣
↪definition
 * @sizeimage:          maximum size in bytes required for data,␣
↪for which
 *                      this plane will be used
 * @bytesperline:       distance in bytes between the leftmost␣
↪pixels in two
 *                      adjacent lines
 */
struct v4l2_plane_pix_format {
        __u32           sizeimage;
        __u32           bytesperline;
        __u16           reserved[6];
} __attribute__ ((packed));

/**
 * struct v4l2_pix_format_mplane - multiplanar format definition
 * @width:              image width in pixels
 * @height:             image height in pixels
 * @pixelformat:        little endian four character code (fourcc)
 * @field:              enum v4l2_field; field order (for␣
↪interlaced video)
 * @colorspace:         enum v4l2_colorspace; supplemental to␣
↪pixelformat
 * @plane_fmt:          per-plane information
 * @num_planes:         number of planes for this format
 * @flags:              format flags (V4L2_PIX_FMT_FLAG_*)
 * @ycbcr_enc:          enum v4l2_ycbcr_encoding, Y'CbCr encoding
 * @quantization:       enum v4l2_quantization, colorspace␣
↪quantization
 * @xfer_func:          enum v4l2_xfer_func, colorspace transfer␣
↪function
 */
struct v4l2_pix_format_mplane {
        __u32                           width;
        __u32                           height;
        __u32                           pixelformat;
        __u32                           field;
        __u32                           colorspace;

        struct v4l2_plane_pix_format    plane_fmt[VIDEO_MAX_PLANES];
        __u8                            num_planes;
        __u8                            flags;
         union {
                __u8                            ycbcr_enc;
                __u8                            hsv_enc;
        };
        __u8                            quantization;
```

```
        __u8                            xfer_func;
        __u8                            reserved[7];
} __attribute__ ((packed));

/**
 * struct v4l2_sdr_format - SDR format definition
 * @pixelformat:        little endian four character code (fourcc)
 * @buffersize:         maximum size in bytes required for data
 */
struct v4l2_sdr_format {
        __u32                           pixelformat;
        __u32                           buffersize;
        __u8                            reserved[24];
} __attribute__ ((packed));

/**
 * struct v4l2_meta_format - metadata format definition
 * @dataformat:         little endian four character code (fourcc)
 * @buffersize:         maximum size in bytes required for data
 */
struct v4l2_meta_format {
        __u32                           dataformat;
        __u32                           buffersize;
} __attribute__ ((packed));

/**
 * struct v4l2_format - stream data format
 * @type:       enum v4l2_buf_type; type of the data stream
 * @pix:        definition of an image format
 * @pix_mp:     definition of a multiplanar image format
 * @win:        definition of an overlaid image
 * @vbi:        raw VBI capture or output parameters
 * @sliced:     sliced VBI capture or output parameters
 * @raw_data:   placeholder for future extensions and custom formats
 */
struct v4l2_format {
        __u32    type;
        union {
                struct v4l2_pix_format          pix;     /* V4L2_
→BUF_TYPE_VIDEO_CAPTURE */
                struct v4l2_pix_format_mplane   pix_mp;  /* V4L2_
→BUF_TYPE_VIDEO_CAPTURE_MPLANE */
                struct v4l2_window              win;     /* V4L2_
→BUF_TYPE_VIDEO_OVERLAY */
                struct v4l2_vbi_format          vbi;     /* V4L2_
→BUF_TYPE_VBI_CAPTURE */
                struct v4l2_sliced_vbi_format   sliced;  /* V4L2_
→BUF_TYPE_SLICED_VBI_CAPTURE */
                struct v4l2_sdr_format          sdr;     /* V4L2_
→BUF_TYPE_SDR_CAPTURE */
                struct v4l2_meta_format         meta;    /* V4L2_
```

```
↪BUF_TYPE_META_CAPTURE */
                __u8    raw_data[200];                  /*␣
↪user-defined */
        } fmt;
};

/*      Stream type-dependent parameters
 */
struct v4l2_streamparm {
        __u32    type;                  /* enum v4l2_buf_type */
        union {
                struct v4l2_captureparm capture;
                struct v4l2_outputparm  output;
                __u8    raw_data[200];  /* user-defined */
        } parm;
};

/*
 *      E V E N T S
 */

#define V4L2_EVENT_ALL                          0
#define V4L2_EVENT_VSYNC                        1
#define V4L2_EVENT_EOS                          2
#define V4L2_EVENT_CTRL                         3
#define V4L2_EVENT_FRAME_SYNC                   4
#define V4L2_EVENT_SOURCE_CHANGE                5
#define V4L2_EVENT_MOTION_DET                   6
#define V4L2_EVENT_PRIVATE_START                0x08000000

/* Payload for V4L2_EVENT_VSYNC */
struct v4l2_event_vsync {
        /* Can be V4L2_FIELD_ANY, _NONE, _TOP or _BOTTOM */
        __u8 field;
} __attribute__ ((packed));

/* Payload for V4L2_EVENT_CTRL */
#define V4L2_EVENT_CTRL_CH_VALUE                (1 << 0)
#define V4L2_EVENT_CTRL_CH_FLAGS                (1 << 1)
#define V4L2_EVENT_CTRL_CH_RANGE                (1 << 2)

struct v4l2_event_ctrl {
        __u32 changes;
        __u32 type;
        union {
                __s32 value;
                __s64 value64;
        };
        __u32 flags;
        __s32 minimum;
        __s32 maximum;
```

```
        __s32 step;
        __s32 default_value;
};

struct v4l2_event_frame_sync {
        __u32 frame_sequence;
};

#define V4L2_EVENT_SRC_CH_RESOLUTION            (1 << 0)

struct v4l2_event_src_change {
        __u32 changes;
};

#define V4L2_EVENT_MD_FL_HAVE_FRAME_SEQ (1 << 0)

/**
 * struct v4l2_event_motion_det - motion detection event
 * @flags:             if V4L2_EVENT_MD_FL_HAVE_FRAME_SEQ is set,␣
↪then the
 *                     frame_sequence field is valid.
 * @frame_sequence:    the frame sequence number associated with␣
↪this event.
 * @region_mask:       which regions detected motion.
 */
struct v4l2_event_motion_det {
        __u32 flags;
        __u32 frame_sequence;
        __u32 region_mask;
};

struct v4l2_event {
        __u32                           type;
        union {
                struct v4l2_event_vsync         vsync;
                struct v4l2_event_ctrl          ctrl;
                struct v4l2_event_frame_sync    frame_sync;
                struct v4l2_event_src_change    src_change;
                struct v4l2_event_motion_det    motion_det;
                __u8                            data[64];
        } u;
        __u32                           pending;
        __u32                           sequence;
#ifdef __KERNEL__
        struct __kernel_timespec        timestamp;
#else
        struct timespec                 timestamp;
#endif
        __u32                           id;
        __u32                           reserved[8];
};
```

```
#define V4L2_EVENT_SUB_FL_SEND_INITIAL          (1 << 0)
#define V4L2_EVENT_SUB_FL_ALLOW_FEEDBACK        (1 << 1)

struct v4l2_event_subscription {
        __u32                           type;
        __u32                           id;
        __u32                           flags;
        __u32                           reserved[5];
};

/*
 *      A D V A N C E D   D E B U G G I N G
 *
 *      NOTE: EXPERIMENTAL API, NEVER RELY ON THIS IN APPLICATIONS!
 *      FOR DEBUGGING, TESTING AND INTERNAL USE ONLY!
 */

/* VIDIOC_DBG_G_REGISTER and VIDIOC_DBG_S_REGISTER */

#define V4L2_CHIP_MATCH_BRIDGE      0  /* Match against chip ID on␣
↪the bridge (0 for the bridge) */
#define V4L2_CHIP_MATCH_SUBDEV      4  /* Match against subdev␣
↪index */

/* The following four defines are no longer in use */
#define V4L2_CHIP_MATCH_HOST V4L2_CHIP_MATCH_BRIDGE
#define V4L2_CHIP_MATCH_I2C_DRIVER  1  /* Match against I2C driver␣
↪name */
#define V4L2_CHIP_MATCH_I2C_ADDR    2  /* Match against I2C 7-bit␣
↪address */
#define V4L2_CHIP_MATCH_AC97        3  /* Match against ancillary␣
↪AC97 chip */

struct v4l2_dbg_match {
        __u32 type; /* Match type */
        union {     /* Match this chip, meaning determined by type␣
↪*/
                __u32 addr;
                char name[32];
        };
} __attribute__ ((packed));

struct v4l2_dbg_register {
        struct v4l2_dbg_match match;
        __u32 size;     /* register size in bytes */
        __u64 reg;
        __u64 val;
} __attribute__ ((packed));

#define V4L2_CHIP_FL_READABLE (1 << 0)
```

```
#define V4L2_CHIP_FL_WRITABLE (1 << 1)

/* VIDIOC_DBG_G_CHIP_INFO */
struct v4l2_dbg_chip_info {
        struct v4l2_dbg_match match;
        char name[32];
        __u32 flags;
        __u32 reserved[32];
} __attribute__ ((packed));

/**
 * struct v4l2_create_buffers - VIDIOC_CREATE_BUFS argument
 * @index:      on return, index of the first created buffer
 * @count:      entry: number of requested buffers,
 *              return: number of created buffers
 * @memory:     enum v4l2_memory; buffer memory type
 * @format:     frame format, for which buffers are requested
 * @capabilities: capabilities of this buffer type.
 * @reserved:   future extensions
 */
struct v4l2_create_buffers {
        __u32                   index;
        __u32                   count;
        __u32                   memory;
        struct v4l2_format      format;
        __u32                   capabilities;
        __u32                   reserved[7];
};

/*
 *      I O C T L   C O D E S   F O R   V I D E O   D E V I C E S
 *
 */
#define VIDIOC_QUERYCAP          _IOR('V',  0, struct v4l2_
↪capability)
#define VIDIOC_ENUM_FMT         _IOWR('V',  2, struct v4l2_fmtdesc)
#define VIDIOC_G_FMT            _IOWR('V',  4, struct v4l2_format)
#define VIDIOC_S_FMT            _IOWR('V',  5, struct v4l2_format)
#define VIDIOC_REQBUFS          _IOWR('V',  8, struct v4l2_
↪requestbuffers)
#define VIDIOC_QUERYBUF         _IOWR('V',  9, struct v4l2_buffer)
#define VIDIOC_G_FBUF            _IOR('V', 10, struct v4l2_
↪framebuffer)
#define VIDIOC_S_FBUF            _IOW('V', 11, struct v4l2_
↪framebuffer)
#define VIDIOC_OVERLAY           _IOW('V', 14, int)
#define VIDIOC_QBUF             _IOWR('V', 15, struct v4l2_buffer)
#define VIDIOC_EXPBUF           _IOWR('V', 16, struct v4l2_
↪exportbuffer)
#define VIDIOC_DQBUF            _IOWR('V', 17, struct v4l2_buffer)
#define VIDIOC_STREAMON          _IOW('V', 18, int)
```

```
#define VIDIOC_STREAMOFF         _IOW('V', 19, int)
#define VIDIOC_G_PARM           _IOWR('V', 21, struct v4l2_
↪streamparm)
#define VIDIOC_S_PARM           _IOWR('V', 22, struct v4l2_
↪streamparm)
#define VIDIOC_G_STD             _IOR('V', 23, v4l2_std_id)
#define VIDIOC_S_STD             _IOW('V', 24, v4l2_std_id)
#define VIDIOC_ENUMSTD          _IOWR('V', 25, struct v4l2_standard)
#define VIDIOC_ENUMINPUT        _IOWR('V', 26, struct v4l2_input)
#define VIDIOC_G_CTRL           _IOWR('V', 27, struct v4l2_control)
#define VIDIOC_S_CTRL           _IOWR('V', 28, struct v4l2_control)
#define VIDIOC_G_TUNER          _IOWR('V', 29, struct v4l2_tuner)
#define VIDIOC_S_TUNER           _IOW('V', 30, struct v4l2_tuner)
#define VIDIOC_G_AUDIO           _IOR('V', 33, struct v4l2_audio)
#define VIDIOC_S_AUDIO           _IOW('V', 34, struct v4l2_audio)
#define VIDIOC_QUERYCTRL        _IOWR('V', 36, struct v4l2_
↪queryctrl)
#define VIDIOC_QUERYMENU        _IOWR('V', 37, struct v4l2_
↪querymenu)
#define VIDIOC_G_INPUT           _IOR('V', 38, int)
#define VIDIOC_S_INPUT          _IOWR('V', 39, int)
#define VIDIOC_G_EDID           _IOWR('V', 40, struct v4l2_edid)
#define VIDIOC_S_EDID           _IOWR('V', 41, struct v4l2_edid)
#define VIDIOC_G_OUTPUT          _IOR('V', 46, int)
#define VIDIOC_S_OUTPUT         _IOWR('V', 47, int)
#define VIDIOC_ENUMOUTPUT       _IOWR('V', 48, struct v4l2_output)
#define VIDIOC_G_AUDOUT          _IOR('V', 49, struct v4l2_audioout)
#define VIDIOC_S_AUDOUT          _IOW('V', 50, struct v4l2_audioout)
#define VIDIOC_G_MODULATOR      _IOWR('V', 54, struct v4l2_
↪modulator)
#define VIDIOC_S_MODULATOR       _IOW('V', 55, struct v4l2_
↪modulator)
#define VIDIOC_G_FREQUENCY      _IOWR('V', 56, struct v4l2_
↪frequency)
#define VIDIOC_S_FREQUENCY       _IOW('V', 57, struct v4l2_
↪frequency)
#define VIDIOC_CROPCAP          _IOWR('V', 58, struct v4l2_cropcap)
#define VIDIOC_G_CROP           _IOWR('V', 59, struct v4l2_crop)
#define VIDIOC_S_CROP            _IOW('V', 60, struct v4l2_crop)
#define VIDIOC_G_JPEGCOMP        _IOR('V', 61, struct v4l2_
↪jpegcompression)
#define VIDIOC_S_JPEGCOMP        _IOW('V', 62, struct v4l2_
↪jpegcompression)
#define VIDIOC_QUERYSTD          _IOR('V', 63, v4l2_std_id)
#define VIDIOC_TRY_FMT          _IOWR('V', 64, struct v4l2_format)
#define VIDIOC_ENUMAUDIO        _IOWR('V', 65, struct v4l2_audio)
#define VIDIOC_ENUMAUDOUT       _IOWR('V', 66, struct v4l2_audioout)
#define VIDIOC_G_PRIORITY        _IOR('V', 67, __u32) /* enum v4l2_
↪priority */
#define VIDIOC_S_PRIORITY        _IOW('V', 68, __u32) /* enum v4l2_
↪priority */
```

```
#define VIDIOC_G_SLICED_VBI_CAP _IOWR('V', 69, struct v4l2_sliced_
↪vbi_cap)
#define VIDIOC_LOG_STATUS         _IO('V', 70)
#define VIDIOC_G_EXT_CTRLS       _IOWR('V', 71, struct v4l2_ext_
↪controls)
#define VIDIOC_S_EXT_CTRLS       _IOWR('V', 72, struct v4l2_ext_
↪controls)
#define VIDIOC_TRY_EXT_CTRLS     _IOWR('V', 73, struct v4l2_ext_
↪controls)
#define VIDIOC_ENUM_FRAMESIZES   _IOWR('V', 74, struct v4l2_
↪frmsizeenum)
#define VIDIOC_ENUM_FRAMEINTERVALS _IOWR('V', 75, struct v4l2_
↪frmivalenum)
#define VIDIOC_G_ENC_INDEX        _IOR('V', 76, struct v4l2_enc_idx)
#define VIDIOC_ENCODER_CMD       _IOWR('V', 77, struct v4l2_encoder_
↪cmd)
#define VIDIOC_TRY_ENCODER_CMD   _IOWR('V', 78, struct v4l2_encoder_
↪cmd)

/*
 * Experimental, meant for debugging, testing and internal use.
 * Only implemented if CONFIG_VIDEO_ADV_DEBUG is defined.
 * You must be root to use these ioctls. Never use these in␣
↪applications!
 */
#define VIDIOC_DBG_S_REGISTER     _IOW('V', 79, struct v4l2_dbg_
↪register)
#define VIDIOC_DBG_G_REGISTER    _IOWR('V', 80, struct v4l2_dbg_
↪register)

#define VIDIOC_S_HW_FREQ_SEEK     _IOW('V', 82, struct v4l2_hw_freq_
↪seek)
#define VIDIOC_S_DV_TIMINGS      _IOWR('V', 87, struct v4l2_dv_
↪timings)
#define VIDIOC_G_DV_TIMINGS      _IOWR('V', 88, struct v4l2_dv_
↪timings)
#define VIDIOC_DQEVENT            _IOR('V', 89, struct v4l2_event)
#define VIDIOC_SUBSCRIBE_EVENT    _IOW('V', 90, struct v4l2_event_
↪subscription)
#define VIDIOC_UNSUBSCRIBE_EVENT  _IOW('V', 91, struct v4l2_event_
↪subscription)
#define VIDIOC_CREATE_BUFS       _IOWR('V', 92, struct v4l2_create_
↪buffers)
#define VIDIOC_PREPARE_BUF       _IOWR('V', 93, struct v4l2_buffer)
#define VIDIOC_G_SELECTION       _IOWR('V', 94, struct v4l2_
↪selection)
#define VIDIOC_S_SELECTION       _IOWR('V', 95, struct v4l2_
↪selection)
#define VIDIOC_DECODER_CMD       _IOWR('V', 96, struct v4l2_decoder_
↪cmd)
#define VIDIOC_TRY_DECODER_CMD   _IOWR('V', 97, struct v4l2_decoder_
```

```
↪cmd)
#define VIDIOC_ENUM_DV_TIMINGS  _IOWR('V', 98, struct v4l2_enum_dv_
↪timings)
#define VIDIOC_QUERY_DV_TIMINGS  _IOR('V', 99, struct v4l2_dv_
↪timings)
#define VIDIOC_DV_TIMINGS_CAP   _IOWR('V', 100, struct v4l2_dv_
↪timings_cap)
#define VIDIOC_ENUM_FREQ_BANDS  _IOWR('V', 101, struct v4l2_
↪frequency_band)

/*
 * Experimental, meant for debugging, testing and internal use.
 * Never use this in applications!
 */
#define VIDIOC_DBG_G_CHIP_INFO  _IOWR('V', 102, struct v4l2_dbg_
↪chip_info)

#define VIDIOC_QUERY_EXT_CTRL   _IOWR('V', 103, struct v4l2_query_
↪ext_ctrl)

/* Reminder: when adding new ioctls please add support for them to
   drivers/media/v4l2-core/v4l2-compat-ioctl32.c as well! */

#define BASE_VIDIOC_PRIVATE     192             /* 192-255 are␣
↪private */

#endif /* _UAPI__LINUX_VIDEODEV2_H */
```

8.2.10 Video Capture Example

file: media/v4l/capture.c

```
/*
 *  V4L2 video capture example
 *
 *  This program can be used and distributed without restrictions.
 *
 *      This program is provided with the V4L2 API
 * see https://linuxtv.org/docs.php for more information
 */

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <assert.h>

#include <getopt.h>             /* getopt_long() */

#include <fcntl.h>              /* low-level i/o */
```

(continues on next page)

(continued from previous page)

```
#include <unistd.h>
#include <errno.h>
#include <sys/stat.h>
#include <sys/types.h>
#include <sys/time.h>
#include <sys/mman.h>
#include <sys/ioctl.h>

#include <linux/videodev2.h>

#define CLEAR(x) memset(&(x), 0, sizeof(x))

enum io_method {
        IO_METHOD_READ,
        IO_METHOD_MMAP,
        IO_METHOD_USERPTR,
};

struct buffer {
        void   *start;
        size_t  length;
};

static char            *dev_name;
static enum io_method   io = IO_METHOD_MMAP;
static int              fd = -1;
struct buffer          *buffers;
static unsigned int     n_buffers;
static int              out_buf;
static int              force_format;
static int              frame_count = 70;

static void errno_exit(const char *s)
{
        fprintf(stderr, "%s error %d, %s\\n", s, errno,␣
↪strerror(errno));
        exit(EXIT_FAILURE);
}

static int xioctl(int fh, int request, void *arg)
{
        int r;

        do {
                r = ioctl(fh, request, arg);
        } while (-1 == r && EINTR == errno);

        return r;
}
```

(continues on next page)

(continued from previous page)

```
static void process_image(const void *p, int size)
{
        if (out_buf)
                fwrite(p, size, 1, stdout);

        fflush(stderr);
        fprintf(stderr, ".");
        fflush(stdout);
}

static int read_frame(void)
{
        struct v4l2_buffer buf;
        unsigned int i;

        switch (io) {
        case IO_METHOD_READ:
                if (-1 == read(fd, buffers[0].start, buffers[0].
↪length)) {
                        switch (errno) {
                        case EAGAIN:
                                return 0;

                        case EIO:
                                /* Could ignore EIO, see spec. */

                                /* fall through */

                        default:
                                errno_exit("read");
                        }
                }

                process_image(buffers[0].start, buffers[0].length);
                break;

        case IO_METHOD_MMAP:
                CLEAR(buf);

                buf.type = V4L2_BUF_TYPE_VIDEO_CAPTURE;
                buf.memory = V4L2_MEMORY_MMAP;

                if (-1 == xioctl(fd, VIDIOC_DQBUF, &buf)) {
                        switch (errno) {
                        case EAGAIN:
                                return 0;

                        case EIO:
```

(continues on next page)

(continued from previous page)

```
                                /* Could ignore EIO, see spec. */

                                /* fall through */

                        default:
                                errno_exit("VIDIOC_DQBUF");
                        }
                }

                assert(buf.index < n_buffers);

                process_image(buffers[buf.index].start, buf.
↪bytesused);

                if (-1 == xioctl(fd, VIDIOC_QBUF, &buf))
                        errno_exit("VIDIOC_QBUF");
                break;

        case IO_METHOD_USERPTR:
                CLEAR(buf);

                buf.type = V4L2_BUF_TYPE_VIDEO_CAPTURE;
                buf.memory = V4L2_MEMORY_USERPTR;

                if (-1 == xioctl(fd, VIDIOC_DQBUF, &buf)) {
                        switch (errno) {
                        case EAGAIN:
                                return 0;

                        case EIO:
                                /* Could ignore EIO, see spec. */

                                /* fall through */

                        default:
                                errno_exit("VIDIOC_DQBUF");
                        }
                }

                for (i = 0; i < n_buffers; ++i)
                        if (buf.m.userptr == (unsigned␣
↪long)buffers[i].start
                            && buf.length == buffers[i].length)
                                break;

                assert(i < n_buffers);

                process_image((void *)buf.m.userptr, buf.bytesused);
```

(continues on next page)

(continued from previous page)

```
                if (-1 == xioctl(fd, VIDIOC_QBUF, &buf))
                        errno_exit("VIDIOC_QBUF");
                break;
        }

        return 1;
}

static void mainloop(void)
{
        unsigned int count;

        count = frame_count;

        while (count-- > 0) {
                for (;;) {
                        fd_set fds;
                        struct timeval tv;
                        int r;

                        FD_ZERO(&fds);
                        FD_SET(fd, &fds);

                        /* Timeout. */
                        tv.tv_sec = 2;
                        tv.tv_usec = 0;

                        r = select(fd + 1, &fds, NULL, NULL, &tv);

                        if (-1 == r) {
                                if (EINTR == errno)
                                        continue;
                                errno_exit("select");
                        }

                        if (0 == r) {
                                fprintf(stderr, "select timeout\\n
↪");

                                exit(EXIT_FAILURE);
                        }

                        if (read_frame())
                                break;
                        /* EAGAIN - continue select loop. */
                }
        }
}

static void stop_capturing(void)
```

(continues on next page)

(continued from previous page)

```
{
        enum v4l2_buf_type type;

        switch (io) {
        case IO_METHOD_READ:
                /* Nothing to do. */
                break;

        case IO_METHOD_MMAP:
        case IO_METHOD_USERPTR:
                type = V4L2_BUF_TYPE_VIDEO_CAPTURE;
                if (-1 == xioctl(fd, VIDIOC_STREAMOFF, &type))
                        errno_exit("VIDIOC_STREAMOFF");
                break;
        }
}

static void start_capturing(void)
{
        unsigned int i;
        enum v4l2_buf_type type;

        switch (io) {
        case IO_METHOD_READ:
                /* Nothing to do. */
                break;

        case IO_METHOD_MMAP:
                for (i = 0; i < n_buffers; ++i) {
                        struct v4l2_buffer buf;

                        CLEAR(buf);
                        buf.type = V4L2_BUF_TYPE_VIDEO_CAPTURE;
                        buf.memory = V4L2_MEMORY_MMAP;
                        buf.index = i;

                        if (-1 == xioctl(fd, VIDIOC_QBUF, &buf))
                                errno_exit("VIDIOC_QBUF");
                }
                type = V4L2_BUF_TYPE_VIDEO_CAPTURE;
                if (-1 == xioctl(fd, VIDIOC_STREAMON, &type))
                        errno_exit("VIDIOC_STREAMON");
                break;

        case IO_METHOD_USERPTR:
                for (i = 0; i < n_buffers; ++i) {
                        struct v4l2_buffer buf;

                        CLEAR(buf);
```

(continues on next page)

(continued from previous page)

```
                                buf.type = V4L2_BUF_TYPE_VIDEO_CAPTURE;
                                buf.memory = V4L2_MEMORY_USERPTR;
                                buf.index = i;
                                buf.m.userptr = (unsigned long)buffers[i].
↪start;
                                buf.length = buffers[i].length;

                                if (-1 == xioctl(fd, VIDIOC_QBUF, &buf))
                                        errno_exit("VIDIOC_QBUF");
                        }
                        type = V4L2_BUF_TYPE_VIDEO_CAPTURE;
                        if (-1 == xioctl(fd, VIDIOC_STREAMON, &type))
                                errno_exit("VIDIOC_STREAMON");
                        break;
        }
}

static void uninit_device(void)
{
        unsigned int i;

        switch (io) {
        case IO_METHOD_READ:
                free(buffers[0].start);
                break;

        case IO_METHOD_MMAP:
                for (i = 0; i < n_buffers; ++i)
                        if (-1 == munmap(buffers[i].start,␣
↪buffers[i].length))
                                errno_exit("munmap");
                break;

        case IO_METHOD_USERPTR:
                for (i = 0; i < n_buffers; ++i)
                        free(buffers[i].start);
                break;
        }

        free(buffers);
}

static void init_read(unsigned int buffer_size)
{
        buffers = calloc(1, sizeof(*buffers));

        if (!buffers) {
                fprintf(stderr, "Out of memory\\n");
                exit(EXIT_FAILURE);
```

(continues on next page)

(continued from previous page)

```
        }

        buffers[0].length = buffer_size;
        buffers[0].start = malloc(buffer_size);

        if (!buffers[0].start) {
                fprintf(stderr, "Out of memory\\n");
                exit(EXIT_FAILURE);
        }
}

static void init_mmap(void)
{
        struct v4l2_requestbuffers req;

        CLEAR(req);

        req.count = 4;
        req.type = V4L2_BUF_TYPE_VIDEO_CAPTURE;
        req.memory = V4L2_MEMORY_MMAP;

        if (-1 == xioctl(fd, VIDIOC_REQBUFS, &req)) {
                if (EINVAL == errno) {
                        fprintf(stderr, "%s does not support "
                                 "memory mappingn", dev_name);
                        exit(EXIT_FAILURE);
                } else {
                        errno_exit("VIDIOC_REQBUFS");
                }
        }

        if (req.count < 2) {
                fprintf(stderr, "Insufficient buffer memory on %s\\n
↪",
                         dev_name);
                exit(EXIT_FAILURE);
        }

        buffers = calloc(req.count, sizeof(*buffers));

        if (!buffers) {
                fprintf(stderr, "Out of memory\\n");
                exit(EXIT_FAILURE);
        }

        for (n_buffers = 0; n_buffers < req.count; ++n_buffers) {
                struct v4l2_buffer buf;

                CLEAR(buf);
```

(continues on next page)

(continued from previous page)

```
                buf.type        = V4L2_BUF_TYPE_VIDEO_CAPTURE;
                buf.memory      = V4L2_MEMORY_MMAP;
                buf.index       = n_buffers;

                if (-1 == xioctl(fd, VIDIOC_QUERYBUF, &buf))
                        errno_exit("VIDIOC_QUERYBUF");

                buffers[n_buffers].length = buf.length;
                buffers[n_buffers].start =
                        mmap(NULL /* start anywhere */,
                              buf.length,
                              PROT_READ | PROT_WRITE /* required */,
                              MAP_SHARED /* recommended */,
                              fd, buf.m.offset);

                if (MAP_FAILED == buffers[n_buffers].start)
                        errno_exit("mmap");
        }
}

static void init_userp(unsigned int buffer_size)
{
        struct v4l2_requestbuffers req;

        CLEAR(req);

        req.count  = 4;
        req.type   = V4L2_BUF_TYPE_VIDEO_CAPTURE;
        req.memory = V4L2_MEMORY_USERPTR;

        if (-1 == xioctl(fd, VIDIOC_REQBUFS, &req)) {
                if (EINVAL == errno) {
                        fprintf(stderr, "%s does not support "
                                 "user pointer i/on", dev_name);
                        exit(EXIT_FAILURE);
                } else {
                        errno_exit("VIDIOC_REQBUFS");
                }
        }

        buffers = calloc(4, sizeof(*buffers));

        if (!buffers) {
                fprintf(stderr, "Out of memory\\n");
                exit(EXIT_FAILURE);
        }

        for (n_buffers = 0; n_buffers < 4; ++n_buffers) {
```

(continues on next page)

(continued from previous page)

```
                buffers[n_buffers].length = buffer_size;
                buffers[n_buffers].start = malloc(buffer_size);

                if (!buffers[n_buffers].start) {
                        fprintf(stderr, "Out of memory\\n");
                        exit(EXIT_FAILURE);
                }
        }
}

static void init_device(void)
{
        struct v4l2_capability cap;
        struct v4l2_cropcap cropcap;
        struct v4l2_crop crop;
        struct v4l2_format fmt;
        unsigned int min;

        if (-1 == xioctl(fd, VIDIOC_QUERYCAP, &cap)) {
                if (EINVAL == errno) {
                        fprintf(stderr, "%s is no V4L2 device\\n",
                                 dev_name);
                        exit(EXIT_FAILURE);
                } else {
                        errno_exit("VIDIOC_QUERYCAP");
                }
        }

        if (!(cap.capabilities & V4L2_CAP_VIDEO_CAPTURE)) {
                fprintf(stderr, "%s is no video capture device\\n",
                         dev_name);
                exit(EXIT_FAILURE);
        }

        switch (io) {
        case IO_METHOD_READ:
                if (!(cap.capabilities & V4L2_CAP_READWRITE)) {
                        fprintf(stderr, "%s does not support read i/
→o\\n",
                                 dev_name);
                        exit(EXIT_FAILURE);
                }
                break;

        case IO_METHOD_MMAP:
        case IO_METHOD_USERPTR:
                if (!(cap.capabilities & V4L2_CAP_STREAMING)) {
                        fprintf(stderr, "%s does not support␣
→streaming i/o\\n",
```

(continues on next page)

(continued from previous page)

```
                                dev_name);
                        exit(EXIT_FAILURE);
                }
                break;
        }


        /* Select video input, video standard and tune here. */


        CLEAR(cropcap);

        cropcap.type = V4L2_BUF_TYPE_VIDEO_CAPTURE;

        if (0 == xioctl(fd, VIDIOC_CROPCAP, &cropcap)) {
                crop.type = V4L2_BUF_TYPE_VIDEO_CAPTURE;
                crop.c = cropcap.defrect; /* reset to default */

                if (-1 == xioctl(fd, VIDIOC_S_CROP, &crop)) {
                        switch (errno) {
                        case EINVAL:
                                /* Cropping not supported. */
                                break;
                        default:
                                /* Errors ignored. */
                                break;
                        }
                }
        } else {
                /* Errors ignored. */
        }


        CLEAR(fmt);

        fmt.type = V4L2_BUF_TYPE_VIDEO_CAPTURE;
        if (force_format) {
                fmt.fmt.pix.width       = 640;
                fmt.fmt.pix.height      = 480;
                fmt.fmt.pix.pixelformat = V4L2_PIX_FMT_YUYV;
                fmt.fmt.pix.field       = V4L2_FIELD_INTERLACED;

                if (-1 == xioctl(fd, VIDIOC_S_FMT, &fmt))
                        errno_exit("VIDIOC_S_FMT");

                /* Note VIDIOC_S_FMT may change width and height. */
        } else {
                /* Preserve original settings as set by v4l2-ctl␣
↪for example */
```

(continues on next page)

(continued from previous page)

```
                if (-1 == xioctl(fd, VIDIOC_G_FMT, &fmt))
                        errno_exit("VIDIOC_G_FMT");
        }

        /* Buggy driver paranoia. */
        min = fmt.fmt.pix.width * 2;
        if (fmt.fmt.pix.bytesperline < min)
                fmt.fmt.pix.bytesperline = min;
        min = fmt.fmt.pix.bytesperline * fmt.fmt.pix.height;
        if (fmt.fmt.pix.sizeimage < min)
                fmt.fmt.pix.sizeimage = min;

        switch (io) {
        case IO_METHOD_READ:
                init_read(fmt.fmt.pix.sizeimage);
                break;

        case IO_METHOD_MMAP:
                init_mmap();
                break;

        case IO_METHOD_USERPTR:
                init_userp(fmt.fmt.pix.sizeimage);
                break;
        }
}

static void close_device(void)
{
        if (-1 == close(fd))
                errno_exit("close");

        fd = -1;
}

static void open_device(void)
{
        struct stat st;

        if (-1 == stat(dev_name, &st)) {
                fprintf(stderr, "Cannot identify '%s': %d, %s\\n",
                         dev_name, errno, strerror(errno));
                exit(EXIT_FAILURE);
        }

        if (!S_ISCHR(st.st_mode)) {
                fprintf(stderr, "%s is no devicen", dev_name);
                exit(EXIT_FAILURE);
        }
```

(continues on next page)

(continued from previous page)

```
        fd = open(dev_name, O_RDWR /* required */ | O_NONBLOCK, 0);

        if (-1 == fd) {
                fprintf(stderr, "Cannot open '%s': %d, %s\\n",
                         dev_name, errno, strerror(errno));
                exit(EXIT_FAILURE);
        }
}

static void usage(FILE *fp, int argc, char **argv)
{
        fprintf(fp,
                 "Usage: %s [options]\\n\\n"
                 "Version 1.3\\n"
                 "Options:\\n"
                 "-d | --device name   Video device name [%s]n"
                 "-h | --help          Print this messagen"
                 "-m | --mmap          Use memory mapped buffers␣
 ↪[default]n"
                 "-r | --read          Use read() callsn"
                 "-u | --userp         Use application allocated␣
 ↪buffersn"
                 "-o | --output        Outputs stream to stdoutn"
                 "-f | --format        Force format to 640x480 YUYVn
 ↪"
                 "-c | --count         Number of frames to grab [
 ↪%i]n"
                 "",
                 argv[0], dev_name, frame_count);
}

static const char short_options[] = "d:hmruofc:";

static const struct option
long_options[] = {
        { "device", required_argument, NULL, 'd' },
        { "help",   no_argument,       NULL, 'h' },
        { "mmap",   no_argument,       NULL, 'm' },
        { "read",   no_argument,       NULL, 'r' },
        { "userp",  no_argument,       NULL, 'u' },
        { "output", no_argument,       NULL, 'o' },
        { "format", no_argument,       NULL, 'f' },
        { "count",  required_argument, NULL, 'c' },
        { 0, 0, 0, 0 }
};

int main(int argc, char **argv)
{
```

(continues on next page)

(continued from previous page)

```
        dev_name = "/dev/video0";

        for (;;) {
                int idx;
                int c;

                c = getopt_long(argc, argv,
                                short_options, long_options, &idx);

                if (-1 == c)
                        break;

                switch (c) {
                case 0: /* getopt_long() flag */
                        break;

                case 'd':
                        dev_name = optarg;
                        break;

                case 'h':
                        usage(stdout, argc, argv);
                        exit(EXIT_SUCCESS);

                case 'm':
                        io = IO_METHOD_MMAP;
                        break;

                case 'r':
                        io = IO_METHOD_READ;
                        break;

                case 'u':
                        io = IO_METHOD_USERPTR;
                        break;

                case 'o':
                        out_buf++;
                        break;

                case 'f':
                        force_format++;
                        break;

                case 'c':
                        errno = 0;
                        frame_count = strtol(optarg, NULL, 0);
                        if (errno)
                                errno_exit(optarg);
```

(continues on next page)

(continued from previous page)

```
                        break;

                default:
                        usage(stderr, argc, argv);
                        exit(EXIT_FAILURE);
                }
        }

        open_device();
        init_device();
        start_capturing();
        mainloop();
        stop_capturing();
        uninit_device();
        close_device();
        fprintf(stderr, "\\n");
        return 0;
}
```

8.2.11 Video Grabber example using libv4l

This program demonstrates how to grab V4L2 images in ppm format by using libv4l handlers. The advantage is that this grabber can potentially work with any V4L2 driver.

file: media/v4l/v4l2grab.c

```
/* V4L2 video picture grabber
   Copyright (C) 2009 Mauro Carvalho Chehab <mchehab@kernel.org>

   This program is free software; you can redistribute it and/or␣
↪modify
   it under the terms of the GNU General Public License as␣
↪published by
   the Free Software Foundation version 2 of the License.

   This program is distributed in the hope that it will be useful,
   but WITHOUT ANY WARRANTY; without even the implied warranty of
   MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
   GNU General Public License for more details.
 */

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <fcntl.h>
#include <errno.h>
```

(continues on next page)

(continued from previous page)

```
#include <sys/ioctl.h>
#include <sys/types.h>
#include <sys/time.h>
#include <sys/mman.h>
#include <linux/videodev2.h>
#include "../libv4l/include/libv4l2.h"

#define CLEAR(x) memset(&(x), 0, sizeof(x))

struct buffer {
        void   *start;
        size_t length;
};

static void xioctl(int fh, int request, void *arg)
{
        int r;

        do {
                r = v4l2_ioctl(fh, request, arg);
        } while (r == -1 && ((errno == EINTR) || (errno ==␣
↪EAGAIN)));

        if (r == -1) {
                fprintf(stderr, "error %d, %s\\n", errno,␣
↪strerror(errno));
                exit(EXIT_FAILURE);
        }
}

int main(int argc, char **argv)
{
        struct v4l2_format              fmt;
        struct v4l2_buffer              buf;
        struct v4l2_requestbuffers      req;
        enum v4l2_buf_type              type;
        fd_set                          fds;
        struct timeval                  tv;
        int                             r, fd = -1;
        unsigned int                    i, n_buffers;
        char                            *dev_name = "/dev/video0";
        char                            out_name[256];
        FILE                            *fout;
        struct buffer                   *buffers;

        fd = v4l2_open(dev_name, O_RDWR | O_NONBLOCK, 0);
        if (fd < 0) {
                perror("Cannot open device");
                exit(EXIT_FAILURE);
```

(continues on next page)

(continued from previous page)

```
        }

        CLEAR(fmt);
        fmt.type = V4L2_BUF_TYPE_VIDEO_CAPTURE;
        fmt.fmt.pix.width       = 640;
        fmt.fmt.pix.height      = 480;
        fmt.fmt.pix.pixelformat = V4L2_PIX_FMT_RGB24;
        fmt.fmt.pix.field       = V4L2_FIELD_INTERLACED;
        xioctl(fd, VIDIOC_S_FMT, &fmt);
        if (fmt.fmt.pix.pixelformat != V4L2_PIX_FMT_RGB24) {
                printf("Libv4l didn't accept RGB24 format. Can't␣
↪proceed.\\n");
                exit(EXIT_FAILURE);
        }
        if ((fmt.fmt.pix.width != 640) || (fmt.fmt.pix.height !=␣
↪480))
                printf("Warning: driver is sending image at %dx%d\\n
↪",
                        fmt.fmt.pix.width, fmt.fmt.pix.height);

        CLEAR(req);
        req.count = 2;
        req.type = V4L2_BUF_TYPE_VIDEO_CAPTURE;
        req.memory = V4L2_MEMORY_MMAP;
        xioctl(fd, VIDIOC_REQBUFS, &req);

        buffers = calloc(req.count, sizeof(*buffers));
        for (n_buffers = 0; n_buffers < req.count; ++n_buffers) {
                CLEAR(buf);

                buf.type        = V4L2_BUF_TYPE_VIDEO_CAPTURE;
                buf.memory      = V4L2_MEMORY_MMAP;
                buf.index       = n_buffers;

                xioctl(fd, VIDIOC_QUERYBUF, &buf);

                buffers[n_buffers].length = buf.length;
                buffers[n_buffers].start = v4l2_mmap(NULL, buf.
↪length,
                              PROT_READ | PROT_WRITE, MAP_SHARED,
                              fd, buf.m.offset);

                if (MAP_FAILED == buffers[n_buffers].start) {
                        perror("mmap");
                        exit(EXIT_FAILURE);
                }
        }

        for (i = 0; i < n_buffers; ++i) {
```

(continues on next page)

(continued from previous page)

```
                CLEAR(buf);
                buf.type = V4L2_BUF_TYPE_VIDEO_CAPTURE;
                buf.memory = V4L2_MEMORY_MMAP;
                buf.index = i;
                xioctl(fd, VIDIOC_QBUF, &buf);
        }
        type = V4L2_BUF_TYPE_VIDEO_CAPTURE;

        xioctl(fd, VIDIOC_STREAMON, &type);
        for (i = 0; i < 20; i++) {
                do {
                        FD_ZERO(&fds);
                        FD_SET(fd, &fds);

                        /* Timeout. */
                        tv.tv_sec = 2;
                        tv.tv_usec = 0;

                        r = select(fd + 1, &fds, NULL, NULL, &tv);
                } while ((r == -1 && (errno = EINTR)));
                if (r == -1) {
                        perror("select");
                        return errno;
                }

                CLEAR(buf);
                buf.type = V4L2_BUF_TYPE_VIDEO_CAPTURE;
                buf.memory = V4L2_MEMORY_MMAP;
                xioctl(fd, VIDIOC_DQBUF, &buf);

                sprintf(out_name, "out%03d.ppm", i);
                fout = fopen(out_name, "w");
                if (!fout) {
                        perror("Cannot open image");
                        exit(EXIT_FAILURE);
                }
                fprintf(fout, "P6\\n%d %d 255\\n",
                        fmt.fmt.pix.width, fmt.fmt.pix.height);
                fwrite(buffers[buf.index].start, buf.bytesused, 1,␣
↪fout);
                fclose(fout);

                xioctl(fd, VIDIOC_QBUF, &buf);
        }

        type = V4L2_BUF_TYPE_VIDEO_CAPTURE;
        xioctl(fd, VIDIOC_STREAMOFF, &type);
        for (i = 0; i < n_buffers; ++i)
                v4l2_munmap(buffers[i].start, buffers[i].length);
```

(continues on next page)

(continued from previous page)

```
        v4l2_close(fd);

        return 0;
}
```

8.2.12 References

CEA 608-E

title
CEA-608-E R-2014 "Line 21 Data Services"

author
Consumer Electronics Association (http://www.ce.org)

EN 300 294

title
EN 300 294 "625-line television Wide Screen Signalling (WSS)"

author
European Telecommunication Standards Institute (http://www.etsi.org)

ETS 300 231

title
ETS 300 231 "Specification of the domestic video Programme Delivery Control system (PDC)"

author
European Telecommunication Standards Institute (http://www.etsi.org)

ETS 300 706

title
ETS 300 706 "Enhanced Teletext specification"

author
European Telecommunication Standards Institute (http://www.etsi.org)

ISO 13818-1

title
ITU-T Rec. H.222.0 | ISO/IEC 13818-1 "Information technology —Generic coding of moving pictures and associated audio information: Systems"

author
International Telecommunication Union (http://www.itu.ch), International Organisation for Standardisation (http://www.iso.ch)

ISO 13818-2

title
ITU-T Rec. H.262 | ISO/IEC 13818-2 "Information technology — Generic coding of moving pictures and associated audio information: Video"

author
International Telecommunication Union (http://www.itu.ch), International Organisation for Standardisation (http://www.iso.ch)

ITU BT.470

title
ITU-R Recommendation BT.470-6 "Conventional Television Systems"

author
International Telecommunication Union (http://www.itu.ch)

ITU BT.601

title
ITU-R Recommendation BT.601-5 "Studio Encoding Parameters of Digital Television for Standard 4:3 and Wide-Screen 16:9 Aspect Ratios"

author
International Telecommunication Union (http://www.itu.ch)

ITU BT.653

title
ITU-R Recommendation BT.653-3 "Teletext systems"

author
International Telecommunication Union (http://www.itu.ch)

ITU BT.709

title
ITU-R Recommendation BT.709-5 "Parameter values for the HDTV standards for production and international programme exchange"

author
International Telecommunication Union (http://www.itu.ch)

ITU BT.1119

title
ITU-R Recommendation BT.1119 "625-line television Wide Screen Signalling (WSS)"

author
International Telecommunication Union (http://www.itu.ch)

ITU-T Rec. H.264 Specification (04/2017 Edition)

title
ITU-T Recommendation H.264 "Advanced Video Coding for Generic Audiovisual Services"

author
International Telecommunication Union (http://www.itu.ch)

ITU H.265/HEVC

title
ITU-T Rec. H.265 | ISO/IEC 23008-2 "High Efficiency Video Coding"

author
International Telecommunication Union (http://www.itu.ch), International Organisation for Standardisation (http://www.iso.ch)

JFIF

title
JPEG File Interchange Format

subtitle
Version 1.02

author
Independent JPEG Group (http://www.ijg.org)

ITU-T.81

title
ITU-T Recommendation T.81 "Information Technology —Digital Compression and Coding of Continous-Tone Still Images —Requirements and Guidelines"

author
International Telecommunication Union (http://www.itu.int)

W3C JPEG JFIF

title
JPEG JFIF

author
The World Wide Web Consortium (http://www.w3.org)

SMPTE 12M

title
SMPTE 12M-1999 "Television, Audio and Film - Time and Control Code"

author
Society of Motion Picture and Television Engineers (http://www.smpte.org)

SMPTE 170M

title
SMPTE 170M-1999 "Television - Composite Analog Video Signal - NTSC for Studio Applications"

author
Society of Motion Picture and Television Engineers (http://www.smpte.org)

SMPTE 240M

title
SMPTE 240M-1999 "Television - Signal Parameters - 1125-Line High-Definition Production"

author
Society of Motion Picture and Television Engineers (http://www.smpte.org)

SMPTE RP 431-2

title
SMPTE RP 431-2:2011 "D-Cinema Quality - Reference Projector and Environment"

author
Society of Motion Picture and Television Engineers (http://www.smpte.org)

SMPTE ST 2084

title
SMPTE ST 2084:2014 "High Dynamic Range Electro-Optical Transfer Function of Master Reference Displays"

author
Society of Motion Picture and Television Engineers (http://www.smpte.org)

sRGB

title
IEC 61966-2-1 ed1.0 "Multimedia systems and equipment - Colour measurement and management - Part 2-1: Colour management - Default RGB colour space - sRGB"

author
International Electrotechnical Commission (http://www.iec.ch)

sYCC

title
IEC 61966-2-1-am1 ed1.0 "Amendment 1 - Multimedia systems and equipment - Colour measurement and management - Part 2-1: Colour management - Default RGB colour space - sRGB"

author
International Electrotechnical Commission (http://www.iec.ch)

xvYCC

title
IEC 61966-2-4 ed1.0 "Multimedia systems and equipment - Colour measurement and management - Part 2-4: Colour management - Extended-gamut YCC colour space for video applications - xvYCC"

author
International Electrotechnical Commission (http://www.iec.ch)

opRGB

title
IEC 61966-2-5 "Multimedia systems and equipment - Colour measurement and management - Part 2-5: Colour management - Optional RGB colour space - opRGB"

author
International Electrotechnical Commission (http://www.iec.ch)

ITU BT.2020

title
ITU-R Recommendation BT.2020 (08/2012) "Parameter values for ultra-high definition television systems for production and international programme exchange"

author
International Telecommunication Union (http://www.itu.ch)

EBU Tech 3213

title
E.B.U. Standard for Chromaticity Tolerances for Studio Monitors"

author
European Broadcast Union (http://www.ebu.ch)

IEC 62106

title
Specification of the radio data system (RDS) for VHF/FM sound broadcasting in the frequency range from 87,5 to 108,0 MHz

author
International Electrotechnical Commission (http://www.iec.ch)

NRSC-4-B

title
NRSC-4-B: United States RBDS Standard

author
National Radio Systems Committee (http://www.nrscstandards.org)

ISO 12232:2006

title
Photography —Digital still cameras —Determination of exposure index, ISO speed ratings, standard output sensitivity, and recommended exposure index

author
International Organization for Standardization (http://www.iso.org)

CEA-861-E

title
A DTV Profile for Uncompressed High Speed Digital Interfaces

author
Consumer Electronics Association (http://www.ce.org)

VESA DMT

title
VESA and Industry Standards and Guidelines for Computer Display Monitor Timing (DMT)

author
Video Electronics Standards Association (http://www.vesa.org)

EDID

title
VESA Enhanced Extended Display Identification Data Standard

subtitle
Release A, Revision 2

author
Video Electronics Standards Association (http://www.vesa.org)

HDCP

title
High-bandwidth Digital Content Protection System

subtitle
Revision 1.3

author
Digital Content Protection LLC (http://www.digital-cp.com)

HDMI

title
High-Definition Multimedia Interface

subtitle
Specification Version 1.4a

author
HDMI Licensing LLC (http://www.hdmi.org)

HDMI2

title
High-Definition Multimedia Interface

subtitle
Specification Version 2.0

author
HDMI Licensing LLC (http://www.hdmi.org)

DP

title
VESA DisplayPort Standard

subtitle
Version 1, Revision 2

author
Video Electronics Standards Association (http://www.vesa.org)

poynton

title
Digital Video and HDTV, Algorithms and Interfaces

author
Charles Poynton

colimg

title
Color Imaging: Fundamentals and Applications

author
Erik Reinhard et al.

VP8

title
RFC 6386: "VP8 Data Format and Decoding Guide"

author
J. Bankoski et al.

8.2.13 Revision and Copyright

Authors, in alphabetical order:

- Ailus, Sakari <sakari.ailus@iki.fi>
  - Subdev selections API.
- Carvalho Chehab, Mauro <mchehab+samsung@kernel.org>
  - Documented libv4l, designed and added v4l2grab example, Remote Controller chapter.
- Dirks, Bill
  - Original author of the V4L2 API and documentation.
- Figa, Tomasz <tfiga@chromium.org>
  - Documented the memory-to-memory decoder interface.
  - Documented the memory-to-memory encoder interface.
- H Schimek, Michael <mschimek@gmx.at>
  - Original author of the V4L2 API and documentation.
- Karicheri, Muralidharan <m-karicheri2@ti.com>
  - Documented the Digital Video timings API.
- Osciak, Pawel <posciak@chromium.org>
  - Documented the memory-to-memory decoder interface.
  - Documented the memory-to-memory encoder interface.
- Osciak, Pawel <pawel@osciak.com>
  - Designed and documented the multi-planar API.
- Palosaari, Antti <crope@iki.fi>
  - SDR API.
- Ribalda, Ricardo
  - Introduce HSV formats and other minor changes.
- Rubli, Martin
  - Designed and documented the VIDIOC_ENUM_FRAMESIZES and VIDIOC_ENUM_FRAMEINTERVALS ioctls.
- Walls, Andy <awalls@md.metrocast.net>

  - Documented the fielded V4L2_MPEG_STREAM_VBI_FMT_IVTV MPEG stream embedded, sliced VBI data format in this specification.

- Verkuil, Hans <hverkuil@xs4all.nl>
  - Designed and documented the VIDIOC_LOG_STATUS ioctl, the extended control ioctls, major parts of the sliced VBI API, the MPEG encoder and decoder APIs and the DV Timings API.

8.2.14 Revision History

revision
4.10 / 2016-07-15 (*rr*)

Introduce HSV formats.

revision
4.5 / 2015-10-29 (*rr*)

Extend VIDIOC_G_EXT_CTRLS;. Replace ctrl_class with a new union with ctrl_class and which. Which is used to select the current value of the control or the default value.

revision
4.4 / 2015-05-26 (*ap*)

Renamed V4L2_TUNER_ADC to V4L2_TUNER_SDR. Added V4L2_CID_RF_TUNER_RF_GAIN control. Added transmitter support for Software Defined Radio (SDR) Interface.

revision
4.1 / 2015-02-13 (*mcc*)

Fix documentation for media controller device nodes and add support for DVB device nodes. Add support for Tuner sub-device.

revision
3.19 / 2014-12-05 (*hv*)

Rewrote Colorspace chapter, added new enum *v4l2_ycbcr_encoding* and enum *v4l2_quantization* fields to struct *v4l2_pix_format*, struct *v4l2_pix_format_mplane* and struct *v4l2_mbus_framefmt*.

revision
3.17 / 2014-08-04 (*lp, hv*)

Extended struct *v4l2_pix_format*. Added format flags. Added compound control types and VIDIOC_QUERY_EXT_CTRL.

revision
3.15 / 2014-02-03 (*hv, ap*)

Update several sections of "Common API Elements": "Opening and Closing Devices" "Querying Capabilities", "Application Priority", "Video Inputs and Outputs", "Audio Inputs and Outputs" "Tuners and Modulators", "Video Standards" and "Digital Video (DV) Timings". Added SDR API.

revision
3.14 / 2013-11-25 (*rr*)

Set width and height as unsigned on v4l2_rect.

revision
3.11 / 2013-05-26 (*hv*)

Remove obsolete VIDIOC_DBG_G_CHIP_IDENT ioctl.

revision
3.10 / 2013-03-25 (*hv*)

Remove obsolete and unused DV_PRESET ioctls: VIDIOC_G_DV_PRESET, VIDIOC_S_DV_PRESET, VIDIOC_QUERY_DV_PRESET and VIDIOC_ENUM_DV_PRESET. Remove the related v4l2_input/output capability flags V4L2_IN_CAP_PRESETS and V4L2_OUT_CAP_PRESETS. Added VIDIOC_DBG_G_CHIP_INFO.

revision
3.9 / 2012-12-03 (*sa, sn*)

Added timestamp types to v4l2_buffer. Added V4L2_EVENT_CTRL_CH_RANGE control event changes flag.

revision
3.6 / 2012-07-02 (*hv*)

Added VIDIOC_ENUM_FREQ_BANDS.

revision
3.5 / 2012-05-07 (*sa, sn, hv*)

Added V4L2_CTRL_TYPE_INTEGER_MENU and V4L2 subdev selections API. Improved the description of V4L2_CID_COLORFX control, added V4L2_CID_COLORFX_CBCR control. Added camera controls V4L2_CID_AUTO_EXPOSURE_BIAS, V4L2_CID_AUTO_N_PRESET_WHITE_BALANCE, V4L2_CID_IMAGE_STABILIZATION, V4L2_CID_ISO_SENSITIVITY, V4L2_CID_ISO_SENSITIVITY_AUTO, V4L2_CID_EXPOSURE_METERING, V4L2_CID_SCENE_MODE, V4L2_CID_3A_LOCK, V4L2_CID_AUTO_FOCUS_START, V4L2_CID_AUTO_FOCUS_STOP, V4L2_CID_AUTO_FOCUS_STATUS and V4L2_CID_AUTO_FOCUS_RANGE. Added VIDIOC_ENUM_DV_TIMINGS, VIDIOC_QUERY_DV_TIMINGS and VIDIOC_DV_TIMINGS_CAP.

revision
3.4 / 2012-01-25 (*sn*)

Added *JPEG compression control class.*

revision
3.3 / 2012-01-11 (*hv*)

Added device_caps field to struct v4l2_capabilities.

revision
3.2 / 2011-08-26 (*hv*)

Added V4L2_CTRL_FLAG_VOLATILE.

revision
3.1 / 2011-06-27 (*mcc, po, hv*)

Documented that VIDIOC_QUERYCAP now returns a per-subsystem version instead of a per-driver one. Standardize an error code for invalid ioctl. Added V4L2_CTRL_TYPE_BITMASK.

revision
2.6.39 / 2011-03-01 (*mcc, po*)

Removed VIDIOC_*_OLD from videodev2.h header and update it to reflect latest changes. Added the *multi-planar API*.

revision
2.6.37 / 2010-08-06 (*hv*)

Removed obsolete vtx (videotext) API.

revision
2.6.33 / 2009-12-03 (*mk*)

Added documentation for the Digital Video timings API.

revision
2.6.32 / 2009-08-31 (*mcc*)

Now, revisions will match the kernel version where the V4L2 API changes will be used by the Linux Kernel. Also added Remote Controller chapter.

revision
0.29 / 2009-08-26 (*ev*)

Added documentation for string controls and for FM Transmitter controls.

revision
0.28 / 2009-08-26 (*gl*)

Added V4L2_CID_BAND_STOP_FILTER documentation.

revision
0.27 / 2009-08-15 (*mcc*)

Added libv4l and Remote Controller documentation; added v4l2grab and keytable application examples.

revision
0.26 / 2009-07-23 (*hv*)

Finalized the RDS capture API. Added modulator and RDS encoder capabilities. Added support for string controls.

revision
0.25 / 2009-01-18 (*hv*)

Added pixel formats VYUY, NV16 and NV61, and changed the debug ioctls VIDIOC_DBG_G/S_REGISTER and VIDIOC_DBG_G_CHIP_IDENT. Added camera controls V4L2_CID_ZOOM_ABSOLUTE, V4L2_CID_ZOOM_RELATIVE, V4L2_CID_ZOOM_CONTINUOUS and V4L2_CID_PRIVACY.

revision
0.24 / 2008-03-04 (*mhs*)

Added pixel formats Y16 and SBGGR16, new controls and a camera controls class. Removed VIDIOC_G/S_MPEGCOMP.

revision
0.23 / 2007-08-30 (*mhs*)

Fixed a typo in VIDIOC_DBG_G/S_REGISTER. Clarified the byte order of packed pixel formats.

revision
0.22 / 2007-08-29 (*mhs*)

Added the Video Output Overlay interface, new MPEG controls, V4L2_FIELD_INTERLACED_TB and V4L2_FIELD_INTERLACED_BT, VIDIOC_DBG_G/S_REGISTER, VIDIOC_(TRY_)ENCODER_CMD, VIDIOC_G_CHIP_IDENT, VIDIOC_G_ENC_INDEX, new pixel formats. Clarifications in the cropping chapter, about RGB pixel formats, the mmap(), poll(), select(), read() and write() functions. Typographical fixes.

revision
0.21 / 2006-12-19 (*mhs*)

Fixed a link in the VIDIOC_G_EXT_CTRLS section.

revision
0.20 / 2006-11-24 (*mhs*)

Clarified the purpose of the audioset field in struct v4l2_input and v4l2_output.

revision
0.19 / 2006-10-19 (*mhs*)

Documented V4L2_PIX_FMT_RGB444.

revision
0.18 / 2006-10-18 (*mhs*)

Added the description of extended controls by Hans Verkuil. Linked V4L2_PIX_FMT_MPEG to V4L2_CID_MPEG_STREAM_TYPE.

revision
0.17 / 2006-10-12 (*mhs*)

Corrected V4L2_PIX_FMT_HM12 description.

revision
0.16 / 2006-10-08 (*mhs*)

VIDIOC_ENUM_FRAMESIZES and VIDIOC_ENUM_FRAMEINTERVALS are now part of the API.

revision
0.15 / 2006-09-23 (*mhs*)

Cleaned up the bibliography, added BT.653 and BT.1119. capture.c/start_capturing() for user pointer I/O did not initialize the buffer index. Documented the V4L MPEG and MJPEG VID_TYPEs and V4L2_PIX_FMT_SBGGR8. Updated the list of reserved pixel formats. See the history chapter for API changes.

revision
0.14 / 2006-09-14 (*mr*)

Added VIDIOC_ENUM_FRAMESIZES and VIDIOC_ENUM_FRAMEINTERVALS proposal for frame format enumeration of digital devices.

revision
0.13 / 2006-04-07 (*mhs*)

Corrected the description of *struct v4l2_window* clips. New V4L2_STD_ and V4L2_TUNER_MODE_LANG1_LANG2 defines.

revision
0.12 / 2006-02-03 (*mhs*)

Corrected the description of struct v4l2_captureparm and v4l2_outputparm.

revision
0.11 / 2006-01-27 (*mhs*)

Improved the description of struct v4l2_tuner.

revision
0.10 / 2006-01-10 (*mhs*)

VIDIOC_G_INPUT and VIDIOC_S_PARM clarifications.

revision
0.9 / 2005-11-27 (*mhs*)

Improved the 525 line numbering diagram. Hans Verkuil and I rewrote the sliced VBI section. He also contributed a VIDIOC_LOG_STATUS page. Fixed VIDIOC_S_STD call in the video standard selection example. Various updates.

revision
0.8 / 2004-10-04 (*mhs*)

Somehow a piece of junk slipped into the capture example, removed.

revision
0.7 / 2004-09-19 (*mhs*)

Fixed video standard selection, control enumeration, downscaling and aspect example. Added read and user pointer i/o to video capture example.

revision
0.6 / 2004-08-01 (*mhs*)

v4l2_buffer changes, added video capture example, various corrections.

revision
0.5 / 2003-11-05 (*mhs*)

Pixel format erratum.

revision
0.4 / 2003-09-17 (*mhs*)

Corrected source and Makefile to generate a PDF. SGML fixes. Added latest API changes. Closed gaps in the history chapter.

revision
0.3 / 2003-02-05 (*mhs*)

Another draft, more corrections.

revision
0.2 / 2003-01-15 (*mhs*)

Second draft, with corrections pointed out by Gerd Knorr.

revision
0.1 / 2002-12-01 (*mhs*)

First draft, based on documentation by Bill Dirks and discussions on the V4L mailing list.

8.3 Part II - Digital TV API

Note: This API is also known as Linux **DVB API**.

It it was originally written to support the European digital TV standard (DVB), and later extended to support all digital TV standards.

In order to avoid confusion, within this document, it was opted to refer to it, and to associated hardware as **Digital TV**.

The word **DVB** is reserved to be used for:

- the Digital TV API version (e. g. DVB API version 3 or DVB API version 5);
- digital TV data types (enums, structs, defines, etc);
- digital TV device nodes (`/dev/dvb/...`);
- the European DVB standard.

Version 5.10

8.3.1 Introduction

What you need to know

The reader of this document is required to have some knowledge in the area of digital video broadcasting (Digital TV) and should be familiar with part I of the MPEG2 specification ISO/IEC 13818 (aka ITU-T H.222), i.e you should know what a program/transport stream (PS/TS) is and what is meant by a packetized elementary stream (PES) or an I-frame.

Various Digital TV standards documents are available for download at:

- European standards (DVB): http://www.dvb.org and/or http://www.etsi.org.
- American standards (ATSC): https://www.atsc.org/standards/

- Japanese standards (ISDB): http://www.dibeg.org/

It is also necessary to know how to access Linux devices and how to use ioctl calls. This also includes the knowledge of C or C++.

History

The first API for Digital TV cards we used at Convergence in late 1999 was an extension of the Video4Linux API which was primarily developed for frame grabber cards. As such it was not really well suited to be used for Digital TV cards and their new features like recording MPEG streams and filtering several section and PES data streams at the same time.

In early 2000, Convergence was approached by Nokia with a proposal for a new standard Linux Digital TV API. As a commitment to the development of terminals based on open standards, Nokia and Convergence made it available to all Linux developers and published it on https://linuxtv.org in September 2000. With the Linux driver for the Siemens/Hauppauge DVB PCI card, Convergence provided a first implementation of the Linux Digital TV API. Convergence was the maintainer of the Linux Digital TV API in the early days.

Now, the API is maintained by the LinuxTV community (i.e. you, the reader of this document). The Linux Digital TV API is constantly reviewed and improved together with the improvements at the subsystem's core at the Kernel.

Overview

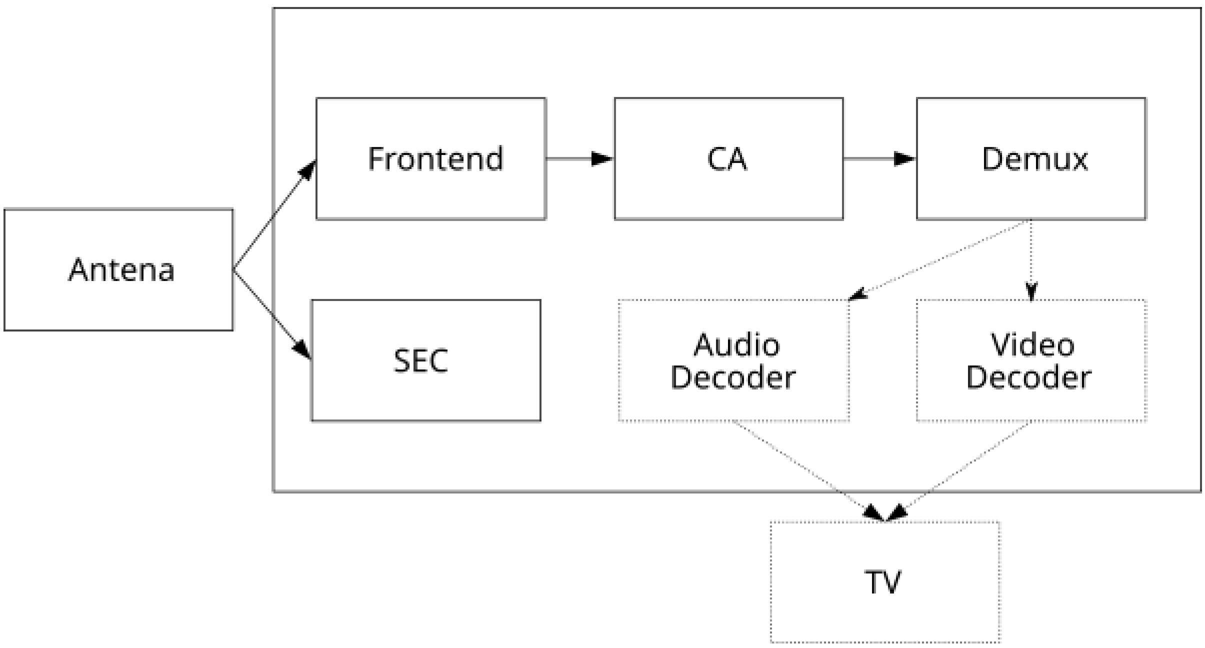

Fig. 19: Components of a Digital TV card/STB

A Digital TV card or set-top-box (STB) usually consists of the following main hardware components:

Frontend consisting of tuner and digital TV demodulator
Here the raw signal reaches the digital TV hardware from a satellite dish or

antenna or directly from cable. The frontend down-converts and demodulates this signal into an MPEG transport stream (TS). In case of a satellite frontend, this includes a facility for satellite equipment control (SEC), which allows control of LNB polarization, multi feed switches or dish rotors.

Conditional Access (CA) hardware like CI adapters and smartcard slots
The complete TS is passed through the CA hardware. Programs to which the user has access (controlled by the smart card) are decoded in real time and re-inserted into the TS.

Note: Not every digital TV hardware provides conditional access hardware.

Demultiplexer which filters the incoming Digital TV MPEG-TS stream
The demultiplexer splits the TS into its components like audio and video streams. Besides usually several of such audio and video streams it also contains data streams with information about the programs offered in this or other streams of the same provider.

Audio and video decoder
The main targets of the demultiplexer are audio and video decoders. After decoding, they pass on the uncompressed audio and video to the computer screen or to a TV set.

Note: Modern hardware usually doesn't have a separate decoder hardware, as such functionality can be provided by the main CPU, by the graphics adapter of the system or by a signal processing hardware embedded on a Systems on a Chip (SoC) integrated circuit.

It may also not be needed for certain usages (e.g. for data-only uses like "internet over satellite").

Components of a Digital TV card/STB shows a crude schematic of the control and data flow between those components.

Linux Digital TV Devices

The Linux Digital TV API lets you control these hardware components through currently six Unix-style character devices for video, audio, frontend, demux, CA and IP-over-DVB networking. The video and audio devices control the MPEG2 decoder hardware, the frontend device the tuner and the Digital TV demodulator. The demux device gives you control over the PES and section filters of the hardware. If the hardware does not support filtering these filters can be implemented in software. Finally, the CA device controls all the conditional access capabilities of the hardware. It can depend on the individual security requirements of the platform, if and how many of the CA functions are made available to the application through this device.

All devices can be found in the `/dev` tree under `/dev/dvb`. The individual devices are called:

- `/dev/dvb/adapterN/audioM`,

- /dev/dvb/adapterN/videoM,
- /dev/dvb/adapterN/frontendM,
- /dev/dvb/adapterN/netM,
- /dev/dvb/adapterN/demuxM,
- /dev/dvb/adapterN/dvrM,
- /dev/dvb/adapterN/caM,

where N enumerates the Digital TV cards in a system starting from 0, and M enumerates the devices of each type within each adapter, starting from 0, too. We will omit the "/dev/dvb/adapterN/" in the further discussion of these devices.

More details about the data structures and function calls of all the devices are described in the following chapters.

API include files

For each of the Digital TV devices a corresponding include file exists. The Digital TV API include files should be included in application sources with a partial path like:

```
#include <linux/dvb/ca.h>

#include <linux/dvb/dmx.h>

#include <linux/dvb/frontend.h>

#include <linux/dvb/net.h>
```

To enable applications to support different API version, an additional include file linux/dvb/version.h exists, which defines the constant DVB_API_VERSION. This document describes DVB_API_VERSION 5.10.

8.3.2 Digital TV Frontend API

The Digital TV frontend API was designed to support three groups of delivery systems: Terrestrial, cable and Satellite. Currently, the following delivery systems are supported:

- Terrestrial systems: DVB-T, DVB-T2, ATSC, ATSC M/H, ISDB-T, DVB-H, DTMB, CMMB
- Cable systems: DVB-C Annex A/C, ClearQAM (DVB-C Annex B)
- Satellite systems: DVB-S, DVB-S2, DVB Turbo, ISDB-S, DSS

The Digital TV frontend controls several sub-devices including:

- Tuner
- Digital TV demodulator
- Low noise amplifier (LNA)

- Satellite Equipment Control (SEC)[1].

The frontend can be accessed through `/dev/dvb/adapter?/frontend?`. Data types and ioctl definitions can be accessed by including `linux/dvb/frontend.h` in your application.

Note: Transmission via the internet (DVB-IP) and MMT (MPEG Media Transport) is not yet handled by this API but a future extension is possible.

Querying frontend information

Usually, the first thing to do when the frontend is opened is to check the frontend capabilities. This is done using *ioctl FE_GET_INFO*. This ioctl will enumerate the Digital TV API version and other characteristics about the frontend, and can be opened either in read only or read/write mode.

Querying frontend status and statistics

Once *FE_SET_PROPERTY* is called, the frontend will run a kernel thread that will periodically check for the tuner lock status and provide statistics about the quality of the signal.

The information about the frontend tuner locking status can be queried using *ioctl FE_READ_STATUS*.

Signal statistics are provided via *ioctl FE_SET_PROPERTY, FE_GET_PROPERTY*.

Note: Most statistics require the demodulator to be fully locked (e. g. with *FE_HAS_LOCK* bit set). See *Frontend statistics indicators* for more details.

Property types

Tuning into a Digital TV physical channel and starting decoding it requires changing a set of parameters, in order to control the tuner, the demodulator, the Linear Low-noise Amplifier (LNA) and to set the antenna subsystem via Satellite Equipment Control - SEC (on satellite systems). The actual parameters are specific to each particular digital TV standards, and may change as the digital TV specs evolves.

In the past (up to DVB API version 3 - DVBv3), the strategy used was to have a union with the parameters needed to tune for DVB-S, DVB-C, DVB-T and ATSC delivery systems grouped there. The problem is that, as the second generation standards appeared, the size of such union was not big enough to group the structs

[1] On Satellite systems, the API support for the Satellite Equipment Control (SEC) allows to power control and to send/receive signals to control the antenna subsystem, selecting the polarization and choosing the Intermediate Frequency IF) of the Low Noise Block Converter Feed Horn (LNBf). It supports the DiSEqC and V-SEC protocols. The DiSEqC (digital SEC) specification is available at Eutelsat.

that would be required for those new standards. Also, extending it would break userspace.

So, the legacy union/struct based approach was deprecated, in favor of a properties set approach. On such approach, *FE_GET_PROPERTY and FE_SET_PROPERTY* are used to setup the frontend and read its status.

The actual action is determined by a set of dtv_property cmd/data pairs. With one single ioctl, is possible to get/set up to 64 properties.

This section describes the new and recommended way to set the frontend, with supports all digital TV delivery systems.

Note:

1. On Linux DVB API version 3, setting a frontend was done via struct *dvb_frontend_parameters*.
2. Don't use DVB API version 3 calls on hardware with supports newer standards. Such API provides no support or a very limited support to new standards and/or new hardware.
3. Nowadays, most frontends support multiple delivery systems. Only with DVB API version 5 calls it is possible to switch between the multiple delivery systems supported by a frontend.
4. DVB API version 5 is also called *S2API*, as the first new standard added to it was DVB-S2.

Example: in order to set the hardware to tune into a DVB-C channel at 651 kHz, modulated with 256-QAM, FEC 3/4 and symbol rate of 5.217 Mbauds, those properties should be sent to *FE_SET_PROPERTY* ioctl:

DTV_DELIVERY_SYSTEM = SYS_DVBC_ANNEX_A

DTV_FREQUENCY = 651000000

DTV_MODULATION = QAM_256

DTV_INVERSION = INVERSION_AUTO

DTV_SYMBOL_RATE = 5217000

DTV_INNER_FEC = FEC_3_4

DTV_TUNE

The code that would that would do the above is show in *Example: Setting digital TV frontend properties*.

Listing 1: Example: Setting digital TV frontend properties

```
#include <stdio.h>
#include <fcntl.h>
#include <sys/ioctl.h>
#include <linux/dvb/frontend.h>
```

(continues on next page)

(continued from previous page)

```
static struct dtv_property props[] = {
    { .cmd = DTV_DELIVERY_SYSTEM, .u.data = SYS_DVBC_ANNEX_A },
    { .cmd = DTV_FREQUENCY,       .u.data = 651000000 },
    { .cmd = DTV_MODULATION,      .u.data = QAM_256 },
    { .cmd = DTV_INVERSION,       .u.data = INVERSION_AUTO },
    { .cmd = DTV_SYMBOL_RATE,     .u.data = 5217000 },
    { .cmd = DTV_INNER_FEC,       .u.data = FEC_3_4 },
    { .cmd = DTV_TUNE }
};

static struct dtv_properties dtv_prop = {
    .num = 6, .props = props
};

int main(void)
{
    int fd = open("/dev/dvb/adapter0/frontend0", O_RDWR);

    if (!fd) {
        perror ("open");
        return -1;
    }
    if (ioctl(fd, FE_SET_PROPERTY, &dtv_prop) == -1) {
        perror("ioctl");
        return -1;
    }
    printf("Frontend set\\n");
    return 0;
}
```

Attention: While it is possible to directly call the Kernel code like the above example, it is strongly recommended to use libdvbv5, as it provides abstraction to work with the supported digital TV standards and provides methods for usual operations like program scanning and to read/write channel descriptor files.

Digital TV property parameters

There are several different Digital TV parameters that can be used by *FE_SET_PROPERTY and FE_GET_PROPERTY ioctls*. This section describes each of them. Please notice, however, that only a subset of them are needed to setup a frontend.

DTV_UNDEFINED

Used internally. A GET/SET operation for it won't change or return anything.

DTV_TUNE

Interpret the cache of data, build either a traditional frontend tunerequest so we can pass validation in the `FE_SET_FRONTEND` ioctl.

DTV_CLEAR

Reset a cache of data specific to the frontend here. This does not effect hardware.

DTV_FREQUENCY

Frequency of the digital TV transponder/channel.

Note:

1. For satellite delivery systems, the frequency is in kHz.
2. For cable and terrestrial delivery systems, the frequency is in Hz.
3. On most delivery systems, the frequency is the center frequency of the transponder/channel. The exception is for ISDB-T, where the main carrier has a 1/7 offset from the center.
4. For ISDB-T, the channels are usually transmitted with an offset of about 143kHz. E.g. a valid frequency could be 474,143 kHz. The stepping is bound to the bandwidth of the channel which is typically 6MHz.
5. In ISDB-Tsb, the channel consists of only one or three segments the frequency step is 429kHz, 3*429 respectively.

DTV_MODULATION

Specifies the frontend modulation type for delivery systems that supports more multiple modulations.

The modulation can be one of the types defined by enum *fe_modulation*.

Most of the digital TV standards offers more than one possible modulation type.

The table below presents a summary of the types of modulation types supported by each delivery system, as currently defined by specs.

Standard	Modulation types
ATSC (version 1)	8-VSB and 16-VSB.
DMTB	4-QAM, 16-QAM, 32-QAM, 64-QAM and 4-QAM-NR.
DVB-C Annex A/C	16-QAM, 32-QAM, 64-QAM and 256-QAM.
DVB-C Annex B	64-QAM.
DVB-T	QPSK, 16-QAM and 64-QAM.
DVB-T2	QPSK, 16-QAM, 64-QAM and 256-QAM.
DVB-S	No need to set. It supports only QPSK.
DVB-S2	QPSK, 8-PSK, 16-APSK and 32-APSK.
ISDB-T	QPSK, DQPSK, 16-QAM and 64-QAM.
ISDB-S	8-PSK, QPSK and BPSK.

Note: Please notice that some of the above modulation types may not be defined currently at the Kernel. The reason is simple: no driver needed such definition yet.

DTV_BANDWIDTH_HZ

Bandwidth for the channel, in HZ.

Should be set only for terrestrial delivery systems.

Possible values: `1712000`, `5000000`, `6000000`, `7000000`, `8000000`, `10000000`.

Terrestrial Standard	Possible values for bandwidth
ATSC (version 1)	No need to set. It is always 6MHz.
DMTB	No need to set. It is always 8MHz.
DVB-T	6MHz, 7MHz and 8MHz.
DVB-T2	1.172 MHz, 5MHz, 6MHz, 7MHz, 8MHz and 10MHz
ISDB-T	5MHz, 6MHz, 7MHz and 8MHz, although most places use 6MHz.

Note:

1. For ISDB-Tsb, the bandwidth can vary depending on the number of connected segments.

   It can be easily derived from other parameters (DTV_ISDBT_SB_SEGMENT_IDX, DTV_ISDBT_SB_SEGMENT_COUNT).

2. On Satellite and Cable delivery systems, the bandwidth depends on the symbol rate. So, the Kernel will silently ignore any setting *DTV_BANDWIDTH_HZ*. I will however fill it back with a bandwidth estimation.

   Such bandwidth estimation takes into account the symbol rate set with *DTV_SYMBOL_RATE*, and the rolloff factor, with is fixed for DVB-C and DVB-S.

   For DVB-S2, the rolloff should also be set via *DTV_ROLLOFF*.

DTV_INVERSION

Specifies if the frontend should do spectral inversion or not.

The acceptable values are defined by `fe_spectral_inversion`.

DTV_DISEQC_MASTER

Currently not implemented.

DTV_SYMBOL_RATE

Used on cable and satellite delivery systems.

Digital TV symbol rate, in bauds (symbols/second).

DTV_INNER_FEC

Used on cable and satellite delivery systems.

The acceptable values are defined by `fe_code_rate`.

DTV_VOLTAGE

Used on satellite delivery systems.

The voltage is usually used with non-DiSEqC capable LNBs to switch the polarzation (horizontal/vertical). When using DiSEqC epuipment this voltage has to be switched consistently to the DiSEqC commands as described in the DiSEqC spec.

The acceptable values are defined by `fe_sec_voltage`.

DTV_TONE

Currently not used.

DTV_PILOT

Used on DVB-S2.

Sets DVB-S2 pilot.

The acceptable values are defined by *fe_pilot*.

DTV_ROLLOFF

Used on DVB-S2.

Sets DVB-S2 rolloff.

The acceptable values are defined by *fe_rolloff*.

DTV_DISEQC_SLAVE_REPLY

Currently not implemented.

DTV_FE_CAPABILITY_COUNT

Currently not implemented.

DTV_FE_CAPABILITY

Currently not implemented.

DTV_DELIVERY_SYSTEM

Specifies the type of the delivery system.

The acceptable values are defined by *fe_delivery_system*.

DTV_ISDBT_PARTIAL_RECEPTION

Used only on ISDB.

If `DTV_ISDBT_SOUND_BROADCASTING` is '0' this bit-field represents whether the channel is in partial reception mode or not.

If '1' `DTV_ISDBT_LAYERA_*` values are assigned to the center segment and `DTV_ISDBT_LAYERA_SEGMENT_COUNT` has to be '1'.

If in addition DTV_ISDBT_SOUND_BROADCASTING is '1' DTV_ISDBT_PARTIAL_RECEPTION represents whether this ISDB-Tsb channel is consisting of one segment and layer or three segments and two layers.

Possible values: 0, 1, -1 (AUTO)

DTV_ISDBT_SOUND_BROADCASTING

Used only on ISDB.

This field represents whether the other DTV_ISDBT_*-parameters are referring to an ISDB-T and an ISDB-Tsb channel. (See also DTV_ISDBT_PARTIAL_RECEPTION).

Possible values: 0, 1, -1 (AUTO)

DTV_ISDBT_SB_SUBCHANNEL_ID

Used only on ISDB.

This field only applies if DTV_ISDBT_SOUND_BROADCASTING is '1'.

(Note of the author: This might not be the correct description of the SUBCHANNEL-ID in all details, but it is my understanding of the technical background needed to program a device)

An ISDB-Tsb channel (1 or 3 segments) can be broadcasted alone or in a set of connected ISDB-Tsb channels. In this set of channels every channel can be received independently. The number of connected ISDB-Tsb segment can vary, e.g. depending on the frequency spectrum bandwidth available.

Example: Assume 8 ISDB-Tsb connected segments are broadcasted. The broadcaster has several possibilities to put those channels in the air: Assuming a normal 13-segment ISDB-T spectrum he can align the 8 segments from position 1-8 to 5-13 or anything in between.

The underlying layer of segments are subchannels: each segment is consisting of several subchannels with a predefined IDs. A sub-channel is used to help the demodulator to synchronize on the channel.

An ISDB-T channel is always centered over all sub-channels. As for the example above, in ISDB-Tsb it is no longer as simple as that.

The DTV_ISDBT_SB_SUBCHANNEL_ID parameter is used to give the sub-channel ID of the segment to be demodulated.

Possible values: 0 .. 41, -1 (AUTO)

DTV_ISDBT_SB_SEGMENT_IDX

Used only on ISDB.

This field only applies if `DTV_ISDBT_SOUND_BROADCASTING` is '1'.

`DTV_ISDBT_SB_SEGMENT_IDX` gives the index of the segment to be demodulated for an ISDB-Tsb channel where several of them are transmitted in the connected manner.

Possible values: 0 .. `DTV_ISDBT_SB_SEGMENT_COUNT` - 1

Note: This value cannot be determined by an automatic channel search.

DTV_ISDBT_SB_SEGMENT_COUNT

Used only on ISDB.

This field only applies if `DTV_ISDBT_SOUND_BROADCASTING` is '1'.

`DTV_ISDBT_SB_SEGMENT_COUNT` gives the total count of connected ISDB-Tsb channels.

Possible values: 1 .. 13

Note: This value cannot be determined by an automatic channel search.

DTV-ISDBT-LAYER[A-C] parameters

Used only on ISDB.

ISDB-T channels can be coded hierarchically. As opposed to DVB-T in ISDB-T hierarchical layers can be decoded simultaneously. For that reason a ISDB-T demodulator has 3 Viterbi and 3 Reed-Solomon decoders.

ISDB-T has 3 hierarchical layers which each can use a part of the available segments. The total number of segments over all layers has to 13 in ISDB-T.

There are 3 parameter sets, for Layers A, B and C.

DTV_ISDBT_LAYER_ENABLED

Used only on ISDB.

Hierarchical reception in ISDB-T is achieved by enabling or disabling layers in the decoding process. Setting all bits of `DTV_ISDBT_LAYER_ENABLED` to '1' forces all layers (if applicable) to be demodulated. This is the default.

If the channel is in the partial reception mode (`DTV_ISDBT_PARTIAL_RECEPTION` = 1) the central segment can be decoded independently of the other 12 segments. In that mode layer A has to have a `SEGMENT_COUNT` of 1.

In ISDB-Tsb only layer A is used, it can be 1 or 3 in ISDB-Tsb according to `DTV_ISDBT_PARTIAL_RECEPTION`. `SEGMENT_COUNT` must be filled accordingly.

Only the values of the first 3 bits are used. Other bits will be silently ignored:

DTV_ISDBT_LAYER_ENABLED bit 0: layer A enabled

DTV_ISDBT_LAYER_ENABLED bit 1: layer B enabled

DTV_ISDBT_LAYER_ENABLED bit 2: layer C enabled

DTV_ISDBT_LAYER_ENABLED bits 3-31: unused

DTV_ISDBT_LAYER[A-C]_FEC

Used only on ISDB.

The Forward Error Correction mechanism used by a given ISDB Layer, as defined by *fe_code_rate*.

Possible values are: FEC_AUTO, FEC_1_2, FEC_2_3, FEC_3_4, FEC_5_6, FEC_7_8

DTV_ISDBT_LAYER[A-C]_MODULATION

Used only on ISDB.

The modulation used by a given ISDB Layer, as defined by *fe_modulation*.

Possible values are: QAM_AUTO, QPSK, QAM_16, QAM_64, DQPSK

Note:

1. If layer C is DQPSK, then layer B has to be DQPSK.
2. If layer B is DQPSK and DTV_ISDBT_PARTIAL_RECEPTION= 0, then layer has to be DQPSK.

DTV_ISDBT_LAYER[A-C]_SEGMENT_COUNT

Used only on ISDB.

Possible values: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, -1 (AUTO)

Note: Truth table for DTV_ISDBT_SOUND_BROADCASTING and DTV_ISDBT_PARTIAL_RECEPTION and LAYER[A-C]_SEGMENT_COUNT

Table 223: Truth table for ISDB-T Sound Broadcasting

Partial Reception	Sound Broadcasting	Layer A width	Layer B width	Layer C width	total width
0	0	1 .. 13	1 .. 13	1 .. 13	13
1	0	1	1 .. 13	1 .. 13	13
0	1	1	0	0	1
1	1	1	2	0	13

DTV_ISDBT_LAYER[A-C]_TIME_INTERLEAVING

Used only on ISDB.

Valid values: 0, 1, 2, 4, -1 (AUTO)

when DTV_ISDBT_SOUND_BROADCASTING is active, value 8 is also valid.

Note: The real time interleaving length depends on the mode (fft-size). The values here are referring to what can be found in the TMCC-structure, as shown in the table below.

type **isdbt_layer_interleaving_table**

Table 224: ISDB-T time interleaving modes

DTV_ISDBT_LAYER[A-C]_TIME_INTERLI	Mode 1 (2K FFT)	Mode 2 (4K FFT)	Mode 3 (8K FFT)
0	0	0	0
1	4	2	1
2	8	4	2
4	16	8	4

DTV_ATSCMH_FIC_VER

Used only on ATSC-MH.

Version number of the FIC (Fast Information Channel) signaling data.

FIC is used for relaying information to allow rapid service acquisition by the receiver.

Possible values: 0, 1, 2, 3, ⋯, 30, 31

DTV_ATSCMH_PARADE_ID

Used only on ATSC-MH.

Parade identification number

A parade is a collection of up to eight MH groups, conveying one or two ensembles.

Possible values: 0, 1, 2, 3, ⋯, 126, 127

DTV_ATSCMH_NOG

Used only on ATSC-MH.

Number of MH groups per MH subframe for a designated parade.

Possible values: 1, 2, 3, 4, 5, 6, 7, 8

DTV_ATSCMH_TNOG

Used only on ATSC-MH.

Total number of MH groups including all MH groups belonging to all MH parades in one MH subframe.

Possible values: 0, 1, 2, 3, ⋯, 30, 31

DTV_ATSCMH_SGN

Used only on ATSC-MH.

Start group number.

Possible values: 0, 1, 2, 3, ⋯, 14, 15

DTV_ATSCMH_PRC

Used only on ATSC-MH.

Parade repetition cycle.

Possible values: 1, 2, 3, 4, 5, 6, 7, 8

DTV_ATSCMH_RS_FRAME_MODE

Used only on ATSC-MH.

Reed Solomon (RS) frame mode.

The acceptable values are defined by *atscmh_rs_frame_mode*.

DTV_ATSCMH_RS_FRAME_ENSEMBLE

Used only on ATSC-MH.

Reed Solomon(RS) frame ensemble.

The acceptable values are defined by *atscmh_rs_frame_ensemble*.

DTV_ATSCMH_RS_CODE_MODE_PRI

Used only on ATSC-MH.

Reed Solomon (RS) code mode (primary).

The acceptable values are defined by *atscmh_rs_code_mode*.

DTV_ATSCMH_RS_CODE_MODE_SEC

Used only on ATSC-MH.

Reed Solomon (RS) code mode (secondary).

The acceptable values are defined by *atscmh_rs_code_mode*.

DTV_ATSCMH_SCCC_BLOCK_MODE

Used only on ATSC-MH.

Series Concatenated Convolutional Code Block Mode.

The acceptable values are defined by *atscmh_sccc_block_mode*.

DTV_ATSCMH_SCCC_CODE_MODE_A

Used only on ATSC-MH.

Series Concatenated Convolutional Code Rate.

The acceptable values are defined by *atscmh_sccc_code_mode*.

DTV_ATSCMH_SCCC_CODE_MODE_B

Used only on ATSC-MH.

Series Concatenated Convolutional Code Rate.

Possible values are the same as documented on enum *atscmh_sccc_code_mode*.

DTV_ATSCMH_SCCC_CODE_MODE_C

Used only on ATSC-MH.

Series Concatenated Convolutional Code Rate.

Possible values are the same as documented on enum *atscmh_sccc_code_mode*.

DTV_ATSCMH_SCCC_CODE_MODE_D

Used only on ATSC-MH.

Series Concatenated Convolutional Code Rate.

Possible values are the same as documented on enum *atscmh_sccc_code_mode*.

DTV_API_VERSION

Returns the major/minor version of the Digital TV API

DTV_CODE_RATE_HP

Used on terrestrial transmissions.

The acceptable values are defined by *fe_transmit_mode*.

DTV_CODE_RATE_LP

Used on terrestrial transmissions.

The acceptable values are defined by *fe_transmit_mode*.

DTV_GUARD_INTERVAL

The acceptable values are defined by *fe_guard_interval*.

Note:

1. If `DTV_GUARD_INTERVAL` is set the `GUARD_INTERVAL_AUTO` the hardware will try to find the correct guard interval (if capable) and will use TMCC to fill in the missing parameters.
2. Intervals `GUARD_INTERVAL_1_128`, `GUARD_INTERVAL_19_128` and `GUARD_INTERVAL_19_256` are used only for DVB-T2 at present.
3. Intervals `GUARD_INTERVAL_PN420`, `GUARD_INTERVAL_PN595` and `GUARD_INTERVAL_PN945` are used only for DMTB at the present. On such standard, only those intervals and `GUARD_INTERVAL_AUTO` are valid.

DTV_TRANSMISSION_MODE

Used only on OFTM-based standards, e. g. DVB-T/T2, ISDB-T, DTMB.

Specifies the FFT size (with corresponds to the approximate number of carriers) used by the standard.

The acceptable values are defined by *fe_transmit_mode*.

Note:

1. ISDB-T supports three carrier/symbol-size: 8K, 4K, 2K. It is called **mode** on such standard, and are numbered from 1 to 3:

Mode	FFT size	Transmission mode
1	2K	`TRANSMISSION_MODE_2K`
2	4K	`TRANSMISSION_MODE_4K`
3	8K	`TRANSMISSION_MODE_8K`

2. If `DTV_TRANSMISSION_MODE` is set the `TRANSMISSION_MODE_AUTO` the hardware will try to find the correct FFT-size (if capable) and will use TMCC to fill in the missing parameters.
3. DVB-T specifies 2K and 8K as valid sizes.
4. DVB-T2 specifies 1K, 2K, 4K, 8K, 16K and 32K.
5. DTMB specifies C1 and C3780.

DTV_HIERARCHY

Used only on DVB-T and DVB-T2.

Frontend hierarchy.

The acceptable values are defined by *fe_hierarchy*.

DTV_STREAM_ID

Used on DVB-S2, DVB-T2 and ISDB-S.

DVB-S2, DVB-T2 and ISDB-S support the transmission of several streams on a single transport stream. This property enables the digital TV driver to handle substream filtering, when supported by the hardware. By default, substream filtering is disabled.

For DVB-S2 and DVB-T2, the valid substream id range is from 0 to 255.

For ISDB, the valid substream id range is from 1 to 65535.

To disable it, you should use the special macro NO_STREAM_ID_FILTER.

Note: any value outside the id range also disables filtering.

DTV_DVBT2_PLP_ID_LEGACY

Obsolete, replaced with DTV_STREAM_ID.

DTV_ENUM_DELSYS

A Multi standard frontend needs to advertise the delivery systems provided. Applications need to enumerate the provided delivery systems, before using any other operation with the frontend. Prior to it's introduction, FE_GET_INFO was used to determine a frontend type. A frontend which provides more than a single delivery system, FE_GET_INFO doesn't help much. Applications which intends to use a multistandard frontend must enumerate the delivery systems associated with it, rather than trying to use FE_GET_INFO. In the case of a legacy frontend, the result is just the same as with FE_GET_INFO, but in a more structured format

The acceptable values are defined by *fe_delivery_system*.

DTV_INTERLEAVING

Time interleaving to be used.

The acceptable values are defined by *fe_interleaving*.

DTV_LNA

Low-noise amplifier.

Hardware might offer controllable LNA which can be set manually using that parameter. Usually LNA could be found only from terrestrial devices if at all.

Possible values: 0, 1, LNA_AUTO

0, LNA off

1, LNA on

use the special macro LNA_AUTO to set LNA auto

DTV_SCRAMBLING_SEQUENCE_INDEX

Used on DVB-S2.

This 18 bit field, when present, carries the index of the DVB-S2 physical layer scrambling sequence as defined in clause 5.5.4 of EN 302 307. There is no explicit signalling method to convey scrambling sequence index to the receiver. If S2 satellite delivery system descriptor is available it can be used to read the scrambling sequence index (EN 300 468 table 41).

By default, gold scrambling sequence index 0 is used.

The valid scrambling sequence index range is from 0 to 262142.

Frontend statistics indicators

The values are returned via `dtv_property.stat`. If the property is supported, `dtv_property.stat.len` is bigger than zero.

For most delivery systems, `dtv_property.stat.len` will be 1 if the stats is supported, and the properties will return a single value for each parameter.

It should be noted, however, that new OFDM delivery systems like ISDB can use different modulation types for each group of carriers. On such standards, up to 3 groups of statistics can be provided, and `dtv_property.stat.len` is updated to reflect the "global" metrics, plus one metric per each carrier group (called "layer" on ISDB).

So, in order to be consistent with other delivery systems, the first value at *`dtv_property.stat.dtv_stats`* array refers to the global metric. The other elements of the array represent each layer, starting from layer A(index 1), layer B (index 2) and so on.

The number of filled elements are stored at `dtv_property.stat.len`.

Each element of the `dtv_property.stat.dtv_stats` array consists on two elements:

- `svalue` or `uvalue`, where `svalue` is for signed values of the measure (dB measures) and `uvalue` is for unsigned values (counters, relative scale)
- `scale` - Scale for the value. It can be:
  - `FE_SCALE_NOT_AVAILABLE` - The parameter is supported by the frontend, but it was not possible to collect it (could be a transitory or permanent condition)
  - `FE_SCALE_DECIBEL` - parameter is a signed value, measured in 1/1000 dB
  - `FE_SCALE_RELATIVE` - parameter is a unsigned value, where 0 means 0% and 65535 means 100%.
  - `FE_SCALE_COUNTER` - parameter is a unsigned value that counts the occurrence of an event, like bit error, block error, or lapsed time.

DTV_STAT_SIGNAL_STRENGTH

Indicates the signal strength level at the analog part of the tuner or of the demod.

Possible scales for this metric are:

- `FE_SCALE_NOT_AVAILABLE` - it failed to measure it, or the measurement was not complete yet.
- `FE_SCALE_DECIBEL` - signal strength is in 0.001 dBm units, power measured in miliwatts. This value is generally negative.
- `FE_SCALE_RELATIVE` - The frontend provides a 0% to 100% measurement for power (actually, 0 to 65535).

DTV_STAT_CNR

Indicates the Signal to Noise ratio for the main carrier.

Possible scales for this metric are:

- `FE_SCALE_NOT_AVAILABLE` - it failed to measure it, or the measurement was not complete yet.
- `FE_SCALE_DECIBEL` - Signal/Noise ratio is in 0.001 dB units.
- `FE_SCALE_RELATIVE` - The frontend provides a 0% to 100% measurement for Signal/Noise (actually, 0 to 65535).

DTV_STAT_PRE_ERROR_BIT_COUNT

Measures the number of bit errors before the forward error correction (FEC) on the inner coding block (before Viterbi, LDPC or other inner code).

This measure is taken during the same interval as `DTV_STAT_PRE_TOTAL_BIT_COUNT`.

In order to get the BER (Bit Error Rate) measurement, it should be divided by *DTV_STAT_PRE_TOTAL_BIT_COUNT*.

This measurement is monotonically increased, as the frontend gets more bit count measurements. The frontend may reset it when a channel/transponder is tuned.

Possible scales for this metric are:

- `FE_SCALE_NOT_AVAILABLE` - it failed to measure it, or the measurement was not complete yet.
- `FE_SCALE_COUNTER` - Number of error bits counted before the inner coding.

DTV_STAT_PRE_TOTAL_BIT_COUNT

Measures the amount of bits received before the inner code block, during the same period as *DTV_STAT_PRE_ERROR_BIT_COUNT* measurement was taken.

It should be noted that this measurement can be smaller than the total amount of bits on the transport stream, as the frontend may need to manually restart the measurement, losing some data between each measurement interval.

This measurement is monotonically increased, as the frontend gets more bit count measurements. The frontend may reset it when a channel/transponder is tuned.

Possible scales for this metric are:

- `FE_SCALE_NOT_AVAILABLE` - it failed to measure it, or the measurement was not complete yet.
- `FE_SCALE_COUNTER` - Number of bits counted while measuring *DTV_STAT_PRE_ERROR_BIT_COUNT*.

DTV_STAT_POST_ERROR_BIT_COUNT

Measures the number of bit errors after the forward error correction (FEC) done by inner code block (after Viterbi, LDPC or other inner code).

This measure is taken during the same interval as `DTV_STAT_POST_TOTAL_BIT_COUNT`.

In order to get the BER (Bit Error Rate) measurement, it should be divided by *DTV_STAT_POST_TOTAL_BIT_COUNT*.

This measurement is monotonically increased, as the frontend gets more bit count measurements. The frontend may reset it when a channel/transponder is tuned.

Possible scales for this metric are:

- `FE_SCALE_NOT_AVAILABLE` - it failed to measure it, or the measurement was not complete yet.
- `FE_SCALE_COUNTER` - Number of error bits counted after the inner coding.

DTV_STAT_POST_TOTAL_BIT_COUNT

Measures the amount of bits received after the inner coding, during the same period as *DTV_STAT_POST_ERROR_BIT_COUNT* measurement was taken.

It should be noted that this measurement can be smaller than the total amount of bits on the transport stream, as the frontend may need to manually restart the measurement, losing some data between each measurement interval.

This measurement is monotonically increased, as the frontend gets more bit count measurements. The frontend may reset it when a channel/transponder is tuned.

Possible scales for this metric are:

- `FE_SCALE_NOT_AVAILABLE` - it failed to measure it, or the measurement was not complete yet.
- `FE_SCALE_COUNTER` - Number of bits counted while measuring *DTV_STAT_POST_ERROR_BIT_COUNT*.

DTV_STAT_ERROR_BLOCK_COUNT

Measures the number of block errors after the outer forward error correction coding (after Reed-Solomon or other outer code).

This measurement is monotonically increased, as the frontend gets more bit count measurements. The frontend may reset it when a channel/transponder is tuned.

Possible scales for this metric are:

- `FE_SCALE_NOT_AVAILABLE` - it failed to measure it, or the measurement was not complete yet.
- `FE_SCALE_COUNTER` - Number of error blocks counted after the outer coding.

DTV-STAT_TOTAL_BLOCK_COUNT

Measures the total number of blocks received during the same period as *DTV_STAT_ERROR_BLOCK_COUNT* measurement was taken.

It can be used to calculate the PER indicator, by dividing *DTV_STAT_ERROR_BLOCK_COUNT* by *DTV-STAT_TOTAL_BLOCK_COUNT*.

Possible scales for this metric are:

- `FE_SCALE_NOT_AVAILABLE` - it failed to measure it, or the measurement was not complete yet.
- `FE_SCALE_COUNTER` - Number of blocks counted while measuring *DTV_STAT_ERROR_BLOCK_COUNT*.

Properties used on terrestrial delivery systems

DVB-T delivery system

The following parameters are valid for DVB-T:

- *DTV_API_VERSION*
- *DTV_DELIVERY_SYSTEM*
- *DTV_TUNE*
- *DTV_CLEAR*
- *DTV_FREQUENCY*
- *DTV_MODULATION*
- *DTV_BANDWIDTH_HZ*
- *DTV_INVERSION*
- *DTV_CODE_RATE_HP*
- *DTV_CODE_RATE_LP*
- *DTV_GUARD_INTERVAL*
- *DTV_TRANSMISSION_MODE*
- *DTV_HIERARCHY*
- *DTV_LNA*

In addition, the *DTV QoS statistics* are also valid.

DVB-T2 delivery system

DVB-T2 support is currently in the early stages of development, so expect that this section maygrow and become more detailed with time.

The following parameters are valid for DVB-T2:

- *DTV_API_VERSION*
- *DTV_DELIVERY_SYSTEM*
- *DTV_TUNE*
- *DTV_CLEAR*
- *DTV_FREQUENCY*
- *DTV_MODULATION*
- *DTV_BANDWIDTH_HZ*
- *DTV_INVERSION*
- *DTV_CODE_RATE_HP*
- *DTV_CODE_RATE_LP*
- *DTV_GUARD_INTERVAL*
- *DTV_TRANSMISSION_MODE*
- *DTV_HIERARCHY*
- *DTV_STREAM_ID*
- *DTV_LNA*

In addition, the *DTV QoS statistics* are also valid.

ISDB-T delivery system

This ISDB-T/ISDB-Tsb API extension should reflect all information needed to tune any ISDB-T/ISDB-Tsb hardware. Of course it is possible that some very sophisticated devices won't need certain parameters to tune.

The information given here should help application writers to know how to handle ISDB-T and ISDB-Tsb hardware using the Linux Digital TV API.

The details given here about ISDB-T and ISDB-Tsb are just enough to basically show the dependencies between the needed parameter values, but surely some information is left out. For more detailed information see the following documents:

ARIB STD-B31 - "Transmission System for Digital Terrestrial Television Broadcasting" and

ARIB TR-B14 - "Operational Guidelines for Digital Terrestrial Television Broadcasting".

In order to understand the ISDB specific parameters, one has to have some knowledge the channel structure in ISDB-T and ISDB-Tsb. I.e. it has to be known to the reader that an ISDB-T channel consists of 13 segments, that it can have up to 3 layer sharing those segments, and things like that.

The following parameters are valid for ISDB-T:

- *DTV_API_VERSION*
- *DTV_DELIVERY_SYSTEM*
- *DTV_TUNE*
- *DTV_CLEAR*
- *DTV_FREQUENCY*
- *DTV_BANDWIDTH_HZ*
- *DTV_INVERSION*
- *DTV_GUARD_INTERVAL*
- *DTV_TRANSMISSION_MODE*
- *DTV_ISDBT_LAYER_ENABLED*
- *DTV_ISDBT_PARTIAL_RECEPTION*
- *DTV_ISDBT_SOUND_BROADCASTING*
- *DTV_ISDBT_SB_SUBCHANNEL_ID*
- *DTV_ISDBT_SB_SEGMENT_IDX*
- *DTV_ISDBT_SB_SEGMENT_COUNT*
- *DTV_ISDBT_LAYERA_FEC*
- *DTV_ISDBT_LAYERA_MODULATION*
- *DTV_ISDBT_LAYERA_SEGMENT_COUNT*
- *DTV_ISDBT_LAYERA_TIME_INTERLEAVING*
- *DTV_ISDBT_LAYERB_FEC*
- *DTV_ISDBT_LAYERB_MODULATION*
- *DTV_ISDBT_LAYERB_SEGMENT_COUNT*
- *DTV_ISDBT_LAYERB_TIME_INTERLEAVING*
- *DTV_ISDBT_LAYERC_FEC*
- *DTV_ISDBT_LAYERC_MODULATION*
- *DTV_ISDBT_LAYERC_SEGMENT_COUNT*
- *DTV_ISDBT_LAYERC_TIME_INTERLEAVING*

In addition, the *DTV QoS statistics* are also valid.

ATSC delivery system

The following parameters are valid for ATSC:

- *DTV_API_VERSION*
- *DTV_DELIVERY_SYSTEM*
- *DTV_TUNE*
- *DTV_CLEAR*
- *DTV_FREQUENCY*
- *DTV_MODULATION*
- *DTV_BANDWIDTH_HZ*

In addition, the *DTV QoS statistics* are also valid.

ATSC-MH delivery system

The following parameters are valid for ATSC-MH:

- *DTV_API_VERSION*
- *DTV_DELIVERY_SYSTEM*
- *DTV_TUNE*
- *DTV_CLEAR*
- *DTV_FREQUENCY*
- *DTV_BANDWIDTH_HZ*
- *DTV_ATSCMH_FIC_VER*
- *DTV_ATSCMH_PARADE_ID*
- *DTV_ATSCMH_NOG*
- *DTV_ATSCMH_TNOG*
- *DTV_ATSCMH_SGN*
- *DTV_ATSCMH_PRC*
- *DTV_ATSCMH_RS_FRAME_MODE*
- *DTV_ATSCMH_RS_FRAME_ENSEMBLE*
- *DTV_ATSCMH_RS_CODE_MODE_PRI*
- *DTV_ATSCMH_RS_CODE_MODE_SEC*
- *DTV_ATSCMH_SCCC_BLOCK_MODE*
- *DTV_ATSCMH_SCCC_CODE_MODE_A*
- *DTV_ATSCMH_SCCC_CODE_MODE_B*
- *DTV_ATSCMH_SCCC_CODE_MODE_C*

- *DTV_ATSCMH_SCCC_CODE_MODE_D*

In addition, the *DTV QoS statistics* are also valid.

DTMB delivery system

The following parameters are valid for DTMB:

- *DTV_API_VERSION*
- *DTV_DELIVERY_SYSTEM*
- *DTV_TUNE*
- *DTV_CLEAR*
- *DTV_FREQUENCY*
- *DTV_MODULATION*
- *DTV_BANDWIDTH_HZ*
- *DTV_INVERSION*
- *DTV_INNER_FEC*
- *DTV_GUARD_INTERVAL*
- *DTV_TRANSMISSION_MODE*
- *DTV_INTERLEAVING*
- *DTV_LNA*

In addition, the *DTV QoS statistics* are also valid.

Properties used on cable delivery systems

DVB-C delivery system

The DVB-C Annex-A is the widely used cable standard. Transmission uses QAM modulation.

The DVB-C Annex-C is optimized for 6MHz, and is used in Japan. It supports a subset of the Annex A modulation types, and a roll-off of 0.13, instead of 0.15

The following parameters are valid for DVB-C Annex A/C:

- *DTV_API_VERSION*
- *DTV_DELIVERY_SYSTEM*
- *DTV_TUNE*
- *DTV_CLEAR*
- *DTV_FREQUENCY*
- *DTV_MODULATION*
- *DTV_INVERSION*

- *DTV_SYMBOL_RATE*
- *DTV_INNER_FEC*
- *DTV_LNA*

In addition, the *DTV QoS statistics* are also valid.

DVB-C Annex B delivery system

The DVB-C Annex-B is only used on a few Countries like the United States.

The following parameters are valid for DVB-C Annex B:

- *DTV_API_VERSION*
- *DTV_DELIVERY_SYSTEM*
- *DTV_TUNE*
- *DTV_CLEAR*
- *DTV_FREQUENCY*
- *DTV_MODULATION*
- *DTV_INVERSION*
- *DTV_LNA*

In addition, the *DTV QoS statistics* are also valid.

Properties used on satellite delivery systems

DVB-S delivery system

The following parameters are valid for DVB-S:

- *DTV_API_VERSION*
- *DTV_DELIVERY_SYSTEM*
- *DTV_TUNE*
- *DTV_CLEAR*
- *DTV_FREQUENCY*
- *DTV_INVERSION*
- *DTV_SYMBOL_RATE*
- *DTV_INNER_FEC*
- *DTV_VOLTAGE*
- *DTV_TONE*

In addition, the *DTV QoS statistics* are also valid.

Future implementations might add those two missing parameters:

- *DTV_DISEQC_MASTER*
- *DTV_DISEQC_SLAVE_REPLY*

DVB-S2 delivery system

In addition to all parameters valid for DVB-S, DVB-S2 supports the following parameters:

- *DTV_MODULATION*
- *DTV_PILOT*
- *DTV_ROLLOFF*
- *DTV_STREAM_ID*
- *DTV_SCRAMBLING_SEQUENCE_INDEX*

In addition, the *DTV QoS statistics* are also valid.

Turbo code delivery system

In addition to all parameters valid for DVB-S, turbo code supports the following parameters:

- *DTV_MODULATION*

ISDB-S delivery system

The following parameters are valid for ISDB-S:

- *DTV_API_VERSION*
- *DTV_DELIVERY_SYSTEM*
- *DTV_TUNE*
- *DTV_CLEAR*
- *DTV_FREQUENCY*
- *DTV_INVERSION*
- *DTV_SYMBOL_RATE*
- *DTV_INNER_FEC*
- *DTV_VOLTAGE*
- *DTV_STREAM_ID*

Frontend uAPI data types

enum **fe_caps**

Frontend capabilities

Constants

FE_IS_STUPID
There' s something wrong at the frontend, and it can' t report its capabilities.

FE_CAN_INVERSION_AUTO
Can auto-detect frequency spectral band inversion

FE_CAN_FEC_1_2
Supports FEC 1/2

FE_CAN_FEC_2_3
Supports FEC 2/3

FE_CAN_FEC_3_4
Supports FEC 3/4

FE_CAN_FEC_4_5
Supports FEC 4/5

FE_CAN_FEC_5_6
Supports FEC 5/6

FE_CAN_FEC_6_7
Supports FEC 6/7

FE_CAN_FEC_7_8
Supports FEC 7/8

FE_CAN_FEC_8_9
Supports FEC 8/9

FE_CAN_FEC_AUTO
Can auto-detect FEC

FE_CAN_QPSK
Supports QPSK modulation

FE_CAN_QAM_16
Supports 16-QAM modulation

FE_CAN_QAM_32
Supports 32-QAM modulation

FE_CAN_QAM_64
Supports 64-QAM modulation

FE_CAN_QAM_128
Supports 128-QAM modulation

FE_CAN_QAM_256
Supports 256-QAM modulation

FE_CAN_QAM_AUTO
Can auto-detect QAM modulation

FE_CAN_TRANSMISSION_MODE_AUTO
Can auto-detect transmission mode

FE_CAN_BANDWIDTH_AUTO
Can auto-detect bandwidth

FE_CAN_GUARD_INTERVAL_AUTO
Can auto-detect guard interval

FE_CAN_HIERARCHY_AUTO
Can auto-detect hierarchy

FE_CAN_8VSB
Supports 8-VSB modulation

FE_CAN_16VSB
Supporta 16-VSB modulation

FE_HAS_EXTENDED_CAPS
Unused

FE_CAN_MULTISTREAM
Supports multistream filtering

FE_CAN_TURBO_FEC
Supports "turbo FEC" modulation

FE_CAN_2G_MODULATION
Supports "2nd generation" modulation, e. g. DVB-S2, DVB-T2, DVB-C2

FE_NEEDS_BENDING
Unused

FE_CAN_RECOVER
Can recover from a cable unplug automatically

FE_CAN_MUTE_TS
Can stop spurious TS data output

struct **dvb_frontend_info**

Frontend properties and capabilities

Definition

```
struct dvb_frontend_info {
  char name[128];
  enum fe_type type;
  __u32 frequency_min;
  __u32 frequency_max;
  __u32 frequency_stepsize;
  __u32 frequency_tolerance;
  __u32 symbol_rate_min;
  __u32 symbol_rate_max;
  __u32 symbol_rate_tolerance;
  __u32 notifier_delay;
  enum fe_caps caps;
};
```

Members

name
Name of the frontend

type
DEPRECATED. Should not be used on modern programs, as a frontend may have more than one type. In order to get the support types of a given frontend, use DTV_ENUM_DELSYS instead.

frequency_min
Minimal frequency supported by the frontend.

frequency_max
Minimal frequency supported by the frontend.

frequency_stepsize
All frequencies are multiple of this value.

frequency_tolerance
Frequency tolerance.

symbol_rate_min
Minimal symbol rate, in bauds (for Cable/Satellite systems).

symbol_rate_max
Maximal symbol rate, in bauds (for Cable/Satellite systems).

symbol_rate_tolerance
Maximal symbol rate tolerance, in ppm (for Cable/Satellite systems).

notifier_delay
DEPRECATED. Not used by any driver.

caps
Capabilities supported by the frontend, as specified in *enum fe_caps*.

Description

struct **dvb_diseqc_master_cmd**

DiSEqC master command

Definition

```
struct dvb_diseqc_master_cmd {
  __u8 msg[6];
  __u8 msg_len;
};
```

Members

msg
DiSEqC message to be sent. It contains a 3 bytes header with: framing + address + command, and an optional argument of up to 3 bytes of data.

msg_len
Length of the DiSEqC message. Valid values are 3 to 6.

Description

Check out the DiSEqC bus spec available on http://www.eutelsat.org/ for the possible messages that can be used.

struct **dvb_diseqc_slave_reply**

DiSEqC received data

Definition

```
struct dvb_diseqc_slave_reply {
  __u8 msg[4];
  __u8 msg_len;
  int timeout;
};
```

Members

msg
: DiSEqC message buffer to store a message received via DiSEqC. It contains one byte header with: framing and an optional argument of up to 3 bytes of data.

msg_len
: Length of the DiSEqC message. Valid values are 0 to 4, where 0 means no message.

timeout
: Return from ioctl after timeout ms with errorcode when no message was received.

Description

Check out the DiSEqC bus spec available on http://www.eutelsat.org/ for the possible messages that can be used.

enum **fe_sec_voltage**

DC Voltage used to feed the LNBf

Constants

SEC_VOLTAGE_13
: Output 13V to the LNBf

SEC_VOLTAGE_18
: Output 18V to the LNBf

SEC_VOLTAGE_OFF
: Don't feed the LNBf with a DC voltage

enum **fe_sec_tone_mode**

Type of tone to be send to the LNBf.

Constants

SEC_TONE_ON
: Sends a 22kHz tone burst to the antenna.

SEC_TONE_OFF
: Don't send a 22kHz tone to the antenna (except if the `FE_DISEQC_*` ioctls are called).

enum **fe_sec_mini_cmd**

Type of mini burst to be sent

Constants

SEC_MINI_A
Sends a mini-DiSEqC 22kHz '0' Tone Burst to select satellite-A

SEC_MINI_B
Sends a mini-DiSEqC 22kHz '1' Data Burst to select satellite-B

enum **fe_status**

Enumerates the possible frontend status.

Constants

FE_NONE
The frontend doesn't have any kind of lock. That's the initial frontend status

FE_HAS_SIGNAL
Has found something above the noise level.

FE_HAS_CARRIER
Has found a signal.

FE_HAS_VITERBI
FEC inner coding (Viterbi, LDPC or other inner code). is stable.

FE_HAS_SYNC
Synchronization bytes was found.

FE_HAS_LOCK
Digital TV were locked and everything is working.

FE_TIMEDOUT
Fo lock within the last about 2 seconds.

FE_REINIT
Frontend was reinitialized, application is recommended to reset DiSEqC, tone and parameters.

enum **fe_spectral_inversion**

Type of inversion band

Constants

INVERSION_OFF
Don't do spectral band inversion.

INVERSION_ON
Do spectral band inversion.

INVERSION_AUTO
Autodetect spectral band inversion.

Description

This parameter indicates if spectral inversion should be presumed or not. In the automatic setting (`INVERSION_AUTO`) the hardware will try to figure out the correct setting by itself. If the hardware doesn't support, the `dvb_frontend` will try to lock at the carrier first with inversion off. If it fails, it will try to enable inversion.

enum **fe_code_rate**

Type of Forward Error Correction (FEC)

Constants

FEC_NONE
No Forward Error Correction Code

FEC_1_2
Forward Error Correction Code 1/2

FEC_2_3
Forward Error Correction Code 2/3

FEC_3_4
Forward Error Correction Code 3/4

FEC_4_5
Forward Error Correction Code 4/5

FEC_5_6
Forward Error Correction Code 5/6

FEC_6_7
Forward Error Correction Code 6/7

FEC_7_8
Forward Error Correction Code 7/8

FEC_8_9
Forward Error Correction Code 8/9

FEC_AUTO
Autodetect Error Correction Code

FEC_3_5
Forward Error Correction Code 3/5

FEC_9_10
Forward Error Correction Code 9/10

FEC_2_5
Forward Error Correction Code 2/5

Description

Please note that not all FEC types are supported by a given standard.

enum **fe_modulation**

Type of modulation/constellation

Constants

QPSK
QPSK modulation

QAM_16
16-QAM modulation

QAM_32
32-QAM modulation

QAM_64
64-QAM modulation

QAM_128
128-QAM modulation

QAM_256
256-QAM modulation

QAM_AUTO
Autodetect QAM modulation

VSB_8
8-VSB modulation

VSB_16
16-VSB modulation

PSK_8
8-PSK modulation

APSK_16
16-APSK modulation

APSK_32
32-APSK modulation

DQPSK
DQPSK modulation

QAM_4_NR
4-QAM-NR modulation

Description

Please note that not all modulations are supported by a given standard.

enum **fe_transmit_mode**
Transmission mode

Constants

TRANSMISSION_MODE_2K
Transmission mode 2K

TRANSMISSION_MODE_8K
Transmission mode 8K

TRANSMISSION_MODE_AUTO
Autodetect transmission mode. The hardware will try to find the correct FFT-size (if capable) to fill in the missing parameters.

TRANSMISSION_MODE_4K
Transmission mode 4K

TRANSMISSION_MODE_1K
Transmission mode 1K

TRANSMISSION_MODE_16K
Transmission mode 16K

TRANSMISSION_MODE_32K
Transmission mode 32K

TRANSMISSION_MODE_C1
Single Carrier (C=1) transmission mode (DTMB only)

TRANSMISSION_MODE_C3780
Multi Carrier (C=3780) transmission mode (DTMB only)

Description

Please note that not all transmission modes are supported by a given standard.

enum **fe_guard_interval**
Guard interval

Constants

GUARD_INTERVAL_1_32
Guard interval 1/32

GUARD_INTERVAL_1_16
Guard interval 1/16

GUARD_INTERVAL_1_8
Guard interval 1/8

GUARD_INTERVAL_1_4
Guard interval 1/4

GUARD_INTERVAL_AUTO
Autodetect the guard interval

GUARD_INTERVAL_1_128
Guard interval 1/128

GUARD_INTERVAL_19_128
Guard interval 19/128

GUARD_INTERVAL_19_256
Guard interval 19/256

GUARD_INTERVAL_PN420
PN length 420 (1/4)

GUARD_INTERVAL_PN595
PN length 595 (1/6)

GUARD_INTERVAL_PN945
PN length 945 (1/9)

Description

Please note that not all guard intervals are supported by a given standard.

enum **fe_hierarchy**
Hierarchy

Constants

HIERARCHY_NONE
No hierarchy

HIERARCHY_1
Hierarchy 1

HIERARCHY_2
Hierarchy 2

HIERARCHY_4
Hierarchy 4

HIERARCHY_AUTO
Autodetect hierarchy (if supported)

Description

Please note that not all hierarchy types are supported by a given standard.

enum **fe_interleaving**
Interleaving

Constants

INTERLEAVING_NONE
No interleaving.

INTERLEAVING_AUTO
Auto-detect interleaving.

INTERLEAVING_240
Interleaving of 240 symbols.

INTERLEAVING_720
Interleaving of 720 symbols.

Description

Please note that, currently, only DTMB uses it.

enum **fe_pilot**
Type of pilot tone

Constants

PILOT_ON
Pilot tones enabled

PILOT_OFF
Pilot tones disabled

PILOT_AUTO
Autodetect pilot tones

enum **fe_rolloff**
Rolloff factor

Constants

ROLLOFF_35
Roloff factor: α=35%

ROLLOFF_20
Roloff factor: α=20%

ROLLOFF_25
Roloff factor: α=25%

ROLLOFF_AUTO
Auto-detect the roloff factor.

Description

enum **fe_delivery_system**
Type of the delivery system

Constants

SYS_UNDEFINED
Undefined standard. Generally, indicates an error

SYS_DVBC_ANNEX_A
Cable TV: DVB-C following ITU-T J.83 Annex A spec

SYS_DVBC_ANNEX_B
Cable TV: DVB-C following ITU-T J.83 Annex B spec (ClearQAM)

SYS_DVBT
Terrestrial TV: DVB-T

SYS_DSS
Satellite TV: DSS (not fully supported)

SYS_DVBS
Satellite TV: DVB-S

SYS_DVBS2
Satellite TV: DVB-S2

SYS_DVBH
Terrestrial TV (mobile): DVB-H (standard deprecated)

SYS_ISDBT
Terrestrial TV: ISDB-T

SYS_ISDBS
Satellite TV: ISDB-S

SYS_ISDBC
Cable TV: ISDB-C (no drivers yet)

SYS_ATSC
Terrestrial TV: ATSC

SYS_ATSCMH
Terrestrial TV (mobile): ATSC-M/H

SYS_DTMB
Terrestrial TV: DTMB

SYS_CMMB
Terrestrial TV (mobile): CMMB (not fully supported)

SYS_DAB
Digital audio: DAB (not fully supported)

SYS_DVBT2
Terrestrial TV: DVB-T2

SYS_TURBO
Satellite TV: DVB-S Turbo

SYS_DVBC_ANNEX_C
Cable TV: DVB-C following ITU-T J.83 Annex C spec

enum **atscmh_sccc_block_mode**

Type of Series Concatenated Convolutional Code Block Mode.

Constants

ATSCMH_SCCC_BLK_SEP
Separate SCCC: the SCCC outer code mode shall be set independently for each Group Region (A, B, C, D)

ATSCMH_SCCC_BLK_COMB
Combined SCCC: all four Regions shall have the same SCCC outer code mode.

ATSCMH_SCCC_BLK_RES
Reserved. Shouldn't be used.

enum **atscmh_sccc_code_mode**

Type of Series Concatenated Convolutional Code Rate.

Constants

ATSCMH_SCCC_CODE_HLF
The outer code rate of a SCCC Block is 1/2 rate.

ATSCMH_SCCC_CODE_QTR
The outer code rate of a SCCC Block is 1/4 rate.

ATSCMH_SCCC_CODE_RES
Reserved. Should not be used.

enum **atscmh_rs_frame_ensemble**

Reed Solomon(RS) frame ensemble.

Constants

ATSCMH_RSFRAME_ENS_PRI
Primary Ensemble.

ATSCMH_RSFRAME_ENS_SEC
Secondary Ensemble.

enum **atscmh_rs_frame_mode**

Reed Solomon (RS) frame mode.

Constants

ATSCMH_RSFRAME_PRI_ONLY
Single Frame: There is only a primary RS Frame for all Group Regions.

ATSCMH_RSFRAME_PRI_SEC
Dual Frame: There are two separate RS Frames: Primary RS Frame for Group Region A and B and Secondary RS Frame for Group Region C and D.

ATSCMH_RSFRAME_RES
: Reserved. Shouldn't be used.

enum **atscmh_rs_code_mode**

Constants

ATSCMH_RSCODE_211_187
: Reed Solomon code (211,187).

ATSCMH_RSCODE_223_187
: Reed Solomon code (223,187).

ATSCMH_RSCODE_235_187
: Reed Solomon code (235,187).

ATSCMH_RSCODE_RES
: Reserved. Shouldn't be used.

enum **fecap_scale_params**

scale types for the quality parameters.

Constants

FE_SCALE_NOT_AVAILABLE
: That QoS measure is not available. That could indicate a temporary or a permanent condition.

FE_SCALE_DECIBEL
: The scale is measured in 0.001 dB steps, typically used on signal measures.

FE_SCALE_RELATIVE
: The scale is a relative percentual measure, ranging from 0 (0%) to 0xffff (100%).

FE_SCALE_COUNTER
: The scale counts the occurrence of an event, like bit error, block error, lapsed time.

struct **dtv_stats**

Used for reading a DTV status property

Definition

```
struct dtv_stats {
  __u8 scale;
  union {
    __u64 uvalue;
    __s64 svalue;
  };
};
```

Members

scale
: Filled with *enum fecap_scale_params* - the scale in usage for that parameter

{unnamed_union}
: anonymous

uvalue
unsigned integer value of the measure, used when **scale** is either `FE_SCALE_RELATIVE` or `FE_SCALE_COUNTER`.

svalue
integer value of the measure, for `FE_SCALE_DECIBEL`, used for dB measures. The unit is 0.001 dB.

Description

For most delivery systems, this will return a single value for each parameter.

It should be noticed, however, that new OFDM delivery systems like ISDB can use different modulation types for each group of carriers. On such standards, up to 8 groups of statistics can be provided, one for each carrier group (called "layer" on ISDB).

In order to be consistent with other delivery systems, the first value refers to the entire set of carriers ("global").

scale should use the value `FE_SCALE_NOT_AVAILABLE` when the value for the entire group of carriers or from one specific layer is not provided by the hardware.

len should be filled with the latest filled status + 1.

In other words, for ISDB, those values should be filled like:

```
u.st.stat.svalue[0] = global statistics;
u.st.stat.scale[0] = FE_SCALE_DECIBEL;
u.st.stat.value[1] = layer A statistics;
u.st.stat.scale[1] = FE_SCALE_NOT_AVAILABLE (if not available);
u.st.stat.svalue[2] = layer B statistics;
u.st.stat.scale[2] = FE_SCALE_DECIBEL;
u.st.stat.svalue[3] = layer C statistics;
u.st.stat.scale[3] = FE_SCALE_DECIBEL;
u.st.len = 4;
```

struct **dtv_fe_stats**
store Digital TV frontend statistics

Definition

```
struct dtv_fe_stats {
  __u8 len;
  struct dtv_stats stat[MAX_DTV_STATS];
};
```

Members

len
length of the statistics - if zero, stats is disabled.

stat
array with digital TV statistics.

Description

On most standards, **len** can either be 0 or 1. However, for ISDB, each layer is modulated in separate. So, each layer may have its own set of statistics. If so, stat[0] carries on a global value for the property. Indexes 1 to 3 means layer A to B.

struct **dtv_property**

store one of frontend command and its value

Definition

```
struct dtv_property {
  __u32 cmd;
  __u32 reserved[3];
  union {
    __u32 data;
    struct dtv_fe_stats st;
    struct {
      __u8 data[32];
      __u32 len;
      __u32 reserved1[3];
      void *reserved2;
    } buffer;
  } u;
  int result;
};
```

Members

cmd
Digital TV command.

reserved
Not used.

u
Union with the values for the command.

u.data
A unsigned 32 bits integer with command value.

u.st
a *struct dtv_fe_stats* array of statistics.

u.buffer
Struct to store bigger properties. Currently unused.

u.buffer.data
an unsigned 32-bits array.

u.buffer.len
number of elements of the buffer.

u.buffer.reserved1
Reserved.

u.buffer.reserved2
Reserved.

result
Currently unused.

struct **dtv_properties**
a set of command/value pairs.

Definition

```
struct dtv_properties {
  __u32 num;
  struct dtv_property *props;
};
```

Members

num
amount of commands stored at the struct.

props
a pointer to *struct dtv_property*.

Frontend Function Calls

Digital TV frontend open()

Name

fe-open - Open a frontend device

Synopsis

```
#include <fcntl.h>
```

int **open**(const char *device_name, int flags)

Arguments

device_name
Device to be opened.

flags
Open flags. Access can either be `O_RDWR` or `O_RDONLY`.

Multiple opens are allowed with `O_RDONLY`. In this mode, only query and read ioctls are allowed.

Only one open is allowed in `O_RDWR`. In this mode, all ioctls are allowed.

When the `O_NONBLOCK` flag is given, the system calls may return `EAGAIN` error code when no data is available or when the device driver is temporarily busy.

Other flags have no effect.

Description

This system call opens a named frontend device (`/dev/dvb/adapter?/frontend?`) for subsequent use. Usually the first thing to do after a successful open is to find out the frontend type with *ioctl FE_GET_INFO*.

The device can be opened in read-only mode, which only allows monitoring of device status and statistics, or read/write mode, which allows any kind of use (e.g. performing tuning operations.)

In a system with multiple front-ends, it is usually the case that multiple devices cannot be open in read/write mode simultaneously. As long as a front-end device is opened in read/write mode, other open() calls in read/write mode will either fail or block, depending on whether non-blocking or blocking mode was specified. A front-end device opened in blocking mode can later be put into non-blocking mode (and vice versa) using the F_SETFL command of the fcntl system call. This is a standard system call, documented in the Linux manual page for fcntl. When an open() call has succeeded, the device will be ready for use in the specified mode. This implies that the corresponding hardware is powered up, and that other front-ends may have been powered down to make that possible.

Return Value

On success `open()` returns the new file descriptor. On error, -1 is returned, and the `errno` variable is set appropriately.

Possible error codes are:

On success 0 is returned, and `ca_slot_info` is filled.

On error -1 is returned, and the `errno` variable is set appropriately.

EPERM	The caller has no permission to access the device.
EBUSY	The the device driver is already in use.
EMFILE	The process already has the maximum number of files open.
ENFILE	The limit on the total number of files open on the system has been reached.

The generic error codes are described at the *Generic Error Codes* chapter.

Digital TV frontend close()

Name

fe-close - Close a frontend device

Synopsis

```
#include <unistd.h>
```

int **close**(int fd)

Arguments

fd
File descriptor returned by *open()*.

Description

This system call closes a previously opened front-end device. After closing a front-end device, its corresponding hardware might be powered down automatically.

Return Value

On success 0 is returned.

On error -1 is returned, and the `errno` variable is set appropriately.

Generic error codes are described at the *Generic Error Codes* chapter.

ioctl FE_GET_INFO

Name

FE_GET_INFO - Query Digital TV frontend capabilities and returns information about the - front-end. This call only requires read-only access to the device.

Synopsis

FE_GET_INFO

`int ioctl(int fd, FE_GET_INFO, struct dvb_frontend_info *argp)`

Arguments

fd
File descriptor returned by *open()*.

argp
pointer to struct *dvb_frontend_info*

Description

All Digital TV frontend devices support the *ioctl FE_GET_INFO* ioctl. It is used to identify kernel devices compatible with this specification and to obtain information about driver and hardware capabilities. The ioctl takes a pointer to dvb_frontend_info which is filled by the driver. When the driver is not compatible with this specification the ioctl returns an error.

frontend capabilities

Capabilities describe what a frontend can do. Some capabilities are supported only on some specific frontend types.

The frontend capabilities are described at *fe_caps*.

Return Value

On success 0 is returned.

On error -1 is returned, and the `errno` variable is set appropriately.

Generic error codes are described at the *Generic Error Codes* chapter.

ioctl FE_READ_STATUS

Name

FE_READ_STATUS - Returns status information about the front-end. This call only requires - read-only access to the device

Synopsis

FE_READ_STATUS

`int ioctl(int fd, FE_READ_STATUS, unsigned int *status)`

Arguments

fd
: File descriptor returned by *open()*.

status
: pointer to a bitmask integer filled with the values defined by enum *fe_status*.

Description

All Digital TV frontend devices support the `FE_READ_STATUS` ioctl. It is used to check about the locking status of the frontend after being tuned. The ioctl takes a pointer to an integer where the status will be written.

Note: The size of status is actually sizeof(*enum fe_status*), with varies according with the architecture. This needs to be fixed in the future.

int fe_status

The fe_status parameter is used to indicate the current state and/or state changes of the frontend hardware. It is produced using the enum *fe_status* values on a bitmask

Return Value

On success 0 is returned.

On error -1 is returned, and the `errno` variable is set appropriately.

Generic error codes are described at the *Generic Error Codes* chapter.

ioctl FE_SET_PROPERTY, FE_GET_PROPERTY

Name

FE_SET_PROPERTY - FE_GET_PROPERTY - FE_SET_PROPERTY sets one or more frontend properties. - FE_GET_PROPERTY returns one or more frontend properties.

Synopsis

FE_GET_PROPERTY

`int ioctl(int fd, FE_GET_PROPERTY, struct dtv_properties *argp)`

FE_SET_PROPERTY

`int ioctl(int fd, FE_SET_PROPERTY, struct dtv_properties *argp)`

Arguments

fd
File descriptor returned by *open()*.

argp
Pointer to struct *dtv_properties*.

Description

All Digital TV frontend devices support the FE_SET_PROPERTY and FE_GET_PROPERTY ioctls. The supported properties and statistics depends on the delivery system and on the device:

- FE_SET_PROPERTY:
  - This ioctl is used to set one or more frontend properties.
  - This is the basic command to request the frontend to tune into some frequency and to start decoding the digital TV signal.
  - This call requires read/write access to the device.

Note: At return, the values aren't updated to reflect the actual parameters used. If the actual parameters are needed, an explicit call to FE_GET_PROPERTY is needed.

- FE_GET_PROPERTY:
  - This ioctl is used to get properties and statistics from the frontend.
  - No properties are changed, and statistics aren't reset.
  - This call only requires read-only access to the device.

Return Value

On success 0 is returned.

On error -1 is returned, and the errno variable is set appropriately.

Generic error codes are described at the *Generic Error Codes* chapter.

ioctl FE_DISEQC_RESET_OVERLOAD

Name

FE_DISEQC_RESET_OVERLOAD - Restores the power to the antenna subsystem, if it was powered off due - to power overload.

Synopsis

FE_DISEQC_RESET_OVERLOAD

`int ioctl(int fd, FE_DISEQC_RESET_OVERLOAD, NULL)`

Arguments

fd
File descriptor returned by *open()*.

Description

If the bus has been automatically powered off due to power overload, this ioctl call restores the power to the bus. The call requires read/write access to the device. This call has no effect if the device is manually powered off. Not all Digital TV adapters support this ioctl.

Return Value

On success 0 is returned.

On error -1 is returned, and the `errno` variable is set appropriately.

Generic error codes are described at the *Generic Error Codes* chapter.

ioctl FE_DISEQC_SEND_MASTER_CMD

Name

FE_DISEQC_SEND_MASTER_CMD - Sends a DiSEqC command

Synopsis

FE_DISEQC_SEND_MASTER_CMD

```
int ioctl(int fd, FE_DISEQC_SEND_MASTER_CMD, struct
dvb_diseqc_master_cmd *argp)
```

Arguments

fd
File descriptor returned by *open()*.

argp
pointer to struct *dvb_diseqc_master_cmd*

Description

Sends the DiSEqC command pointed by *dvb_diseqc_master_cmd* to the antenna subsystem.

Return Value

On success 0 is returned.

On error -1 is returned, and the `errno` variable is set appropriately.

Generic error codes are described at the *Generic Error Codes* chapter.

ioctl FE_DISEQC_RECV_SLAVE_REPLY

Name

FE_DISEQC_RECV_SLAVE_REPLY - Receives reply from a DiSEqC 2.0 command

Synopsis

FE_DISEQC_RECV_SLAVE_REPLY

```
int ioctl(int fd, FE_DISEQC_RECV_SLAVE_REPLY, struct
dvb_diseqc_slave_reply *argp)
```

Arguments

fd
File descriptor returned by *open()*.

argp
pointer to struct *dvb_diseqc_slave_reply*.

Description

Receives reply from a DiSEqC 2.0 command.

The received message is stored at the buffer pointed by `argp`.

Return Value

On success 0 is returned.

On error -1 is returned, and the `errno` variable is set appropriately.

Generic error codes are described at the *Generic Error Codes* chapter.

ioctl FE_DISEQC_SEND_BURST

Name

FE_DISEQC_SEND_BURST - Sends a 22KHz tone burst for 2x1 mini DiSEqC satellite selection.

Synopsis

FE_DISEQC_SEND_BURST

`int ioctl(int fd, FE_DISEQC_SEND_BURST, enum fe_sec_mini_cmd tone)`

Arguments

fd
File descriptor returned by *open()*.

tone
An integer enumered value described at *fe_sec_mini_cmd*.

Description

This ioctl is used to set the generation of a 22kHz tone burst for mini DiSEqC satellite selection for 2x1 switches. This call requires read/write permissions.

It provides support for what's specified at Digital Satellite Equipment Control (DiSEqC) - Simple "ToneBurst" Detection Circuit specification.

Return Value

On success 0 is returned.

On error -1 is returned, and the `errno` variable is set appropriately.

Generic error codes are described at the *Generic Error Codes* chapter.

ioctl FE_SET_TONE

Name

FE_SET_TONE - Sets/resets the generation of the continuous 22kHz tone.

Synopsis

FE_SET_TONE

`int ioctl(int fd, FE_SET_TONE, enum fe_sec_tone_mode tone)`

Arguments

fd
: File descriptor returned by *open()*.

tone
: an integer enumered value described at *fe_sec_tone_mode*

Description

This ioctl is used to set the generation of the continuous 22kHz tone. This call requires read/write permissions.

Usually, satellite antenna subsystems require that the digital TV device to send a 22kHz tone in order to select between high/low band on some dual-band LNBf. It is also used to send signals to DiSEqC equipment, but this is done using the DiSEqC ioctls.

> **Attention:** If more than one device is connected to the same antenna, setting a tone may interfere on other devices, as they may lose the capability of selecting the band. So, it is recommended that applications would change to SEC_TONE_OFF when the device is not used.

Return Value

On success 0 is returned.

On error -1 is returned, and the `errno` variable is set appropriately.

Generic error codes are described at the *Generic Error Codes* chapter.

ioctl FE_SET_VOLTAGE

Name

FE_SET_VOLTAGE - Allow setting the DC level sent to the antenna subsystem.

Synopsis

FE_SET_VOLTAGE

`int ioctl(int fd, FE_SET_VOLTAGE, enum fe_sec_voltage voltage)`

Arguments

fd
File descriptor returned by *open()*.

voltage
an integer enumered value described at *fe_sec_voltage*

Description

This ioctl allows to set the DC voltage level sent through the antenna cable to 13V, 18V or off.

Usually, a satellite antenna subsystems require that the digital TV device to send a DC voltage to feed power to the LNBf. Depending on the LNBf type, the polarization or the intermediate frequency (IF) of the LNBf can controlled by the voltage level. Other devices (for example, the ones that implement DISEqC and multipoint LNBf's don't need to control the voltage level, provided that either 13V or 18V is sent to power up the LNBf.

Attention: if more than one device is connected to the same antenna, setting a voltage level may interfere on other devices, as they may lose the capability of setting polarization or IF. So, on those cases, setting the voltage to SEC_VOLTAGE_OFF while the device is not is used is recommended.

Return Value

On success 0 is returned.

On error -1 is returned, and the `errno` variable is set appropriately.

Generic error codes are described at the *Generic Error Codes* chapter.

ioctl FE_ENABLE_HIGH_LNB_VOLTAGE

Name

FE_ENABLE_HIGH_LNB_VOLTAGE - Select output DC level between normal LNBf voltages or higher LNBf - voltages.

Synopsis

FE_ENABLE_HIGH_LNB_VOLTAGE

`int ioctl(int fd, FE_ENABLE_HIGH_LNB_VOLTAGE, unsigned int high)`

Arguments

fd
: File descriptor returned by *open()*.

high
: Valid flags:

  - 0 - normal 13V and 18V.
  - >0 - enables slightly higher voltages instead of 13/18V, in order to compensate for long antenna cables.

Description

Select output DC level between normal LNBf voltages or higher LNBf voltages between 0 (normal) or a value grater than 0 for higher voltages.

Return Value

On success 0 is returned.

On error -1 is returned, and the `errno` variable is set appropriately.

Generic error codes are described at the *Generic Error Codes* chapter.

ioctl FE_SET_FRONTEND_TUNE_MODE

Name

FE_SET_FRONTEND_TUNE_MODE - Allow setting tuner mode flags to the frontend.

Synopsis

FE_SET_FRONTEND_TUNE_MODE

`int ioctl(int fd, FE_SET_FRONTEND_TUNE_MODE, unsigned int flags)`

Arguments

fd
File descriptor returned by *open()*.

flags
Valid flags:

- 0 - normal tune mode
- `FE_TUNE_MODE_ONESHOT` - When set, this flag will disable any zigzagging or other "normal" tuning behaviour. Additionally, there will be no automatic monitoring of the lock status, and hence no frontend events will be generated. If a frontend device is closed, this flag will be automatically turned off when the device is reopened read-write.

Description

Allow setting tuner mode flags to the frontend, between 0 (normal) or `FE_TUNE_MODE_ONESHOT` mode

Return Value

On success 0 is returned.

On error -1 is returned, and the `errno` variable is set appropriately.

Generic error codes are described at the *Generic Error Codes* chapter.

8.3.3 Digital TV Demux Device

The Digital TV demux device controls the MPEG-TS filters for the digital TV. If the driver and hardware supports, those filters are implemented at the hardware. Otherwise, the Kernel provides a software emulation.

It can be accessed through `/dev/adapter?/demux?`. Data types and and ioctl definitions can be accessed by including `linux/dvb/dmx.h` in your application.

Demux Data Types

enum **dmx_output**

Output for the demux.

Constants

DMX_OUT_DECODER
Streaming directly to decoder.

DMX_OUT_TAP
Output going to a memory buffer (to be retrieved via the read command). Delivers the stream output to the demux device on which the ioctl is called.

DMX_OUT_TS_TAP
Output multiplexed into a new TS (to be retrieved by reading from the logical DVR device). Routes output to the logical DVR device `/dev/dvb/adapter?/dvr?`, which delivers a TS multiplexed from all filters for which **DMX_OUT_TS_TAP** was specified.

DMX_OUT_TSDEMUX_TAP
Like **DMX_OUT_TS_TAP** but retrieved from the DMX device.

enum **dmx_input**

Input from the demux.

Constants

DMX_IN_FRONTEND
Input from a front-end device.

DMX_IN_DVR
Input from the logical DVR device.

enum **dmx_ts_pes**

type of the PES filter.

Constants

DMX_PES_AUDIO0
first audio PID. Also referred as **DMX_PES_AUDIO**.

DMX_PES_VIDEO0
first video PID. Also referred as **DMX_PES_VIDEO**.

DMX_PES_TELETEXT0
first teletext PID. Also referred as **DMX_PES_TELETEXT**.

DMX_PES_SUBTITLE0
first subtitle PID. Also referred as **DMX_PES_SUBTITLE**.

DMX_PES_PCR0
first Program Clock Reference PID. Also referred as **DMX_PES_PCR**.

DMX_PES_AUDIO1
second audio PID.

DMX_PES_VIDEO1
second video PID.

DMX_PES_TELETEXT1
second teletext PID.

DMX_PES_SUBTITLE1
second subtitle PID.

DMX_PES_PCR1
second Program Clock Reference PID.

DMX_PES_AUDIO2
third audio PID.

DMX_PES_VIDEO2
third video PID.

DMX_PES_TELETEXT2
third teletext PID.

DMX_PES_SUBTITLE2
third subtitle PID.

DMX_PES_PCR2
third Program Clock Reference PID.

DMX_PES_AUDIO3
fourth audio PID.

DMX_PES_VIDEO3
fourth video PID.

DMX_PES_TELETEXT3
fourth teletext PID.

DMX_PES_SUBTITLE3
fourth subtitle PID.

DMX_PES_PCR3
fourth Program Clock Reference PID.

DMX_PES_OTHER
any other PID.

struct **dmx_filter**

Specifies a section header filter.

Definition

```
struct dmx_filter {
  __u8 filter[DMX_FILTER_SIZE];
  __u8 mask[DMX_FILTER_SIZE];
  __u8 mode[DMX_FILTER_SIZE];
};
```

Members

filter
bit array with bits to be matched at the section header.

mask
bits that are valid at the filter bit array.

mode
mode of match: if bit is zero, it will match if equal (positive match); if bit is one, it will match if the bit is negated.

Note

All arrays in this struct have a size of DMX_FILTER_SIZE (16 bytes).

struct **dmx_sct_filter_params**
Specifies a section filter.

Definition

```
struct dmx_sct_filter_params {
  __u16 pid;
  struct dmx_filter filter;
  __u32 timeout;
  __u32 flags;
#define DMX_CHECK_CRC       1;
#define DMX_ONESHOT         2;
#define DMX_IMMEDIATE_START 4;
};
```

Members

pid
PID to be filtered.

filter
section header filter, as defined by *struct dmx_filter*.

timeout
maximum time to filter, in milliseconds.

flags
extra flags for the section filter.

Description

Carries the configuration for a MPEG-TS section filter.

The **flags** can be:

- DMX_CHECK_CRC - only deliver sections where the CRC check succeeded;

- `DMX_ONESHOT` - disable the section filter after one section has been delivered;
- `DMX_IMMEDIATE_START` - Start filter immediately without requiring a *DMX_START*.

struct **dmx_pes_filter_params**

Specifies Packetized Elementary Stream (PES) filter parameters.

Definition

```
struct dmx_pes_filter_params {
  __u16 pid;
  enum dmx_input  input;
  enum dmx_output output;
  enum dmx_ts_pes pes_type;
  __u32 flags;
};
```

Members

pid
PID to be filtered.

input
Demux input, as specified by *enum dmx_input*.

output
Demux output, as specified by *enum dmx_output*.

pes_type
Type of the pes filter, as specified by `enum dmx_pes_type`.

flags
Demux PES flags.

struct **dmx_stc**

Stores System Time Counter (STC) information.

Definition

```
struct dmx_stc {
  unsigned int num;
  unsigned int base;
  __u64 stc;
};
```

Members

num
input data: number of the STC, from 0 to N.

base
output: divisor for STC to get 90 kHz clock.

stc
output: stc in **base** * 90 kHz units.

enum **dmx_buffer_flags**

DMX memory-mapped buffer flags

Constants

DMX_BUFFER_FLAG_HAD_CRC32_DISCARD
Indicates that the Kernel discarded one or more frames due to wrong CRC32 checksum.

DMX_BUFFER_FLAG_TEI
Indicates that the Kernel has detected a Transport Error indicator (TEI) on a filtered pid.

DMX_BUFFER_PKT_COUNTER_MISMATCH
Indicates that the Kernel has detected a packet counter mismatch on a filtered pid.

DMX_BUFFER_FLAG_DISCONTINUITY_DETECTED
Indicates that the Kernel has detected one or more frame discontinuity.

DMX_BUFFER_FLAG_DISCONTINUITY_INDICATOR
Received at least one packet with a frame discontinuity indicator.

struct **dmx_buffer**

dmx buffer info

Definition

```
struct dmx_buffer {
  __u32 index;
  __u32 bytesused;
  __u32 offset;
  __u32 length;
  __u32 flags;
  __u32 count;
};
```

Members

index
id number of the buffer

bytesused
number of bytes occupied by data in the buffer (payload);

offset
for buffers with memory == DMX_MEMORY_MMAP; offset from the start of the device memory for this plane, (or a "cookie" that should be passed to mmap() as offset)

length
size in bytes of the buffer

flags
bit array of buffer flags as defined by *enum dmx_buffer_flags*. Filled only at `DMX_DQBUF`.

count
: monotonic counter for filled buffers. Helps to identify data stream loses. Filled only at `DMX_DQBUF`.

Description

Contains data exchanged by application and driver using one of the streaming I/O methods.

Please notice that, for `DMX_QBUF`, only **index** should be filled. On `DMX_DQBUF` calls, all fields will be filled by the Kernel.

struct **dmx_requestbuffers**
: request dmx buffer information

Definition

```
struct dmx_requestbuffers {
  __u32 count;
  __u32 size;
};
```

Members

count
: number of requested buffers,

size
: size in bytes of the requested buffer

Description

Contains data used for requesting a dmx buffer. All reserved fields must be set to zero.

struct **dmx_exportbuffer**
: export of dmx buffer as DMABUF file descriptor

Definition

```
struct dmx_exportbuffer {
  __u32 index;
  __u32 flags;
  __s32 fd;
};
```

Members

index
: id number of the buffer

flags
: flags for newly created file, currently only O_CLOEXEC is supported, refer to manual of open syscall for more details

fd
: file descriptor associated with DMABUF (set by driver)

Description

Contains data used for exporting a dmx buffer as DMABUF file descriptor. The buffer is identified by a 'cookie' returned by DMX_QUERYBUF (identical to the cookie used to mmap() the buffer to userspace). All reserved fields must be set to zero. The field reserved0 is expected to become a structure 'type' allowing an alternative layout of the structure content. Therefore this field should not be used for any other extensions.

Demux Function Calls

Digital TV demux open()

Name

Digital TV demux open()

Synopsis

int **open**(const char *deviceName, int flags)

Arguments

name
  Name of specific Digital TV demux device.

flags
  A bit-wise OR of the following flags:

`O_RDONLY`	read-only access
`O_RDWR`	read/write access
`O_NONBLOCK`	open in non-blocking mode (blocking mode is the default)

Description

This system call, used with a device name of `/dev/dvb/adapter?/demux?`, allocates a new filter and returns a handle which can be used for subsequent control of that filter. This call has to be made for each filter to be used, i.e. every returned file descriptor is a reference to a single filter. `/dev/dvb/adapter?/dvr?` is a logical device to be used for retrieving Transport Streams for digital video recording. When reading from this device a transport stream containing the packets from all PES filters set in the corresponding demux device (`/dev/dvb/adapter?/demux?`) having the output set to `DMX_OUT_TS_TAP`. A recorded Transport Stream is replayed by writing to this device.

The significance of blocking or non-blocking mode is described in the documentation for functions where there is a difference. It does not affect the semantics of the `open()` call itself. A device opened in blocking mode can later be put into non-blocking mode (and vice versa) using the `F_SETFL` command of the fcntl system call.

Return Value

On success 0 is returned.

On error -1 is returned, and the `errno` variable is set appropriately.

`EMFILE`	"Too many open files", i.e. no more filters available.

The generic error codes are described at the *Generic Error Codes* chapter.

Digital TV demux close()

Name

Digital TV demux close()

Synopsis

int **close**(int fd)

Arguments

fd
: File descriptor returned by a previous call to *open()*.

Description

This system call deactivates and deallocates a filter that was previously allocated via the *open()* call.

Return Value

On success 0 is returned.

On error, -1 is returned and the `errno` variable is set appropriately.

The generic error codes are described at the *Generic Error Codes* chapter.

Digital TV demux read()

Name

Digital TV demux read()

Synopsis

size_t **read**(int fd, void *buf, size_t count)

Arguments

fd

File descriptor returned by a previous call to *open()*.

buf
Buffer to be filled

count
Max number of bytes to read

Description

This system call returns filtered data, which might be section or Packetized Elementary Stream (PES) data. The filtered data is transferred from the driver's internal circular buffer to buf. The maximum amount of data to be transferred is implied by count.

Note: if a section filter created with *DMX_CHECK_CRC* flag set, data that fails on CRC check will be silently ignored.

Return Value

On success 0 is returned.

On error -1 is returned, and the errno variable is set appropriately.

EWOULDBLOCK	No data to return and O_NONBLOCK was specified.
EOVERFLOW	The filtered data was not read from the buffer in due time, resulting in non-read data being lost. The buffer is flushed.
ETIMEDOUT	The section was not loaded within the stated timeout period. See ioctl *DMX_SET_FILTER* for how to set a timeout.
EFAULT	The driver failed to write to the callers buffer due to an invalid *buf pointer.

The generic error codes are described at the *Generic Error Codes* chapter.

Digital TV demux write()

Name

Digital TV demux write()

Synopsis

ssize_t **write**(int fd, const void *buf, size_t count)

Arguments

fd
    File descriptor returned by a previous call to *open()*.

buf
    Buffer with data to be written

count
    Number of bytes at the buffer

Description

This system call is only provided by the logical device `/dev/dvb/adapter?/dvr?`, associated with the physical demux device that provides the actual DVR functionality. It is used for replay of a digitally recorded Transport Stream. Matching filters have to be defined in the corresponding physical demux device, `/dev/dvb/adapter?/demux?`. The amount of data to be transferred is implied by count.

Return Value

On success 0 is returned.

On error -1 is returned, and the `errno` variable is set appropriately.

EWOULDBLOCK	No data was written. This might happen if `O_NONBLOCK` was specified and there is no more buffer space available (if `O_NONBLOCK` is not specified the function will block until buffer space is available).
EBUSY	This error code indicates that there are conflicting requests. The corresponding demux device is setup to receive data from the front- end. Make sure that these filters are stopped and that the filters with input set to `DMX_IN_DVR` are started.

The generic error codes are described at the *Generic Error Codes* chapter.

Digital TV mmap()

Name

dmx-mmap - Map device memory into application address space

> **Warning:** this API is still experimental

Synopsis

```
#include <unistd.h>
#include <sys/mman.h>
```

void ***mmap**(void *start, size_t length, int prot, int flags, int fd, off_t offset)

Arguments

start
: Map the buffer to this address in the application' s address space. When the `MAP_FIXED` flag is specified, `start` must be a multiple of the pagesize and mmap will fail when the specified address cannot be used. Use of this option is discouraged; applications should just specify a `NULL` pointer here.

length
: Length of the memory area to map. This must be a multiple of the DVB packet length (188, on most drivers).

prot
: The `prot` argument describes the desired memory protection. Regardless of the device type and the direction of data exchange it should be set to `PROT_READ` | `PROT_WRITE`, permitting read and write access to image buffers. Drivers should support at least this combination of flags.

flags
: The `flags` parameter specifies the type of the mapped object, mapping options and whether modifications made to the mapped copy of the page are private to the process or are to be shared with other references.

  `MAP_FIXED` requests that the driver selects no other address than the one specified. If the specified address cannot be used, *mmap()* will fail. If `MAP_FIXED` is specified, `start` must be a multiple of the pagesize. Use of this option is discouraged.

  One of the `MAP_SHARED` or `MAP_PRIVATE` flags must be set. `MAP_SHARED` allows applications to share the mapped memory with other (e. g. child-) processes.

  > **Note:** The Linux Digital TV applications should not set the `MAP_PRIVATE`, `MAP_DENYWRITE`, `MAP_EXECUTABLE` or `MAP_ANON` flags.

fd
File descriptor returned by *open()*.

offset
Offset of the buffer in device memory, as returned by *ioctl DMX_QUERYBUF* ioctl.

Description

The *mmap()* function asks to map `length` bytes starting at `offset` in the memory of the device specified by `fd` into the application address space, preferably at address `start`. This latter address is a hint only, and is usually specified as 0.

Suitable length and offset parameters are queried with the *ioctl DMX_QUERYBUF* ioctl. Buffers must be allocated with the *ioctl DMX_REQBUFS* ioctl before they can be queried.

To unmap buffers the *munmap()* function is used.

Return Value

On success *mmap()* returns a pointer to the mapped buffer. On error `MAP_FAILED` (-1) is returned, and the `errno` variable is set appropriately. Possible error codes are:

EBADF
`fd` is not a valid file descriptor.

EACCES
`fd` is not open for reading and writing.

EINVAL
The `start` or `length` or `offset` are not suitable. (E. g. they are too large, or not aligned on a `PAGESIZE` boundary.)

The `flags` or `prot` value is not supported.

No buffers have been allocated with the *ioctl DMX_REQBUFS* ioctl.

ENOMEM
Not enough physical or virtual memory was available to complete the request.

DVB munmap()

Name

dmx-munmap - Unmap device memory

> **Warning:** This API is still experimental.

Synopsis

```
#include <unistd.h>
#include <sys/mman.h>
```

int **munmap**(void *start, size_t length)

Arguments

start
Address of the mapped buffer as returned by the *mmap()* function.

length
Length of the mapped buffer. This must be the same value as given to *mmap()*.

Description

Unmaps a previously with the *mmap()* function mapped buffer and frees it, if possible.

Return Value

On success *munmap()* returns 0, on failure -1 and the `errno` variable is set appropriately:

EINVAL
The `start` or `length` is incorrect, or no buffers have been mapped yet.

DMX_START

Name

DMX_START

Synopsis

DMX_START

`int ioctl(int fd, DMX_START)`

Arguments

fd
: File descriptor returned by *open()*.

Description

This ioctl call is used to start the actual filtering operation defined via the ioctl calls *DMX_SET_FILTER* or *DMX_SET_PES_FILTER*.

Return Value

On success 0 is returned.

On error -1 is returned, and the `errno` variable is set appropriately.

EINVAL	Invalid argument, i.e. no filtering parameters provided via the *DMX_SET_FILTER* or *DMX_SET_PES_FILTER* ioctls.
EBUSY	This error code indicates that there are conflicting requests. There are active filters filtering data from another input source. Make sure that these filters are stopped before starting this filter.

The generic error codes are described at the *Generic Error Codes* chapter.

DMX_STOP

Name

DMX_STOP

Synopsis

DMX_STOP

`int ioctl(int fd, DMX_STOP)`

Arguments

fd
: File descriptor returned by *open()*.

Description

This ioctl call is used to stop the actual filtering operation defined via the ioctl calls *DMX_SET_FILTER* or *DMX_SET_PES_FILTER* and started via the *DMX_START* command.

Return Value

On success 0 is returned.

On error -1 is returned, and the `errno` variable is set appropriately.

The generic error codes are described at the *Generic Error Codes* chapter.

DMX_SET_FILTER

Name

DMX_SET_FILTER

Synopsis

DMX_SET_FILTER

`int ioctl(int fd, DMX_SET_FILTER, struct dmx_sct_filter_params *params)`

Arguments

fd
: File descriptor returned by `open()`.

`params`
: Pointer to structure containing filter parameters.

Description

This ioctl call sets up a filter according to the filter and mask parameters provided. A timeout may be defined stating number of seconds to wait for a section to be loaded. A value of 0 means that no timeout should be applied. Finally there is a flag field where it is possible to state whether a section should be CRC-checked, whether the filter should be a ” one-shot” filter, i.e. if the filtering operation should be stopped after the first section is received, and whether the filtering operation should be started immediately (without waiting for a *DMX_START* ioctl call). If a filter was previously set-up, this filter will be canceled, and the receive buffer will be flushed.

Return Value

On success 0 is returned.

On error -1 is returned, and the `errno` variable is set appropriately.

The generic error codes are described at the *Generic Error Codes* chapter.

DMX_SET_PES_FILTER

Name

DMX_SET_PES_FILTER

Synopsis

DMX_SET_PES_FILTER

`int ioctl(int fd, DMX_SET_PES_FILTER, struct dmx_pes_filter_params *params)`

Arguments

fd
: File descriptor returned by *open()*.

params
: Pointer to structure containing filter parameters.

Description

This ioctl call sets up a PES filter according to the parameters provided. By a PES filter is meant a filter that is based just on the packet identifier (PID), i.e. no PES header or payload filtering capability is supported.

Return Value

On success 0 is returned.

On error -1 is returned, and the `errno` variable is set appropriately.

EBUSY	This error code indicates that there are conflicting requests. There are active filters filtering data from another input source. Make sure that these filters are stopped before starting this filter.

The generic error codes are described at the *Generic Error Codes* chapter.

DMX_SET_BUFFER_SIZE

Name

DMX_SET_BUFFER_SIZE

Synopsis

DMX_SET_BUFFER_SIZE

`int ioctl(int fd, DMX_SET_BUFFER_SIZE, unsigned long size)`

Arguments

fd
File descriptor returned by *open()*.

size
Unsigned long size

Description

This ioctl call is used to set the size of the circular buffer used for filtered data. The default size is two maximum sized sections, i.e. if this function is not called a buffer size of `2 * 4096` bytes will be used.

Return Value

On success 0 is returned.

On error -1 is returned, and the `errno` variable is set appropriately.

The generic error codes are described at the *Generic Error Codes* chapter.

DMX_GET_STC

Name

DMX_GET_STC

Synopsis

DMX_GET_STC

`int ioctl(int fd, DMX_GET_STC, struct dmx_stc *stc)`

Arguments

fd
: File descriptor returned by *open()*.

stc
: Pointer to *dmx_stc* where the stc data is to be stored.

Description

This ioctl call returns the current value of the system time counter (which is driven by a PES filter of type *DMX_PES_PCR*). Some hardware supports more than one STC, so you must specify which one by setting the *num* field of stc before the ioctl (range 0⋯n). The result is returned in form of a ratio with a 64 bit numerator and a 32 bit denominator, so the real 90kHz STC value is `stc->stc / stc->base`.

Return Value

On success 0 is returned.

On error -1 is returned, and the `errno` variable is set appropriately.

`EINVAL`	Invalid stc number.

The generic error codes are described at the *Generic Error Codes* chapter.

DMX_GET_PES_PIDS

Name

DMX_GET_PES_PIDS

Synopsis

DMX_GET_PES_PIDS

`int ioctl(fd, DMX_GET_PES_PIDS, __u16 pids[5])`

Arguments

fd
File descriptor returned by *open()*.

pids
Array used to store 5 Program IDs.

Description

This ioctl allows to query a DVB device to return the first PID used by audio, video, textext, subtitle and PCR programs on a given service. They're stored as:

PID element	position	content
pids[DMX_PES_AUDIO]	0	first audio PID
pids[DMX_PES_VIDEO]	1	first video PID
pids[DMX_PES_TELETEXT]	2	first teletext PID
pids[DMX_PES_SUBTITLE]	3	first subtitle PID
pids[DMX_PES_PCR]	4	first Program Clock Reference PID

Note: A value equal to 0xffff means that the PID was not filled by the Kernel.

Return Value

On success 0 is returned.

On error -1 is returned, and the `errno` variable is set appropriately.

The generic error codes are described at the *Generic Error Codes* chapter.

DMX_ADD_PID

Name

DMX_ADD_PID

Synopsis

DMX_ADD_PID

`int ioctl(fd, DMX_ADD_PID, __u16 *pid)`

Arguments

fd
: File descriptor returned by *open()*.

pid
: PID number to be filtered.

Description

This ioctl call allows to add multiple PIDs to a transport stream filter previously set up with *DMX_SET_PES_FILTER* and output equal to *DMX_OUT_TSDEMUX_TAP*.

Return Value

On success 0 is returned.

On error -1 is returned, and the `errno` variable is set appropriately.

Generic error codes are described at the *Generic Error Codes* chapter.

DMX_REMOVE_PID

Name

DMX_REMOVE_PID

Synopsis

DMX_REMOVE_PID

`int ioctl(fd, DMX_REMOVE_PID, __u16 *pid)`

Arguments

fd
File descriptor returned by *open()*.

pid
PID of the PES filter to be removed.

Description

This ioctl call allows to remove a PID when multiple PIDs are set on a transport stream filter, e. g. a filter previously set up with output equal to *DMX_OUT_TSDEMUX_TAP*, created via either *DMX_SET_PES_FILTER* or *DMX_ADD_PID*.

Return Value

On success 0 is returned.

On error -1 is returned, and the `errno` variable is set appropriately.

The generic error codes are described at the *Generic Error Codes* chapter.

ioctl DMX_REQBUFS

Name

DMX_REQBUFS - Initiate Memory Mapping and/or DMA buffer I/O

> **Warning:** this API is still experimental

Synopsis

DMX_REQBUFS

`int ioctl(int fd, DMX_REQBUFS, struct dmx_requestbuffers *argp)`

Arguments

fd
File descriptor returned by *open()*.

argp
Pointer to struct *dmx_requestbuffers*.

Description

This ioctl is used to initiate a memory mapped or DMABUF based demux I/O.

Memory mapped buffers are located in device memory and must be allocated with this ioctl before they can be mapped into the application's address space. User buffers are allocated by applications themselves, and this ioctl is merely used to switch the driver into user pointer I/O mode and to setup some internal structures. Similarly, DMABUF buffers are allocated by applications through a device driver, and this ioctl only configures the driver into DMABUF I/O mode without performing any direct allocation.

To allocate device buffers applications initialize all fields of the struct *dmx_requestbuffers* structure. They set the `count` field to the desired number of buffers, and `size` to the size of each buffer.

When the ioctl is called with a pointer to this structure, the driver will attempt to allocate the requested number of buffers and it stores the actual number allocated in the `count` field. The `count` can be smaller than the number requested, even zero, when the driver runs out of free memory. A larger number is also possible when the driver requires more buffers to function correctly. The actual allocated buffer size can is returned at `size`, and can be smaller than what's requested.

When this I/O method is not supported, the ioctl returns an `EOPNOTSUPP` error code.

Applications can call *ioctl DMX_REQBUFS* again to change the number of buffers, however this cannot succeed when any buffers are still mapped. A `count` value of zero frees all buffers, after aborting or finishing any DMA in progress.

Return Value

On success 0 is returned, on error -1 and the `errno` variable is set appropriately. The generic error codes are described at the *Generic Error Codes* chapter.

EOPNOTSUPP
The the requested I/O method is not supported.

ioctl DMX_QUERYBUF

Name

DMX_QUERYBUF - Query the status of a buffer

> **Warning:** this API is still experimental

Synopsis

DMX_QUERYBUF

`int ioctl(int fd, DMX_QUERYBUF, struct dvb_buffer *argp)`

Arguments

fd
: File descriptor returned by *open()*.

argp
: Pointer to struct `dvb_buffer`.

Description

This ioctl is part of the mmap streaming I/O method. It can be used to query the status of a buffer at any time after buffers have been allocated with the *ioctl DMX_REQBUFS* ioctl.

Applications set the `index` field. Valid index numbers range from zero to the number of buffers allocated with *ioctl DMX_REQBUFS* (struct `dvb_requestbuffers count`) minus one.

After calling *ioctl DMX_QUERYBUF* with a pointer to this structure, drivers return an error code or fill the rest of the structure.

On success, the `offset` will contain the offset of the buffer from the start of the device memory, the `length` field its size, and the `bytesused` the number of bytes occupied by data in the buffer (payload).

Return Value

On success 0 is returned, the `offset` will contain the offset of the buffer from the start of the device memory, the `length` field its size, and the `bytesused` the number of bytes occupied by data in the buffer (payload).

On error it returns -1 and the `errno` variable is set appropriately. The generic error codes are described at the *Generic Error Codes* chapter.

EINVAL
: The `index` is out of bounds.

ioctl DMX_EXPBUF

Name

DMX_EXPBUF - Export a buffer as a DMABUF file descriptor.

Warning: this API is still experimental

Synopsis

DMX_EXPBUF

`int ioctl(int fd, DMX_EXPBUF, struct dmx_exportbuffer *argp)`

Arguments

fd
: File descriptor returned by *open()*.

argp
: Pointer to struct *dmx_exportbuffer*.

Description

This ioctl is an extension to the memory mapping I/O method. It can be used to export a buffer as a DMABUF file at any time after buffers have been allocated with the *ioctl DMX_REQBUFS* ioctl.

To export a buffer, applications fill struct *dmx_exportbuffer*. Applications must set the `index` field. Valid index numbers range from zero to the number of buffers allocated with *ioctl DMX_REQBUFS* (struct *dmx_requestbuffers* `count`) minus one. Additional flags may be posted in the `flags` field. Refer to a manual for open() for details. Currently only O_CLOEXEC, O_RDONLY, O_WRONLY, and O_RDWR are supported. All other fields must be set to zero. In the case of multi-planar API, every plane is exported separately using multiple *ioctl DMX_EXPBUF* calls.

After calling *ioctl DMX_EXPBUF* the `fd` field will be set by a driver, on success. This is a DMABUF file descriptor. The application may pass it to other DMABUF-aware devices. It is recommended to close a DMABUF file when it is no longer used to allow the associated memory to be reclaimed.

Examples

```
int buffer_export(int v4lfd, enum dmx_buf_type bt, int index, int␣
↪*dmafd)
{
    struct dmx_exportbuffer expbuf;

    memset(&expbuf, 0, sizeof(expbuf));
    expbuf.type = bt;
    expbuf.index = index;
    if (ioctl(v4lfd, DMX_EXPBUF, &expbuf) == -1) {
        perror("DMX_EXPBUF");
        return -1;
    }

    *dmafd = expbuf.fd;

    return 0;
}
```

Return Value

On success 0 is returned, on error -1 and the `errno` variable is set appropriately. The generic error codes are described at the *Generic Error Codes* chapter.

EINVAL
A queue is not in MMAP mode or DMABUF exporting is not supported or `flags` or `index` fields are invalid.

ioctl DMX_QBUF, DMX_DQBUF

Name

DMX_QBUF - DMX_DQBUF - Exchange a buffer with the driver

Warning: this API is still experimental

Synopsis

DMX_QBUF

`int ioctl(int fd, DMX_QBUF, struct dmx_buffer *argp)`

DMX_DQBUF

`int ioctl(int fd, DMX_DQBUF, struct dmx_buffer *argp)`

Arguments

fd
File descriptor returned by *open()*.

argp
Pointer to struct *dmx_buffer*.

Description

Applications call the `DMX_QBUF` ioctl to enqueue an empty (capturing) or filled (output) buffer in the driver's incoming queue. The semantics depend on the selected I/O method.

To enqueue a buffer applications set the `index` field. Valid index numbers range from zero to the number of buffers allocated with *ioctl DMX_REQBUFS* (struct *dmx_requestbuffers* `count`) minus one. The contents of the struct *dmx_buffer* returned by a *ioctl DMX_QUERYBUF* ioctl will do as well.

When `DMX_QBUF` is called with a pointer to this structure, it locks the memory pages of the buffer in physical memory, so they cannot be swapped out to disk. Buffers remain locked until dequeued, until the the device is closed.

Applications call the `DMX_DQBUF` ioctl to dequeue a filled (capturing) buffer from the driver's outgoing queue. They just set the `index` field with the buffer ID to be queued. When `DMX_DQBUF` is called with a pointer to struct *dmx_buffer*, the driver fills the remaining fields or returns an error code.

By default `DMX_DQBUF` blocks when no buffer is in the outgoing queue. When the `O_NONBLOCK` flag was given to the *open()* function, `DMX_DQBUF` returns immediately with an `EAGAIN` error code when no buffer is available.

The struct *dmx_buffer* structure is specified in *Buffers*.

Return Value

On success 0 is returned, on error -1 and the `errno` variable is set appropriately. The generic error codes are described at the *Generic Error Codes* chapter.

EAGAIN
Non-blocking I/O has been selected using `O_NONBLOCK` and no buffer was in the outgoing queue.

EINVAL
The `index` is out of bounds, or no buffers have been allocated yet.

EIO
`DMX_DQBUF` failed due to an internal error. Can also indicate temporary problems like signal loss or CRC errors.

8.3.4 Digital TV CA Device

The Digital TV CA device controls the conditional access hardware. It can be accessed through `/dev/dvb/adapter?/ca?`. Data types and and ioctl definitions can be accessed by including `linux/dvb/ca.h` in your application.

Note: There are three ioctls at this API that aren't documented: *CA_GET_MSG*, *CA_SEND_MSG* and *CA_SET_DESCR*. Documentation for them are welcome.

CA Data Types

struct **ca_slot_info**

CA slot interface types and info.

Definition

```
struct ca_slot_info {
  int num;
  int type;
#define CA_CI            1;
#define CA_CI_LINK       2;
#define CA_CI_PHYS       4;
#define CA_DESCR         8;
#define CA_SC          128;
  unsigned int flags;
#define CA_CI_MODULE_PRESENT 1;
#define CA_CI_MODULE_READY   2;
};
```

Members

num
slot number.

type
slot type.

flags
flags applicable to the slot.

Description

This struct stores the CA slot information.

type can be:

- `CA_CI` - CI high level interface;
- `CA_CI_LINK` - CI link layer level interface;
- `CA_CI_PHYS` - CI physical layer level interface;
- `CA_DESCR` - built-in descrambler;
- `CA_SC` -simple smart card interface.

flags can be:

- CA_CI_MODULE_PRESENT - module (or card) inserted;
- CA_CI_MODULE_READY - module is ready for usage.

struct **ca_descr_info**

descrambler types and info.

Definition

```
struct ca_descr_info {
  unsigned int num;
  unsigned int type;
#define CA_ECD           1;
#define CA_NDS           2;
#define CA_DSS           4;
};
```

Members

num
number of available descramblers (keys).

type
type of supported scrambling system.

Description

Identifies the number of descramblers and their type.

type can be:

- CA_ECD - European Common Descrambler (ECD) hardware;
- CA_NDS - Videoguard (NDS) hardware;
- CA_DSS - Distributed Sample Scrambling (DSS) hardware.

struct **ca_caps**

CA slot interface capabilities.

Definition

```
struct ca_caps {
  unsigned int slot_num;
  unsigned int slot_type;
  unsigned int descr_num;
  unsigned int descr_type;
};
```

Members

slot_num
total number of CA card and module slots.

slot_type
bitmap with all supported types as defined at *struct ca_slot_info* (e. g. CA_CI, CA_CI_LINK, etc).

descr_num
  total number of descrambler slots (keys)

descr_type
  bitmap with all supported types as defined at *struct ca_descr_info* (e. g. CA_ECD, CA_NDS, etc).

struct **ca_msg**
  a message to/from a CI-CAM

Definition

```
struct ca_msg {
  unsigned int index;
  unsigned int type;
  unsigned int length;
  unsigned char msg[256];
};
```

Members

index
  unused

type
  unused

length
  length of the message

msg
  message

Description

This struct carries a message to be send/received from a CI CA module.

struct **ca_descr**
  CA descrambler control words info

Definition

```
struct ca_descr {
  unsigned int index;
  unsigned int parity;
  unsigned char cw[8];
};
```

Members

index
  CA Descrambler slot

parity
  control words parity, where 0 means even and 1 means odd

cw
  CA Descrambler control words

CA Function Calls

Digital TV CA open()

Name

Digital TV CA open()

Synopsis

int **open**(const char *name, int flags)

Arguments

name
Name of specific Digital TV CA device.

flags
A bit-wise OR of the following flags:

O_RDONLY	read-only access
O_RDWR	read/write access
O_NONBLOCK	open in non-blocking mode (blocking mode is the default)

Description

This system call opens a named ca device (e.g. `/dev/dvb/adapter?/ca?`) for subsequent use.

When an `open()` call has succeeded, the device will be ready for use. The significance of blocking or non-blocking mode is described in the documentation for functions where there is a difference. It does not affect the semantics of the `open()` call itself. A device opened in blocking mode can later be put into non-blocking mode (and vice versa) using the `F_SETFL` command of the `fcntl` system call. This is a standard system call, documented in the Linux manual page for fcntl. Only one user can open the CA Device in `O_RDWR` mode. All other attempts to open the device in this mode will fail, and an error code will be returned.

Return Value

On success 0 is returned.

On error -1 is returned, and the `errno` variable is set appropriately.

Generic error codes are described at the *Generic Error Codes* chapter.

Digital TV CA close()

Name

Digital TV CA close()

Synopsis

int **close**(int fd)

Arguments

fd
: File descriptor returned by a previous call to *open()*.

Description

This system call closes a previously opened CA device.

Return Value

On success 0 is returned.

On error -1 is returned, and the `errno` variable is set appropriately.

Generic error codes are described at the *Generic Error Codes* chapter.

CA_RESET

Name

CA_RESET

Synopsis

CA_RESET

`int ioctl(fd, CA_RESET)`

Arguments

fd
File descriptor returned by a previous call to *open()*.

Description

Puts the Conditional Access hardware on its initial state. It should be called before start using the CA hardware.

Return Value

On success 0 is returned.

On error -1 is returned, and the `errno` variable is set appropriately.

Generic error codes are described at the *Generic Error Codes* chapter.

CA_GET_CAP

Name

CA_GET_CAP

Synopsis

CA_GET_CAP

`int ioctl(fd, CA_GET_CAP, struct ca_caps *caps)`

Arguments

fd
File descriptor returned by a previous call to *open()*.

caps
Pointer to struct *ca_caps*.

Description

Queries the Kernel for information about the available CA and descrambler slots, and their types.

Return Value

On success 0 is returned and *ca_caps* is filled.

On error, -1 is returned and the `errno` variable is set appropriately.

The generic error codes are described at the *Generic Error Codes* chapter.

CA_GET_SLOT_INFO

Name

CA_GET_SLOT_INFO

Synopsis

CA_GET_SLOT_INFO

`int ioctl(fd, CA_GET_SLOT_INFO, struct ca_slot_info *info)`

Arguments

fd
: File descriptor returned by a previous call to *open()*.

info
: Pointer to struct *ca_slot_info*.

Description

Returns information about a CA slot identified by *ca_slot_info*.slot_num.

Return Value

On success 0 is returned, and *ca_slot_info* is filled.

On error -1 is returned, and the `errno` variable is set appropriately.

ENODEV	the slot is not available.

The generic error codes are described at the *Generic Error Codes* chapter.

CA_GET_DESCR_INFO

Name

CA_GET_DESCR_INFO

Synopsis

CA_GET_DESCR_INFO

`int ioctl(fd, CA_GET_DESCR_INFO, struct ca_descr_info *desc)`

Arguments

fd
File descriptor returned by a previous call to *open()*.

desc
Pointer to struct *ca_descr_info*.

Description

Returns information about all descrambler slots.

Return Value

On success 0 is returned, and *ca_descr_info* is filled.

On error -1 is returned, and the `errno` variable is set appropriately. The generic error codes are described at the *Generic Error Codes* chapter.

CA_GET_MSG

Name

CA_GET_MSG

Synopsis

CA_GET_MSG

`int ioctl(fd, CA_GET_MSG, struct ca_msg *msg)`

Arguments

fd
File descriptor returned by a previous call to *open()*.

msg
Pointer to struct *ca_msg*.

Description

Receives a message via a CI CA module.

Note: Please notice that, on most drivers, this is done by reading from the /dev/adapter?/ca? device node.

Return Value

On success 0 is returned.

On error -1 is returned, and the `errno` variable is set appropriately.

Generic error codes are described at the *Generic Error Codes* chapter.

CA_SEND_MSG

Name

CA_SEND_MSG

Synopsis

CA_SEND_MSG

`int ioctl(fd, CA_SEND_MSG, struct ca_msg *msg)`

Arguments

fd
File descriptor returned by a previous call to *open()*.

msg
Pointer to struct *ca_msg*.

Description

Sends a message via a CI CA module.

Note: Please notice that, on most drivers, this is done by writing to the /dev/adapter?/ca? device node.

Return Value

On success 0 is returned.

On error -1 is returned, and the `errno` variable is set appropriately.

Generic error codes are described at the *Generic Error Codes* chapter.

CA_SET_DESCR

Name

CA_SET_DESCR

Synopsis

CA_SET_DESCR

`int ioctl(fd, CA_SET_DESCR, struct ca_descr *desc)`

Arguments

fd
: File descriptor returned by a previous call to *open()*.

msg
: Pointer to struct *ca_descr*.

Description

CA_SET_DESCR is used for feeding descrambler CA slots with descrambling keys (referred as control words).

Return Value

On success 0 is returned.

On error -1 is returned, and the `errno` variable is set appropriately.

Generic error codes are described at the *Generic Error Codes* chapter.

The High level CI API

Note: This documentation is outdated.

This document describes the high level CI API as in accordance to the Linux DVB API.

With the High Level CI approach any new card with almost any random architecture can be implemented with this style, the definitions inside the switch statement can be easily adapted for any card, thereby eliminating the need for any additional ioctls.

The disadvantage is that the driver/hardware has to manage the rest. For the application programmer it would be as simple as sending/receiving an array to/from the CI ioctls as defined in the Linux DVB API. No changes have been made in the API to accommodate this feature.

Why the need for another CI interface?

This is one of the most commonly asked question. Well a nice question. Strictly speaking this is not a new interface.

The CI interface is defined in the DVB API in ca.h as:

```
typedef struct ca_slot_info {
        int num;               /* slot number */

        int type;              /* CA interface this slot supports */
#define CA_CI            1     /* CI high level interface */
#define CA_CI_LINK       2     /* CI link layer level interface */
#define CA_CI_PHYS       4     /* CI physical layer level interface␣
↪*/
#define CA_DESCR         8     /* built-in descrambler */
#define CA_SC          128     /* simple smart card interface */

        unsigned int flags;
#define CA_CI_MODULE_PRESENT 1 /* module (or card) inserted */
#define CA_CI_MODULE_READY   2
} ca_slot_info_t;
```

This CI interface follows the CI high level interface, which is not implemented by most applications. Hence this area is revisited.

This CI interface is quite different in the case that it tries to accommodate all other CI based devices, that fall into the other categories.

This means that this CI interface handles the EN50221 style tags in the Application layer only and no session management is taken care of by the application. The driver/hardware will take care of all that.

This interface is purely an EN50221 interface exchanging APDU's. This means that no session management, link layer or a transport layer do exist in this case in the application to driver communication. It is as simple as that. The driver/hardware has to take care of that.

With this High Level CI interface, the interface can be defined with the regular ioctls.

All these ioctls are also valid for the High level CI interface

#define CA_RESET _IO('o' , 128) #define CA_GET_CAP _IOR('o' , 129, ca_caps_t) #define CA_GET_SLOT_INFO _IOR('o' , 130, ca_slot_info_t) #define CA_GET_DESCR_INFO _IOR('o' , 131, ca_descr_info_t) #define CA_GET_MSG _IOR('o' , 132, ca_msg_t) #define CA_SEND_MSG _IOW('o' , 133, ca_msg_t) #define CA_SET_DESCR _IOW('o' , 134, ca_descr_t)

On querying the device, the device yields information thus:

```
CA_GET_SLOT_INFO
----------------------------
Command = [info]
APP: Number=[1]
APP: Type=[1]
APP: flags=[1]
APP: CI High level interface
APP: CA/CI Module Present

CA_GET_CAP
----------------------------
Command = [caps]
APP: Slots=[1]
APP: Type=[1]
APP: Descrambler keys=[16]
APP: Type=[1]

CA_SEND_MSG
----------------------------
Descriptors(Program Level)=[ 09 06 06 04 05 50 ff f1]
Found CA descriptor @ program level

(20) ES type=[2] ES pid=[201]  ES length =[0 (0x0)]
(25) ES type=[4] ES pid=[301]  ES length =[0 (0x0)]
ca_message length is 25 (0x19) bytes
EN50221 CA MSG=[ 9f 80 32 19 03 01 2d d1 f0 08 01 09 06 06 04 05 50 ff f1 02 e0 c9 00 00 04 e1 2d 00 00]
```

Not all ioctl' s are implemented in the driver from the API, the other features of the hardware that cannot be implemented by the API are achieved using the CA_GET_MSG and CA_SEND_MSG ioctls. An EN50221 style wrapper is used to exchange the data to maintain compatibility with other hardware.

```
/* a message to/from a CI-CAM */
typedef struct ca_msg {
        unsigned int index;
        unsigned int type;
        unsigned int length;
        unsigned char msg[256];
} ca_msg_t;
```

The flow of data can be described thus,

```
     App (User)
     -----
     parse
       |
       |
       v
     en50221 APDU (package)
--------------------------------------
|      |                             | High Level CI driver
|      |                             |
|      v                             |
|    en50221 APDU (unpackage)        |
|      |                             |
|      |                             |
|      v                             |
|    sanity checks                   |
|      |                             |
|      |                             |
|      v                             |
|    do (H/W dep)                    |
--------------------------------------
       |    Hardware
       |
       v
```

The High Level CI interface uses the EN50221 DVB standard, following a standard ensures futureproofness.

8.3.5 Digital TV Network API

The Digital TV net device controls the mapping of data packages that are part of a transport stream to be mapped into a virtual network interface, visible through the standard Linux network protocol stack.

Currently, two encapsulations are supported:

- Multi Protocol Encapsulation (MPE)
- Ultra Lightweight Encapsulation (ULE)

In order to create the Linux virtual network interfaces, an application needs to tell to the Kernel what are the PIDs and the encapsulation types that are present on the transport stream. This is done through `/dev/dvb/adapter?/net?` device node. The data will be available via virtual `dvb?_?` network interfaces, and will be controlled/routed via the standard ip tools (like ip, route, netstat, ifconfig, etc).

Data types and and ioctl definitions are defined via `linux/dvb/net.h` header.

Digital TV net Function Calls

Net Data Types

struct **dvb_net_if**

describes a DVB network interface

Definition

```
struct dvb_net_if {
  __u16 pid;
  __u16 if_num;
  __u8 feedtype;
#define DVB_NET_FEEDTYPE_MPE 0  ;
#define DVB_NET_FEEDTYPE_ULE 1  ;
};
```

Members

pid
Packet ID (PID) of the MPEG-TS that contains data

if_num
number of the Digital TV interface.

feedtype
Encapsulation type of the feed.

Description

A MPEG-TS stream may contain packet IDs with IP packages on it. This struct describes it, and the type of encoding.

feedtype can be:

- `DVB_NET_FEEDTYPE_MPE` for MPE encoding

- DVB_NET_FEEDTYPE_ULE for ULE encoding.

ioctl NET_ADD_IF

Name

NET_ADD_IF - Creates a new network interface for a given Packet ID.

Synopsis

NET_ADD_IF

`int ioctl(int fd, NET_ADD_IF, struct dvb_net_if *net_if)`

Arguments

fd
File descriptor returned by `open()`.

net_if
pointer to struct *dvb_net_if*

Description

The NET_ADD_IF ioctl system call selects the Packet ID (PID) that contains a TCP/IP traffic, the type of encapsulation to be used (MPE or ULE) and the interface number for the new interface to be created. When the system call successfully returns, a new virtual network interface is created.

The struct *dvb_net_if*::ifnum field will be filled with the number of the created interface.

Return Value

On success 0 is returned, and *ca_slot_info* is filled.

On error -1 is returned, and the `errno` variable is set appropriately.

The generic error codes are described at the *Generic Error Codes* chapter.

ioctl NET_REMOVE_IF

Name

NET_REMOVE_IF - Removes a network interface.

Synopsis

NET_REMOVE_IF

`int ioctl(int fd, NET_REMOVE_IF, int ifnum)`

Arguments

fd
File descriptor returned by `open()`.

net_if
number of the interface to be removed

Description

The NET_REMOVE_IF ioctl deletes an interface previously created via *NET_ADD_IF*.

Return Value

On success 0 is returned, and *ca_slot_info* is filled.

On error -1 is returned, and the `errno` variable is set appropriately.

The generic error codes are described at the *Generic Error Codes* chapter.

ioctl NET_GET_IF

Name

NET_GET_IF - Read the configuration data of an interface created via - *NET_ADD_IF*.

Synopsis

NET_GET_IF

`int ioctl(int fd, NET_GET_IF, struct dvb_net_if *net_if)`

Arguments

fd
File descriptor returned by `open()`.

net_if
pointer to struct *dvb_net_if*

Description

The NET_GET_IF ioctl uses the interface number given by the struct *dvb_net_if*::ifnum field and fills the content of struct *dvb_net_if* with the packet ID and encapsulation type used on such interface. If the interface was not created yet with *NET_ADD_IF*, it will return -1 and fill the `errno` with `EINVAL` error code.

Return Value

On success 0 is returned, and *ca_slot_info* is filled.

On error -1 is returned, and the `errno` variable is set appropriately.

The generic error codes are described at the *Generic Error Codes* chapter.

8.3.6 Digital TV Deprecated APIs

The APIs described here **should not** be used on new drivers or applications.

The DVBv3 frontend API has issues with new delivery systems, including DVB-S2, DVB-T2, ISDB, etc.

There's just one driver for a very legacy hardware using the Digital TV audio and video APIs. No modern drivers should use it. Instead, audio and video should be using the V4L2 and ALSA APIs, and the pipelines should be set via the Media Controller API.

> **Attention:** The APIs described here doesn't necessarily reflect the current code implementation, as this section of the document was written for DVB version 1, while the code reflects DVB version 3 implementation.

Digital TV Frontend legacy API (a. k. a. DVBv3)

The usage of this API is deprecated, as it doesn't support all digital TV standards, doesn't provide good statistics measurements and provides incomplete information. This is kept only to support legacy applications.

Frontend Legacy Data Types

Frontend type

For historical reasons, frontend types are named by the type of modulation used in transmission. The fontend types are given by fe_type_t type, defined as:

type **fe_type**

Table 225: Frontend types

fe_type	Description	*DTV_DELIVERY_SYSTEM* equivalent type
`FE_QPSK`	For DVB-S standard	`SYS_DVBS`
`FE_QAM`	For DVB-C annex A standard	`SYS_DVBC_ANNEX_A`
`FE_OFDM`	For DVB-T standard	`SYS_DVBT`
`FE_ATSC`	For ATSC standard (terrestrial) or for DVB-C Annex B (cable) used in US.	`SYS_ATSC` (terrestrial) or `SYS_DVBC_ANNEX_B` (cable)

Newer formats like DVB-S2, ISDB-T, ISDB-S and DVB-T2 are not described at the above, as they're supported via the new *FE_GET_PROPERTY/FE_GET_SET_PROPERTY* ioctl's, using the *DTV_DELIVERY_SYSTEM* parameter.

In the old days, struct *dvb_frontend_info* used to contain `fe_type_t` field to indicate the delivery systems, filled with either `FE_QPSK`, `FE_QAM`, `FE_OFDM` or `FE_ATSC`. While this is still filled to keep backward compatibility, the usage of this field is deprecated, as it can report just one delivery system, but some devices support multiple delivery systems. Please use *DTV_ENUM_DELSYS* instead.

On devices that support multiple delivery systems, struct *dvb_frontend_info*::`fe_type_t` is filled with the currently standard, as selected by the last call to *FE_SET_PROPERTY* using the *DTV_DELIVERY_SYSTEM* property.

Frontend bandwidth

type **fe_bandwidth**

Table 226: enum fe_bandwidth

ID	Description
BANDWIDTH_AUTO	Autodetect bandwidth (if supported)
BANDWIDTH_1_712_MHZ	1.712 MHz
BANDWIDTH_5_MHZ	5 MHz
BANDWIDTH_6_MHZ	6 MHz
BANDWIDTH_7_MHZ	7 MHz
BANDWIDTH_8_MHZ	8 MHz
BANDWIDTH_10_MHZ	10 MHz

type **dvb_frontend_parameters**

frontend parameters

The kind of parameters passed to the frontend device for tuning depend on the kind of hardware you are using.

The struct `dvb_frontend_parameters` uses a union with specific per-system parameters. However, as newer delivery systems required more data, the structure size weren't enough to fit, and just extending its size would break the existing applications. So, those parameters were replaced by the usage of *FE_GET_PROPERTY/FE_SET_PROPERTY* ioctl's. The new API is flexible enough to add new parameters to existing delivery systems, and to add newer delivery systems.

So, newer applications should use *FE_GET_PROPERTY/FE_SET_PROPERTY* instead, in order to be able to support the newer System Delivery like DVB-S2, DVB-T2, DVB-C2, ISDB, etc.

All kinds of parameters are combined as a union in the `dvb_frontend_parameters` structure:

```
struct dvb_frontend_parameters {
    uint32_t frequency;     /* (absolute) frequency in Hz for QAM/
↪OFDM */
                /* intermediate frequency in kHz for QPSK */
    fe_spectral_inversion_t inversion;
    union {
        struct dvb_qpsk_parameters qpsk;
        struct dvb_qam_parameters  qam;
        struct dvb_ofdm_parameters ofdm;
```

(continues on next page)

(continued from previous page)

```
        struct dvb_vsb_parameters  vsb;
    } u;
};
```

In the case of QPSK frontends the `frequency` field specifies the intermediate frequency, i.e. the offset which is effectively added to the local oscillator frequency (LOF) of the LNB. The intermediate frequency has to be specified in units of kHz. For QAM and OFDM frontends the `frequency` specifies the absolute frequency and is given in Hz.

type **dvb_qpsk_parameters**

QPSK parameters

For satellite QPSK frontends you have to use the `dvb_qpsk_parameters` structure:

```
struct dvb_qpsk_parameters {
    uint32_t        symbol_rate;  /* symbol rate in Symbols per␣
↪second */
    fe_code_rate_t  fec_inner;    /* forward error correction (see␣
↪above) */
};
```

type **dvb_qam_parameters**

QAM parameters

for cable QAM frontend you use the `dvb_qam_parameters` structure:

```
struct dvb_qam_parameters {
    uint32_t         symbol_rate; /* symbol rate in Symbols per␣
↪second */
    fe_code_rate_t   fec_inner;   /* forward error correction (see␣
↪above) */
    fe_modulation_t  modulation;  /* modulation type (see above) */
};
```

type **dvb_vsb_parameters**

VSB parameters

ATSC frontends are supported by the `dvb_vsb_parameters` structure:

```
struct dvb_vsb_parameters {
    fe_modulation_t modulation; /* modulation type (see above) */
};
```

type **dvb_ofdm_parameters**

OFDM parameters

DVB-T frontends are supported by the `dvb_ofdm_parameters` structure:

```
struct dvb_ofdm_parameters {
    fe_bandwidth_t      bandwidth;
    fe_code_rate_t      code_rate_HP;  /* high priority stream code␣
↪rate */
    fe_code_rate_t      code_rate_LP;  /* low priority stream code␣
↪rate */
    fe_modulation_t     constellation; /* modulation type (see␣
↪above) */
    fe_transmit_mode_t  transmission_mode;
    fe_guard_interval_t guard_interval;
    fe_hierarchy_t      hierarchy_information;
};
```

type **dvb_frontend_event**

frontend events

```
struct dvb_frontend_event {
    fe_status_t status;
    struct dvb_frontend_parameters parameters;
};
```

Frontend Legacy Function Calls

Those functions are defined at DVB version 3. The support is kept in the kernel due to compatibility issues only. Their usage is strongly not recommended

FE_READ_BER

Name

FE_READ_BER

Attention: This ioctl is deprecated.

Synopsis

FE_READ_BER

`int ioctl(int fd, FE_READ_BER, uint32_t *ber)`

Arguments

fd
: File descriptor returned by *open()*.

ber
: The bit error rate is stored into *ber.

Description

This ioctl call returns the bit error rate for the signal currently received/demodulated by the front-end. For this command, read-only access to the device is sufficient.

Return Value

On success 0 is returned.

On error -1 is returned, and the `errno` variable is set appropriately.

Generic error codes are described at the *Generic Error Codes* chapter.

FE_READ_SNR

Name

FE_READ_SNR

Attention: This ioctl is deprecated.

Synopsis

FE_READ_SNR

`int ioctl(int fd, FE_READ_SNR, int16_t *snr)`

Arguments

fd
File descriptor returned by *open()*.

snr
The signal-to-noise ratio is stored into *snr.

Description

This ioctl call returns the signal-to-noise ratio for the signal currently received by the front-end. For this command, read-only access to the device is sufficient.

Return Value

On success 0 is returned.

On error -1 is returned, and the `errno` variable is set appropriately.

Generic error codes are described at the *Generic Error Codes* chapter.

FE_READ_SIGNAL_STRENGTH

Name

FE_READ_SIGNAL_STRENGTH

Attention: This ioctl is deprecated.

Synopsis

FE_READ_SIGNAL_STRENGTH

`int ioctl(int fd, FE_READ_SIGNAL_STRENGTH, uint16_t *strength)`

Arguments

fd
File descriptor returned by *open()*.

strength
The signal strength value is stored into *strength.

Description

This ioctl call returns the signal strength value for the signal currently received by the front-end. For this command, read-only access to the device is sufficient.

Return Value

On success 0 is returned.

On error -1 is returned, and the `errno` variable is set appropriately.

Generic error codes are described at the *Generic Error Codes* chapter.

FE_READ_UNCORRECTED_BLOCKS

Name

FE_READ_UNCORRECTED_BLOCKS

Attention: This ioctl is deprecated.

Synopsis

FE_READ_UNCORRECTED_BLOCKS

```
int ioctl(int fd, FE_READ_UNCORRECTED_BLOCKS, uint32_t *ublocks)
```

Arguments

fd
File descriptor returned by *open()*.

ublocks
The total number of uncorrected blocks seen by the driver so far.

Description

This ioctl call returns the number of uncorrected blocks detected by the device driver during its lifetime. For meaningful measurements, the increment in block count during a specific time interval should be calculated. For this command, read-only access to the device is sufficient.

Return Value

On success 0 is returned.

On error -1 is returned, and the `errno` variable is set appropriately.

Generic error codes are described at the *Generic Error Codes* chapter.

FE_SET_FRONTEND

Attention: This ioctl is deprecated.

Name

FE_SET_FRONTEND

Synopsis

FE_SET_FRONTEND

```
int ioctl(int fd, FE_SET_FRONTEND, struct dvb_frontend_parameters
*p)
```

Arguments

fd
: File descriptor returned by *open()*.

p
: Points to parameters for tuning operation.

Description

This ioctl call starts a tuning operation using specified parameters. The result of this call will be successful if the parameters were valid and the tuning could be initiated. The result of the tuning operation in itself, however, will arrive asynchronously as an event (see documentation for *FE_GET_EVENT* and FrontendEvent.) If a new *FE_SET_FRONTEND* operation is initiated before the previous one was completed, the previous operation will be aborted in favor of the new one. This command requires read/write access to the device.

Return Value

On success 0 is returned.

On error -1 is returned, and the `errno` variable is set appropriately.

`EINVAL`	Maximum supported symbol rate reached.

Generic error codes are described at the *Generic Error Codes* chapter.

FE_GET_FRONTEND

Name

FE_GET_FRONTEND

Attention: This ioctl is deprecated.

Synopsis

`FE_GET_FRONTEND`

`int ioctl(int fd, FE_GET_FRONTEND, struct dvb_frontend_parameters *p)`

Arguments

`fd`

File descriptor returned by *`open()`*.

`p`

Points to parameters for tuning operation.

Description

This ioctl call queries the currently effective frontend parameters. For this command, read-only access to the device is sufficient.

Return Value

On success 0 is returned.

On error -1 is returned, and the `errno` variable is set appropriately.

`EINVAL`	Maximum supported symbol rate reached.

Generic error codes are described at the *Generic Error Codes* chapter.

FE_GET_EVENT

Name

FE_GET_EVENT

Attention: This ioctl is deprecated.

Synopsis

FE_GET_EVENT

`int ioctl(int fd, FE_GET_EVENT, struct dvb_frontend_event *ev)`

Arguments

fd
: File descriptor returned by *open()*.

ev
: Points to the location where the event, if any, is to be stored.

Description

This ioctl call returns a frontend event if available. If an event is not available, the behavior depends on whether the device is in blocking or non-blocking mode. In the latter case, the call fails immediately with errno set to `EWOULDBLOCK`. In the former case, the call blocks until an event becomes available.

Return Value

On success 0 is returned.

On error -1 is returned, and the `errno` variable is set appropriately.

`EWOULDBLOCK`	There is no event pending, and the device is in non-blocking mode.
`EOVERFLOW`	Overflow in event queue - one or more events were lost.

Generic error codes are described at the *Generic Error Codes* chapter.

FE_DISHNETWORK_SEND_LEGACY_CMD

Name

FE_DISHNETWORK_SEND_LEGACY_CMD

Synopsis

FE_DISHNETWORK_SEND_LEGACY_CMD

`int ioctl(int fd, FE_DISHNETWORK_SEND_LEGACY_CMD, unsigned long cmd)`

Arguments

fd
: File descriptor returned by *open()*.

cmd
: Sends the specified raw cmd to the dish via DISEqC.

Description

> **Warning:** This is a very obscure legacy command, used only at stv0299 driver. Should not be used on newer drivers.

It provides a non-standard method for selecting Diseqc voltage on the frontend, for Dish Network legacy switches.

As support for this ioctl were added in 2004, this means that such dishes were already legacy in 2004.

Return Value

On success 0 is returned.

On error -1 is returned, and the `errno` variable is set appropriately.

Generic error codes are described at the *Generic Error Codes* chapter.

Digital TV Video Device

The Digital TV video device controls the MPEG2 video decoder of the Digital TV hardware. It can be accessed through **/dev/dvb/adapter0/video0**. Data types and and ioctl definitions can be accessed by including **linux/dvb/video.h** in your application.

Note that the Digital TV video device only controls decoding of the MPEG video stream, not its presentation on the TV or computer screen. On PCs this is typically handled by an associated video4linux device, e.g. **/dev/video**, which allows scaling and defining output windows.

Some Digital TV cards don't have their own MPEG decoder, which results in the omission of the audio and video device as well as the video4linux device.

The ioctls that deal with SPUs (sub picture units) and navigation packets are only supported on some MPEG decoders made for DVD playback.

These ioctls were also used by V4L2 to control MPEG decoders implemented in V4L2. The use of these ioctls for that purpose has been made obsolete and proper V4L2 ioctls or controls have been created to replace that functionality.

Video Data Types

video_format_t

The `video_format_t` data type defined by

```
typedef enum {
    VIDEO_FORMAT_4_3,     /* Select 4:3 format */
```

(continues on next page)

(continued from previous page)

```
    VIDEO_FORMAT_16_9,    /* Select 16:9 format. */
    VIDEO_FORMAT_221_1    /* 2.21:1 */
} video_format_t;
```

is used in the VIDEO_SET_FORMAT function (??) to tell the driver which aspect ratio the output hardware (e.g. TV) has. It is also used in the data structures video_status (??) returned by VIDEO_GET_STATUS (??) and video_event (??) returned by VIDEO_GET_EVENT (??) which report about the display format of the current video stream.

video_displayformat_t

In case the display format of the video stream and of the display hardware differ the application has to specify how to handle the cropping of the picture. This can be done using the VIDEO_SET_DISPLAY_FORMAT call (??) which accepts

```
typedef enum {
    VIDEO_PAN_SCAN,       /* use pan and scan format */
    VIDEO_LETTER_BOX,     /* use letterbox format */
    VIDEO_CENTER_CUT_OUT  /* use center cut out format */
} video_displayformat_t;
```

as argument.

video_stream_source_t

The video stream source is set through the VIDEO_SELECT_SOURCE call and can take the following values, depending on whether we are replaying from an internal (demuxer) or external (user write) source.

```
typedef enum {
    VIDEO_SOURCE_DEMUX, /* Select the demux as the main source */
    VIDEO_SOURCE_MEMORY /* If this source is selected, the stream
                comes from the user through the write
                system call */
} video_stream_source_t;
```

VIDEO_SOURCE_DEMUX selects the demultiplexer (fed either by the frontend or the DVR device) as the source of the video stream. If VIDEO_SOURCE_MEMORY is selected the stream comes from the application through the **write()** system call.

video_play_state_t

The following values can be returned by the VIDEO_GET_STATUS call representing the state of video playback.

```
typedef enum {
    VIDEO_STOPPED, /* Video is stopped */
    VIDEO_PLAYING, /* Video is currently playing */
    VIDEO_FREEZED  /* Video is freezed */
} video_play_state_t;
```

type **video_command**

struct video_command

The structure must be zeroed before use by the application This ensures it can be extended safely in the future.

```
struct video_command {
    __u32 cmd;
    __u32 flags;
    union {
        struct {
            __u64 pts;
        } stop;

        struct {
            /* 0 or 1000 specifies normal speed,
               1 specifies forward single stepping,
               -1 specifies backward single stepping,
               >>1: playback at speed/1000 of the normal speed,
               <-1: reverse playback at (-speed/1000) of the normal␣
↪speed. */
            __s32 speed;
            __u32 format;
        } play;

        struct {
            __u32 data[16];
        } raw;
    };
};
```

video_size_t

```
typedef struct {
    int w;
    int h;
    video_format_t aspect_ratio;
} video_size_t;
```

type **video_event**

struct video_event

The following is the structure of a video event as it is returned by the VIDEO_GET_EVENT call.

```
struct video_event {
    __s32 type;
#define VIDEO_EVENT_SIZE_CHANGED        1
#define VIDEO_EVENT_FRAME_RATE_CHANGED  2
#define VIDEO_EVENT_DECODER_STOPPED     3
#define VIDEO_EVENT_VSYNC       4
    long timestamp;
    union {
        video_size_t size;
        unsigned int frame_rate;    /* in frames per 1000sec */
        unsigned char vsync_field;  /* unknown/odd/even/progressive␣
↪*/
    } u;
};
```

type **video_status**

struct video_status

The VIDEO_GET_STATUS call returns the following structure informing about various states of the playback operation.

```
struct video_status {
    int                   video_blank;   /* blank video on freeze?␣
↪*/
    video_play_state_t    play_state;    /* current state of␣
↪playback */
    video_stream_source_t stream_source; /* current source (demux/
↪memory) */
    video_format_t        video_format;  /* current aspect ratio of␣
↪stream */
    video_displayformat_t display_format;/* selected cropping mode␣
↪*/
};
```

If video_blank is set video will be blanked out if the channel is changed or if playback is stopped. Otherwise, the last picture will be displayed. play_state indicates if the video is currently frozen, stopped, or being played back. The stream_source corresponds to the selected source for the video stream. It can come either from the demultiplexer or from memory. The video_format indicates the aspect ratio (one of 4:3 or 16:9) of the currently played video stream. Finally, display_format corresponds to the selected cropping mode in case the source video format is not the same as the format of the output device.

type **video_still_picture**

struct video_still_picture

An I-frame displayed via the VIDEO_STILLPICTURE call is passed on within the following structure.

```
/* pointer to and size of a single iframe in memory */
struct video_still_picture {
    char *iFrame;        /* pointer to a single iframe in memory */
    int32_t size;
};
```

video capabilities

A call to VIDEO_GET_CAPABILITIES returns an unsigned integer with the following bits set according to the hardwares capabilities.

```
/* bit definitions for capabilities: */
/* can the hardware decode MPEG1 and/or MPEG2? */
#define VIDEO_CAP_MPEG1   1
#define VIDEO_CAP_MPEG2   2
/* can you send a system and/or program stream to video device?
   (you still have to open the video and the audio device but only
    send the stream to the video device) */
#define VIDEO_CAP_SYS     4
#define VIDEO_CAP_PROG    8
/* can the driver also handle SPU, NAVI and CSS encoded data?
   (CSS API is not present yet) */
#define VIDEO_CAP_SPU    16
#define VIDEO_CAP_NAVI   32
#define VIDEO_CAP_CSS    64
```

Video Function Calls

dvb video open()

Name

dvb video open()

Attention: This ioctl is deprecated.

Synopsis

int **open**(const char *deviceName, int flags)

Arguments

const char *deviceName	Name of specific video device.
int flags	A bit-wise OR of the following flags:
	O_RDONLY read-only access
	O_RDWR read/write access
	O_NONBLOCK open in non-blocking mode
	(blocking mode is the default)

Description

This system call opens a named video device (e.g. /dev/dvb/adapter0/video0) for subsequent use.

When an open() call has succeeded, the device will be ready for use. The significance of blocking or non-blocking mode is described in the documentation for functions where there is a difference. It does not affect the semantics of the open() call itself. A device opened in blocking mode can later be put into non-blocking mode (and vice versa) using the F_SETFL command of the fcntl system call. This is a standard system call, documented in the Linux manual page for fcntl. Only one user can open the Video Device in O_RDWR mode. All other attempts to open the device in this mode will fail, and an error-code will be returned. If the Video Device is opened in O_RDONLY mode, the only ioctl call that can be used is VIDEO_GET_STATUS. All other call will return an error code.

Return Value

ENODEV	Device driver not loaded/available.
EINTERNAL	Internal error.
EBUSY	Device or resource busy.
EINVAL	Invalid argument.

dvb video close()

Name

dvb video close()

Attention: This ioctl is deprecated.

Synopsis

int **close**(int fd)

Arguments

int fd	File descriptor returned by a previous call to open().

Description

This system call closes a previously opened video device.

Return Value

EBADF	fd is not a valid open file descriptor.

dvb video write()

Name

dvb video write()

Attention: This ioctl is deprecated.

Synopsis

size_t **write**(int fd, const void *buf, size_t count)

Arguments

int fd	File descriptor returned by a previous call to open().
void *buf	Pointer to the buffer containing the PES data.
size_t count	Size of buf.

Description

This system call can only be used if VIDEO_SOURCE_MEMORY is selected in the ioctl call VIDEO_SELECT_SOURCE. The data provided shall be in PES format, unless the capability allows other formats. If O_NONBLOCK is not specified the function will block until buffer space is available. The amount of data to be transferred is implied by count.

Return Value

`EPERM`	Mode VIDEO_SOURCE_MEMORY not selected.
`ENOMEM`	Attempted to write more data than the internal buffer can hold.
`EBADF`	fd is not a valid open file descriptor.

VIDEO_STOP

Name

VIDEO_STOP

Attention: This ioctl is deprecated.

Synopsis

VIDEO_STOP

`int ioctl(fd, VIDEO_STOP, boolean mode)`

Arguments

int fd	File descriptor returned by a previous call to open().
int request	Equals VIDEO_STOP for this command.
Boolean mode	Indicates how the screen shall be handled.
	TRUE: Blank screen when stop.
	FALSE: Show last decoded frame.

Description

This ioctl is for Digital TV devices only. To control a V4L2 decoder use the V4L2 *ioctl VIDIOC_DECODER_CMD, VIDIOC_TRY_DECODER_CMD* instead.

This ioctl call asks the Video Device to stop playing the current stream. Depending on the input parameter, the screen can be blanked out or displaying the last decoded frame.

Return Value

On success 0 is returned, on error -1 and the `errno` variable is set appropriately. The generic error codes are described at the *Generic Error Codes* chapter.

VIDEO_PLAY

Name

VIDEO_PLAY

> **Attention:** This ioctl is deprecated.

Synopsis

VIDEO_PLAY

`int ioctl(fd, VIDEO_PLAY)`

Arguments

int fd	File descriptor returned by a previous call to open().
int request	Equals VIDEO_PLAY for this command.

Description

This ioctl is for Digital TV devices only. To control a V4L2 decoder use the V4L2 *ioctl VIDIOC_DECODER_CMD, VIDIOC_TRY_DECODER_CMD* instead.

This ioctl call asks the Video Device to start playing a video stream from the selected source.

Return Value

On success 0 is returned, on error -1 and the `errno` variable is set appropriately. The generic error codes are described at the *Generic Error Codes* chapter.

VIDEO_FREEZE

Name

VIDEO_FREEZE

> **Attention:** This ioctl is deprecated.

Synopsis

VIDEO_FREEZE

`int ioctl(fd, VIDEO_FREEZE)`

Arguments

int fd	File descriptor returned by a previous call to open().
int request	Equals VIDEO_FREEZE for this command.

Description

This ioctl is for Digital TV devices only. To control a V4L2 decoder use the V4L2 *ioctl VIDIOC_DECODER_CMD, VIDIOC_TRY_DECODER_CMD* instead.

This ioctl call suspends the live video stream being played. Decoding and playing are frozen. It is then possible to restart the decoding and playing process of the video stream using the VIDEO_CONTINUE command. If VIDEO_SOURCE_MEMORY is selected in the ioctl call VIDEO_SELECT_SOURCE, the Digital TV subsystem will not decode any more data until the ioctl call VIDEO_CONTINUE or VIDEO_PLAY is performed.

Return Value

On success 0 is returned, on error -1 and the `errno` variable is set appropriately. The generic error codes are described at the *Generic Error Codes* chapter.

VIDEO_CONTINUE

Name

VIDEO_CONTINUE

> **Attention:** This ioctl is deprecated.

Synopsis

VIDEO_CONTINUE

```
int ioctl(fd, VIDEO_CONTINUE)
```

Arguments

int fd	File descriptor returned by a previous call to open().
int request	Equals VIDEO_CONTINUE for this command.

Description

This ioctl is for Digital TV devices only. To control a V4L2 decoder use the V4L2 *ioctl VIDIOC_DECODER_CMD, VIDIOC_TRY_DECODER_CMD* instead.

This ioctl call restarts decoding and playing processes of the video stream which was played before a call to VIDEO_FREEZE was made.

Return Value

On success 0 is returned, on error -1 and the `errno` variable is set appropriately. The generic error codes are described at the *Generic Error Codes* chapter.

VIDEO_SELECT_SOURCE

Name

VIDEO_SELECT_SOURCE

Attention: This ioctl is deprecated.

Synopsis

VIDEO_SELECT_SOURCE

`int ioctl(fd, VIDEO_SELECT_SOURCE, video_stream_source_t source)`

Arguments

int fd	File descriptor returned by a previous call to open().
int request	Equals VIDEO_SELECT_SOURCE for this command.
video_stream_source_t source	Indicates which source shall be used for the Video stream.

Description

This ioctl is for Digital TV devices only. This ioctl was also supported by the V4L2 ivtv driver, but that has been replaced by the ivtv-specific `IVTV_IOC_PASSTHROUGH_MODE` ioctl.

This ioctl call informs the video device which source shall be used for the input data. The possible sources are demux or memory. If memory is selected, the data is fed to the video device through the write command.

type **video_stream_source_t**

```
typedef enum {
        VIDEO_SOURCE_DEMUX, /* Select the demux as the main source␣
↪*/
        VIDEO_SOURCE_MEMORY /* If this source is selected, the␣
↪stream
                               comes from the user through the write
                               system call */
} video_stream_source_t;
```

Return Value

On success 0 is returned, on error -1 and the `errno` variable is set appropriately. The generic error codes are described at the *Generic Error Codes* chapter.

VIDEO_SET_BLANK

Name

VIDEO_SET_BLANK

Attention: This ioctl is deprecated.

Synopsis

VIDEO_SET_BLANK

`int ioctl(fd, VIDEO_SET_BLANK, boolean mode)`

Arguments

int fd	File descriptor returned by a previous call to open().
int request	Equals VIDEO_SET_BLANK for this command.
boolean mode	TRUE: Blank screen when stop.
	FALSE: Show last decoded frame.

Description

This ioctl call asks the Video Device to blank out the picture.

Return Value

On success 0 is returned, on error -1 and the `errno` variable is set appropriately. The generic error codes are described at the *Generic Error Codes* chapter.

VIDEO_GET_STATUS

Name

VIDEO_GET_STATUS

> **Attention:** This ioctl is deprecated.

Synopsis

VIDEO_GET_STATUS

`int ioctl(fd, VIDEO_GET_STATUS, struct video_status *status)`

Arguments

int fd	File descriptor returned by a previous call to open().
int request	Equals VIDEO_GET_STATUS for this command.
struct video_status *status	Returns the current status of the Video Device.

Description

This ioctl call asks the Video Device to return the current status of the device.

type **video_status**

```
struct video_status {
        int                  video_blank;   /* blank video on␣
↪freeze? */
        video_play_state_t    play_state;    /* current state of␣
↪playback */
        video_stream_source_t stream_source; /* current source␣
↪(demux/memory) */
        video_format_t        video_format;  /* current aspect␣
↪ratio of stream*/
        video_displayformat_t display_format;/* selected cropping␣
↪mode */
};
```

Return Value

On success 0 is returned, on error -1 and the `errno` variable is set appropriately. The generic error codes are described at the *Generic Error Codes* chapter.

VIDEO_GET_FRAME_COUNT

Name

VIDEO_GET_FRAME_COUNT

Attention: This ioctl is deprecated.

Synopsis

VIDEO_GET_FRAME_COUNT

`int ioctl(int fd, VIDEO_GET_FRAME_COUNT, __u64 *pts)`

Arguments

int fd	File descriptor returned by a previous call to open().
int request	Equals VIDEO_GET_FRAME_COUNT for this command.
__u64 *pts	Returns the number of frames displayed since the decoder was started.

Description

This ioctl is obsolete. Do not use in new drivers. For V4L2 decoders this ioctl has been replaced by the `V4L2_CID_MPEG_VIDEO_DEC_FRAME` control.

This ioctl call asks the Video Device to return the number of displayed frames since the decoder was started.

Return Value

On success 0 is returned, on error -1 and the `errno` variable is set appropriately. The generic error codes are described at the *Generic Error Codes* chapter.

VIDEO_GET_PTS

Name

VIDEO_GET_PTS

Attention: This ioctl is deprecated.

Synopsis

VIDEO_GET_PTS

```
int ioctl(int fd, VIDEO_GET_PTS, __u64 *pts)
```

Arguments

int fd	File descriptor returned by a previous call to open().
int re-que	Equals VIDEO_GET_PTS for this command.
__u *pts	Returns the 33-bit timestamp as defined in ITU T-REC-H.222.0 / ISO/IEC 13818-1. The PTS should belong to the currently played frame if possible, but may also be a value close to it like the PTS of the last decoded frame or the last PTS extracted by the PES parser.

Description

This ioctl is obsolete. Do not use in new drivers. For V4L2 decoders this ioctl has been replaced by the `V4L2_CID_MPEG_VIDEO_DEC_PTS` control.

This ioctl call asks the Video Device to return the current PTS timestamp.

Return Value

On success 0 is returned, on error -1 and the `errno` variable is set appropriately. The generic error codes are described at the *Generic Error Codes* chapter.

VIDEO_GET_EVENT

Name

VIDEO_GET_EVENT

> **Attention:** This ioctl is deprecated.

Synopsis

VIDEO_GET_EVENT

```
int ioctl(fd, VIDEO_GET_EVENT, struct video_event *ev)
```

Arguments

int fd	File descriptor returned by a previous call to open().
int request	Equals VIDEO_GET_EVENT for this command.
struct video_event *ev	Points to the location where the event, if any, is to be stored.

Description

This ioctl is for Digital TV devices only. To get events from a V4L2 decoder use the V4L2 *ioctl VIDIOC_DQEVENT* ioctl instead.

This ioctl call returns an event of type video_event if available. If an event is not available, the behavior depends on whether the device is in blocking or non-blocking mode. In the latter case, the call fails immediately with errno set to `EWOULDBLOCK`. In the former case, the call blocks until an event becomes available. The standard Linux poll() and/or select() system calls can be used with the device file descriptor to watch for new events. For select(), the file descriptor should be included in the exceptfds argument, and for poll(), POLLPRI should be specified as the wake-up condition. Read-only permissions are sufficient for this ioctl call.

type **video_event**

```
struct video_event {
        __s32 type;
#define VIDEO_EVENT_SIZE_CHANGED        1
#define VIDEO_EVENT_FRAME_RATE_CHANGED  2
#define VIDEO_EVENT_DECODER_STOPPED     3
#define VIDEO_EVENT_VSYNC               4
        long timestamp;
        union {
                video_size_t size;
                unsigned int frame_rate;        /* in frames per␣
↪1000sec */
                unsigned char vsync_field;      /* unknown/odd/even/
↪progressive */
        } u;
};
```

Return Value

On success 0 is returned, on error -1 and the `errno` variable is set appropriately. The generic error codes are described at the *Generic Error Codes* chapter.

`EWOULDBLOCK`	There is no event pending, and the device is in non-blocking mode.
`EOVERFLOW`	Overflow in event queue - one or more events were lost.

VIDEO_COMMAND

Name

VIDEO_COMMAND

Attention: This ioctl is deprecated.

Synopsis

VIDEO_COMMAND

`int ioctl(int fd, VIDEO_COMMAND, struct video_command *cmd)`

Arguments

int fd	File descriptor returned by a previous call to open().
int request	Equals VIDEO_COMMAND for this command.
struct video_command *cmd	Commands the decoder.

Description

This ioctl is obsolete. Do not use in new drivers. For V4L2 decoders this ioctl has been replaced by the *ioctl VIDIOC_DECODER_CMD, VIDIOC_TRY_DECODER_CMD* ioctl.

This ioctl commands the decoder. The `video_command` struct is a subset of the `v4l2_decoder_cmd` struct, so refer to the *ioctl VIDIOC_DECODER_CMD, VIDIOC_TRY_DECODER_CMD* documentation for more information.

type **video_command**

```
/* The structure must be zeroed before use by the application
This ensures it can be extended safely in the future. */
struct video_command {
        __u32 cmd;
        __u32 flags;
        union {
                struct {
                        __u64 pts;
                } stop;

                struct {
                        /* 0 or 1000 specifies normal speed,
```

(continues on next page)

(continued from previous page)

```
                        1 specifies forward single stepping,
                        -1 specifies backward single stepping,
                        >1: playback at speed/1000 of the normal␣
↪speed,
                        <-1: reverse playback at (-speed/1000) of␣
↪the normal speed. */
                        __s32 speed;
                        __u32 format;
                } play;

                struct {
                        __u32 data[16];
                } raw;
        };
};
```

Return Value

On success 0 is returned, on error -1 and the `errno` variable is set appropriately. The generic error codes are described at the *Generic Error Codes* chapter.

VIDEO_TRY_COMMAND

Name

VIDEO_TRY_COMMAND

> **Attention:** This ioctl is deprecated.

Synopsis

VIDEO_TRY_COMMAND

`int ioctl(int fd, VIDEO_TRY_COMMAND, struct video_command *cmd)`

Arguments

int fd	File descriptor returned by a previous call to open().
int request	Equals VIDEO_TRY_COMMAND for this command.
struct video_command *cmd	Try a decoder command.

Description

This ioctl is obsolete. Do not use in new drivers. For V4L2 decoders this ioctl has been replaced by the *VIDIOC_TRY_DECODER_CMD* ioctl.

This ioctl tries a decoder command. The `video_command` struct is a subset of the `v4l2_decoder_cmd` struct, so refer to the *VIDIOC_TRY_DECODER_CMD* documentation for more information.

Return Value

On success 0 is returned, on error -1 and the `errno` variable is set appropriately. The generic error codes are described at the *Generic Error Codes* chapter.

VIDEO_GET_SIZE

Name

VIDEO_GET_SIZE

Attention: This ioctl is deprecated.

Synopsis

`VIDEO_GET_SIZE`

`int ioctl(int fd, VIDEO_GET_SIZE, video_size_t *size)`

Arguments

int fd	File descriptor returned by a previous call to open().
int request	Equals VIDEO_GET_SIZE for this command.
video_size_t *size	Returns the size and aspect ratio.

Description

This ioctl returns the size and aspect ratio.

type **`video_size_t`**

Return Value

On success 0 is returned, on error -1 and the `errno` variable is set appropriately. The generic error codes are described at the *Generic Error Codes* chapter.

VIDEO_SET_DISPLAY_FORMAT

Name

VIDEO_SET_DISPLAY_FORMAT

Attention: This ioctl is deprecated.

Synopsis

`VIDEO_SET_DISPLAY_FORMAT`

`int ioctl(fd, VIDEO_SET_DISPLAY_FORMAT)`

Arguments

int fd	File descriptor returned by a previous call to open().
int request	Equals VIDEO_SET_DISPLAY_FORMAT for this command.
video_display_format_t format	Selects the video format to be used.

Description

This ioctl call asks the Video Device to select the video format to be applied by the MPEG chip on the video.

Return Value

On success 0 is returned, on error -1 and the `errno` variable is set appropriately. The generic error codes are described at the *Generic Error Codes* chapter.

VIDEO_STILLPICTURE

Name

VIDEO_STILLPICTURE

Attention: This ioctl is deprecated.

Synopsis

VIDEO_STILLPICTURE

`int ioctl(fd, VIDEO_STILLPICTURE, struct video_still_picture *sp)`

Arguments

int fd	File descriptor returned by a previous call to open().
int request	Equals VIDEO_STILLPICTURE for this command.
struct video_still_picture *sp	Pointer to a location where an I-frame and size is stored.

Description

This ioctl call asks the Video Device to display a still picture (I-frame). The input data shall contain an I-frame. If the pointer is NULL, then the current displayed still picture is blanked.

Return Value

On success 0 is returned, on error -1 and the `errno` variable is set appropriately. The generic error codes are described at the *Generic Error Codes* chapter.

VIDEO_FAST_FORWARD

Name

VIDEO_FAST_FORWARD

Attention: This ioctl is deprecated.

Synopsis

VIDEO_FAST_FORWARD

`int ioctl(fd, VIDEO_FAST_FORWARD, int nFrames)`

Arguments

int fd	File descriptor returned by a previous call to open().
int request	Equals VIDEO_FAST_FORWARD for this command.
int nFrames	The number of frames to skip.

Description

This ioctl call asks the Video Device to skip decoding of N number of I-frames. This call can only be used if VIDEO_SOURCE_MEMORY is selected.

Return Value

On success 0 is returned, on error -1 and the `errno` variable is set appropriately. The generic error codes are described at the *Generic Error Codes* chapter.

EPERM	Mode VIDEO_SOURCE_MEMORY not selected.

VIDEO_SLOWMOTION

Name

VIDEO_SLOWMOTION

Attention: This ioctl is deprecated.

Synopsis

VIDEO_SLOWMOTION

`int ioctl(fd, VIDEO_SLOWMOTION, int nFrames)`

Arguments

int fd	File descriptor returned by a previous call to open().
int request	Equals VIDEO_SLOWMOTION for this command.
int nFrames	The number of times to repeat each frame.

Description

This ioctl call asks the video device to repeat decoding frames N number of times. This call can only be used if VIDEO_SOURCE_MEMORY is selected.

Return Value

On success 0 is returned, on error -1 and the `errno` variable is set appropriately. The generic error codes are described at the *Generic Error Codes* chapter.

EPERM	Mode VIDEO_SOURCE_MEMORY not selected.

VIDEO_GET_CAPABILITIES

Name

VIDEO_GET_CAPABILITIES

Attention: This ioctl is deprecated.

Synopsis

VIDEO_GET_CAPABILITIES

`int ioctl(fd, VIDEO_GET_CAPABILITIES, unsigned int *cap)`

Arguments

int fd	File descriptor returned by a previous call to open().
int request	Equals VIDEO_GET_CAPABILITIES for this command.
unsigned int *cap	Pointer to a location where to store the capability information.

Description

This ioctl call asks the video device about its decoding capabilities. On success it returns and integer which has bits set according to the defines in section ??.

Return Value

On success 0 is returned, on error -1 and the `errno` variable is set appropriately. The generic error codes are described at the *Generic Error Codes* chapter.

VIDEO_CLEAR_BUFFER

Name

VIDEO_CLEAR_BUFFER

Attention: This ioctl is deprecated.

Synopsis

VIDEO_CLEAR_BUFFER

`int ioctl(fd, VIDEO_CLEAR_BUFFER)`

Arguments

int fd	File descriptor returned by a previous call to open().
int request	Equals VIDEO_CLEAR_BUFFER for this command.

Description

This ioctl call clears all video buffers in the driver and in the decoder hardware.

Return Value

On success 0 is returned, on error -1 and the `errno` variable is set appropriately. The generic error codes are described at the *Generic Error Codes* chapter.

VIDEO_SET_STREAMTYPE

Name

VIDEO_SET_STREAMTYPE

Attention: This ioctl is deprecated.

Synopsis

VIDEO_SET_STREAMTYPE

`int ioctl(fd, VIDEO_SET_STREAMTYPE, int type)`

Arguments

int fd	File descriptor returned by a previous call to open().
int request	Equals VIDEO_SET_STREAMTYPE for this command.
int type	stream type

Description

This ioctl tells the driver which kind of stream to expect being written to it. If this call is not used the default of video PES is used. Some drivers might not support this call and always expect PES.

Return Value

On success 0 is returned, on error -1 and the `errno` variable is set appropriately. The generic error codes are described at the *Generic Error Codes* chapter.

VIDEO_SET_FORMAT

Name

VIDEO_SET_FORMAT

Attention: This ioctl is deprecated.

Synopsis

VIDEO_SET_FORMAT

`int ioctl(fd, VIDEO_SET_FORMAT, video_format_t format)`

Arguments

int fd	File descriptor returned by a previous call to open().
int request	Equals VIDEO_SET_FORMAT for this command.
video_format_t format	video format of TV as defined in section ??.

Description

This ioctl sets the screen format (aspect ratio) of the connected output device (TV) so that the output of the decoder can be adjusted accordingly.

type **video_format_t**

```
typedef enum {
        VIDEO_FORMAT_4_3,     /* Select 4:3 format */
        VIDEO_FORMAT_16_9,    /* Select 16:9 format. */
        VIDEO_FORMAT_221_1    /* 2.21:1 */
} video_format_t;
```

Return Value

On success 0 is returned, on error -1 and the `errno` variable is set appropriately. The generic error codes are described at the *Generic Error Codes* chapter.

`EINVAL`	format is not a valid video format.

Digital TV Audio Device

The Digital TV audio device controls the MPEG2 audio decoder of the Digital TV hardware. It can be accessed through `/dev/dvb/adapter?/audio?`. Data types and and ioctl definitions can be accessed by including `linux/dvb/audio.h` in your application.

Please note that some Digital TV cards don't have their own MPEG decoder, which results in the omission of the audio and video device.

These ioctls were also used by V4L2 to control MPEG decoders implemented in V4L2. The use of these ioctls for that purpose has been made obsolete and proper V4L2 ioctls or controls have been created to replace that functionality.

Audio Data Types

This section describes the structures, data types and defines used when talking to the audio device.

type **audio_stream_source**

The audio stream source is set through the AUDIO_SELECT_SOURCE call and can take the following values, depending on whether we are replaying from an internal (demux) or external (user write) source.

```
typedef enum {
    AUDIO_SOURCE_DEMUX,
    AUDIO_SOURCE_MEMORY
} audio_stream_source_t;
```

AUDIO_SOURCE_DEMUX selects the demultiplexer (fed either by the frontend or the DVR device) as the source of the video stream. If AUDIO_SOURCE_MEMORY is selected the stream comes from the application through the `write()` system call.

type **audio_play_state**

The following values can be returned by the AUDIO_GET_STATUS call representing the state of audio playback.

```
typedef enum {
    AUDIO_STOPPED,
    AUDIO_PLAYING,
    AUDIO_PAUSED
} audio_play_state_t;
```

type **audio_channel_select**

The audio channel selected via AUDIO_CHANNEL_SELECT is determined by the following values.

```
typedef enum {
    AUDIO_STEREO,
    AUDIO_MONO_LEFT,
    AUDIO_MONO_RIGHT,
    AUDIO_MONO,
    AUDIO_STEREO_SWAPPED
} audio_channel_select_t;
```

type **audio_status**

The AUDIO_GET_STATUS call returns the following structure informing about various states of the playback operation.

```
typedef struct audio_status {
    boolean AV_sync_state;
    boolean mute_state;
```

(continues on next page)

(continued from previous page)

```
    audio_play_state_t play_state;
    audio_stream_source_t stream_source;
    audio_channel_select_t channel_select;
    boolean bypass_mode;
    audio_mixer_t mixer_state;
} audio_status_t;
```

type **audio_mixer**

The following structure is used by the AUDIO_SET_MIXER call to set the audio volume.

```
typedef struct audio_mixer {
    unsigned int volume_left;
    unsigned int volume_right;
} audio_mixer_t;
```

audio encodings

A call to AUDIO_GET_CAPABILITIES returns an unsigned integer with the following bits set according to the hardwares capabilities.

```
#define AUDIO_CAP_DTS    1
#define AUDIO_CAP_LPCM   2
#define AUDIO_CAP_MP1    4
#define AUDIO_CAP_MP2    8
#define AUDIO_CAP_MP3   16
#define AUDIO_CAP_AAC   32
#define AUDIO_CAP_OGG   64
#define AUDIO_CAP_SDDS 128
#define AUDIO_CAP_AC3  256
```

Audio Function Calls

Digital TV audio open()

Name

Digital TV audio open()

> **Attention:** This ioctl is deprecated

Synopsis

int **open**(const char *deviceName, int flags)

Arguments

const char *deviceName	Name of specific audio device.
int flags	A bit-wise OR of the following flags:
	O_RDONLY read-only access
	O_RDWR read/write access
	O_NONBLOCK open in non-blocking mode
	(blocking mode is the default)

Description

This system call opens a named audio device (e.g. /dev/dvb/adapter0/audio0) for subsequent use. When an open() call has succeeded, the device will be ready for use. The significance of blocking or non-blocking mode is described in the documentation for functions where there is a difference. It does not affect the semantics of the open() call itself. A device opened in blocking mode can later be put into non-blocking mode (and vice versa) using the F_SETFL command of the fcntl system call. This is a standard system call, documented in the Linux manual page for fcntl. Only one user can open the Audio Device in O_RDWR mode. All other attempts to open the device in this mode will fail, and an error code will be returned. If the Audio Device is opened in O_RDONLY mode, the only ioctl call that can be used is AUDIO_GET_STATUS. All other call will return with an error code.

Return Value

`ENODEV`	Device driver not loaded/available.
`EBUSY`	Device or resource busy.
`EINVAL`	Invalid argument.

Digital TV audio close()

Name

Digital TV audio close()

> **Attention:** This ioctl is deprecated

Synopsis

int **close**(int fd)

Arguments

int fd	File descriptor returned by a previous call to open().

Description

This system call closes a previously opened audio device.

Return Value

EBADF	fd is not a valid open file descriptor.

Digital TV audio write()

Name

Digital TV audio write()

Attention: This ioctl is deprecated

Synopsis

size_t **write**(int fd, const void *buf, size_t count)

Arguments

int fd	File descriptor returned by a previous call to open().
void *buf	Pointer to the buffer containing the PES data.
size_t count	Size of buf.

Description

This system call can only be used if AUDIO_SOURCE_MEMORY is selected in the ioctl call AUDIO_SELECT_SOURCE. The data provided shall be in PES format. If O_NONBLOCK is not specified the function will block until buffer space is available. The amount of data to be transferred is implied by count.

Return Value

`EPERM`	Mode AUDIO_SOURCE_MEMORY not selected.
`ENOMEM`	Attempted to write more data than the internal buffer can hold.
`EBADF`	fd is not a valid open file descriptor.

AUDIO_STOP

Name

AUDIO_STOP

Attention: This ioctl is deprecated

Synopsis

`AUDIO_STOP`

`int ioctl(int fd, AUDIO_STOP)`

Arguments

int fd	File descriptor returned by a previous call to open().

Description

This ioctl call asks the Audio Device to stop playing the current stream.

Return Value

On success 0 is returned, on error -1 and the `errno` variable is set appropriately. The generic error codes are described at the *Generic Error Codes* chapter.

AUDIO_PLAY

Name

AUDIO_PLAY

Attention: This ioctl is deprecated

Synopsis

AUDIO_PLAY

`int ioctl(int fd, AUDIO_PLAY)`

Arguments

int fd	File descriptor returned by a previous call to open().

Description

This ioctl call asks the Audio Device to start playing an audio stream from the selected source.

Return Value

On success 0 is returned, on error -1 and the `errno` variable is set appropriately. The generic error codes are described at the *Generic Error Codes* chapter.

AUDIO_PAUSE

Name

AUDIO_PAUSE

Attention: This ioctl is deprecated

Synopsis

AUDIO_PAUSE

`int ioctl(int fd, AUDIO_PAUSE)`

Arguments

int	fd	File descriptor returned by a previous call to open().

Description

This ioctl call suspends the audio stream being played. Decoding and playing are paused. It is then possible to restart again decoding and playing process of the audio stream using AUDIO_CONTINUE command.

Return Value

On success 0 is returned, on error -1 and the `errno` variable is set appropriately. The generic error codes are described at the *Generic Error Codes* chapter.

AUDIO_CONTINUE

Name

AUDIO_CONTINUE

Attention: This ioctl is deprecated

Synopsis

AUDIO_CONTINUE

`int ioctl(int fd, AUDIO_CONTINUE)`

Arguments

int fd	File descriptor returned by a previous call to open().

Description

This ioctl restarts the decoding and playing process previously paused with AUDIO_PAUSE command.

Return Value

On success 0 is returned, on error -1 and the `errno` variable is set appropriately. The generic error codes are described at the *Generic Error Codes* chapter.

AUDIO_SELECT_SOURCE

Name

AUDIO_SELECT_SOURCE

Attention: This ioctl is deprecated

Synopsis

AUDIO_SELECT_SOURCE

`int ioctl(int fd, AUDIO_SELECT_SOURCE, struct audio_stream_source *source)`

Arguments

int fd	File descriptor returned by a previous call to open().
audio_stream_source_t source	Indicates the source that shall be used for the Audio stream.

Description

This ioctl call informs the audio device which source shall be used for the input data. The possible sources are demux or memory. If AUDIO_SOURCE_MEMORY is selected, the data is fed to the Audio Device through the write command.

Return Value

On success 0 is returned, on error -1 and the `errno` variable is set appropriately. The generic error codes are described at the *Generic Error Codes* chapter.

AUDIO_SET_MUTE

Name

AUDIO_SET_MUTE

Attention: This ioctl is deprecated

Synopsis

AUDIO_SET_MUTE

`int ioctl(int fd, AUDIO_SET_MUTE, boolean state)`

Arguments

int fd	File descriptor returned by a previous call to open().
boolean state	Indicates if audio device shall mute or not. TRUE: Audio Mute FALSE: Audio Un-mute

Description

This ioctl is for Digital TV devices only. To control a V4L2 decoder use the V4L2 *ioctl VIDIOC_DECODER_CMD, VIDIOC_TRY_DECODER_CMD* with the `V4L2_DEC_CMD_START_MUTE_AUDIO` flag instead.

This ioctl call asks the audio device to mute the stream that is currently being played.

Return Value

On success 0 is returned, on error -1 and the `errno` variable is set appropriately. The generic error codes are described at the *Generic Error Codes* chapter.

AUDIO_SET_AV_SYNC

Name

AUDIO_SET_AV_SYNC

> **Attention:** This ioctl is deprecated

Synopsis

AUDIO_SET_AV_SYNC

`int ioctl(int fd, AUDIO_SET_AV_SYNC, boolean state)`

Arguments

int fd	File descriptor returned by a previous call to open().
boolean state	Tells the Digital TV subsystem if A/V synchronization shall be ON or OFF. TRUE: AV-sync ON FALSE: AV-sync OFF

Description

This ioctl call asks the Audio Device to turn ON or OFF A/V synchronization.

Return Value

On success 0 is returned, on error -1 and the `errno` variable is set appropriately. The generic error codes are described at the *Generic Error Codes* chapter.

AUDIO_SET_BYPASS_MODE

Name

AUDIO_SET_BYPASS_MODE

> **Attention:** This ioctl is deprecated

Synopsis

AUDIO_SET_BYPASS_MODE

```
int ioctl(int fd, AUDIO_SET_BYPASS_MODE, boolean mode)
```

Arguments

int fd	File descriptor returned by a previous call to open().
boolean mode	Enables or disables the decoding of the current Audio stream in the Digital TV subsystem. TRUE: Bypass is disabled FALSE: Bypass is enabled

Description

This ioctl call asks the Audio Device to bypass the Audio decoder and forward the stream without decoding. This mode shall be used if streams that can't be handled by the Digital TV system shall be decoded. Dolby DigitalTM streams are automatically forwarded by the Digital TV subsystem if the hardware can handle it.

Return Value

On success 0 is returned, on error -1 and the `errno` variable is set appropriately. The generic error codes are described at the *Generic Error Codes* chapter.

AUDIO_CHANNEL_SELECT

Name

AUDIO_CHANNEL_SELECT

> **Attention:** This ioctl is deprecated

Synopsis

AUDIO_CHANNEL_SELECT

`int ioctl(int fd, AUDIO_CHANNEL_SELECT, struct audio_channel_select *select)`

Arguments

int fd	File descriptor returned by a previous call to open().
audio_channel_select_t ch	Select the output format of the audio (mono left/right, stereo).

Description

This ioctl is for Digital TV devices only. To control a V4L2 decoder use the V4L2 `V4L2_CID_MPEG_AUDIO_DEC_PLAYBACK` control instead.

This ioctl call asks the Audio Device to select the requested channel if possible.

Return Value

On success 0 is returned, on error -1 and the `errno` variable is set appropriately. The generic error codes are described at the *Generic Error Codes* chapter.

AUDIO_BILINGUAL_CHANNEL_SELECT

Name

AUDIO_BILINGUAL_CHANNEL_SELECT

> **Attention:** This ioctl is deprecated

Synopsis

AUDIO_BILINGUAL_CHANNEL_SELECT

```
int ioctl(int fd, AUDIO_BILINGUAL_CHANNEL_SELECT, struct
audio_channel_select *select)
```

Arguments

int fd	File descriptor returned by a previous call to open().
audio_channel_select_t ch	Select the output format of the audio (mono left/right, stereo).

Description

This ioctl is obsolete. Do not use in new drivers. It has been replaced by the V4L2 `V4L2_CID_MPEG_AUDIO_DEC_MULTILINGUAL_PLAYBACK` control for MPEG decoders controlled through V4L2.

This ioctl call asks the Audio Device to select the requested channel for bilingual streams if possible.

Return Value

On success 0 is returned, on error -1 and the `errno` variable is set appropriately. The generic error codes are described at the *Generic Error Codes* chapter.

AUDIO_GET_STATUS

Name

AUDIO_GET_STATUS

Attention: This ioctl is deprecated

Synopsis

AUDIO_GET_STATUS

```
int ioctl(int fd, AUDIO_GET_STATUS, struct audio_status *status)
```

Arguments

int fd	File descriptor returned by a previous call to open().
struct audio_status *status	Returns the current state of Audio Device.

Description

This ioctl call asks the Audio Device to return the current state of the Audio Device.

Return Value

On success 0 is returned, on error -1 and the `errno` variable is set appropriately. The generic error codes are described at the *Generic Error Codes* chapter.

AUDIO_GET_CAPABILITIES

Name

AUDIO_GET_CAPABILITIES

Attention: This ioctl is deprecated

Synopsis

AUDIO_GET_CAPABILITIES

`int ioctl(int fd, AUDIO_GET_CAPABILITIES, unsigned int *cap)`

Arguments

int fd	File descriptor returned by a previous call to open().
unsigned int *cap	Returns a bit array of supported sound formats.

Description

This ioctl call asks the Audio Device to tell us about the decoding capabilities of the audio hardware.

Return Value

On success 0 is returned, on error -1 and the `errno` variable is set appropriately. The generic error codes are described at the *Generic Error Codes* chapter.

AUDIO_CLEAR_BUFFER

Name

AUDIO_CLEAR_BUFFER

> **Attention:** This ioctl is deprecated

Synopsis

`AUDIO_CLEAR_BUFFER`

`int ioctl(int fd, AUDIO_CLEAR_BUFFER)`

Arguments

int fd	File descriptor returned by a previous call to open().

Description

This ioctl call asks the Audio Device to clear all software and hardware buffers of the audio decoder device.

Return Value

On success 0 is returned, on error -1 and the `errno` variable is set appropriately. The generic error codes are described at the *Generic Error Codes* chapter.

AUDIO_SET_ID

Name

AUDIO_SET_ID

Attention: This ioctl is deprecated

Synopsis

AUDIO_SET_ID

`int ioctl(int fd, AUDIO_SET_ID, int id)`

Arguments

int fd	File descriptor returned by a previous call to open().
int id	audio sub-stream id

Description

This ioctl selects which sub-stream is to be decoded if a program or system stream is sent to the video device. If no audio stream type is set the id has to be in [0xC0,0xDF] for MPEG sound, in [0x80,0x87] for AC3 and in [0xA0,0xA7] for LPCM. More specifications may follow for other stream types. If the stream type is set the id just specifies the substream id of the audio stream and only the first 5 bits are recognized.

Return Value

On success 0 is returned, on error -1 and the `errno` variable is set appropriately. The generic error codes are described at the *Generic Error Codes* chapter.

AUDIO_SET_MIXER

Name

AUDIO_SET_MIXER

Attention: This ioctl is deprecated

Synopsis

AUDIO_SET_MIXER

`int ioctl(int fd, AUDIO_SET_MIXER, struct audio_mixer *mix)`

Arguments

int fd	File descriptor returned by a previous call to open().
audio_mixer_t *mix	mixer settings.

Description

This ioctl lets you adjust the mixer settings of the audio decoder.

Return Value

On success 0 is returned, on error -1 and the `errno` variable is set appropriately. The generic error codes are described at the *Generic Error Codes* chapter.

AUDIO_SET_STREAMTYPE

Name

AUDIO_SET_STREAMTYPE

Attention: This ioctl is deprecated

Synopsis

AUDIO_SET_STREAMTYPE

`int ioctl(fd, AUDIO_SET_STREAMTYPE, int type)`

Arguments

int fd	File descriptor returned by a previous call to open().
int type	stream type

Description

This ioctl tells the driver which kind of audio stream to expect. This is useful if the stream offers several audio sub-streams like LPCM and AC3.

Return Value

On success 0 is returned, on error -1 and the `errno` variable is set appropriately. The generic error codes are described at the *Generic Error Codes* chapter.

`EINVAL`	type is not a valid or supported stream type.

8.3.7 Examples

In the past, we used to have a set of examples here. However, those examples got out of date and doesn't even compile nowadays.

Also, nowadays, the best is to use the libdvbv5 DVB API nowadays, with is fully documented.

Please refer to the libdvbv5 for updated/recommended examples.

8.3.8 Digital TV uAPI header files

Digital TV uAPI headers

frontend.h

```
/* SPDX-License-Identifier: LGPL-2.1+ WITH Linux-syscall-note */
/*
 * frontend.h
 *
 * Copyright (C) 2000 Marcus Metzler <marcus@convergence.de>
 *                    Ralph  Metzler <ralph@convergence.de>
 *                    Holger Waechtler <holger@convergence.de>
 *                    Andre Draszik <ad@convergence.de>
 *                    for convergence integrated media GmbH
 *
 * This program is free software; you can redistribute it and/or
 * modify it under the terms of the GNU Lesser General Public␣
↪License
```

```
 * as published by the Free Software Foundation; either version 2.1
 * of the License, or (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU Lesser General Public␣
↪License
 * along with this program; if not, write to the Free Software
 * Foundation, Inc., 59 Temple Place - Suite 330, Boston, MA ␣
↪02111-1307, USA.
 *
 */

#ifndef _DVBFRONTEND_H_
#define _DVBFRONTEND_H_

#include <linux/types.h>

/**
 * enum fe_caps - Frontend capabilities
 *
 * @FE_IS_STUPID:                        There's something wrong at␣
↪the
 *                                       frontend, and it can't␣
↪report its
 *                                       capabilities.
 * @FE_CAN_INVERSION_AUTO:               Can auto-detect frequency␣
↪spectral
 *                                       band inversion
 * @FE_CAN_FEC_1_2:                      Supports FEC 1/2
 * @FE_CAN_FEC_2_3:                      Supports FEC 2/3
 * @FE_CAN_FEC_3_4:                      Supports FEC 3/4
 * @FE_CAN_FEC_4_5:                      Supports FEC 4/5
 * @FE_CAN_FEC_5_6:                      Supports FEC 5/6
 * @FE_CAN_FEC_6_7:                      Supports FEC 6/7
 * @FE_CAN_FEC_7_8:                      Supports FEC 7/8
 * @FE_CAN_FEC_8_9:                      Supports FEC 8/9
 * @FE_CAN_FEC_AUTO:                     Can auto-detect FEC
 * @FE_CAN_QPSK:                         Supports QPSK modulation
 * @FE_CAN_QAM_16:                       Supports 16-QAM modulation
 * @FE_CAN_QAM_32:                       Supports 32-QAM modulation
 * @FE_CAN_QAM_64:                       Supports 64-QAM modulation
 * @FE_CAN_QAM_128:                      Supports 128-QAM modulation
 * @FE_CAN_QAM_256:                      Supports 256-QAM modulation
 * @FE_CAN_QAM_AUTO:                     Can auto-detect QAM␣
↪modulation
 * @FE_CAN_TRANSMISSION_MODE_AUTO:       Can auto-detect␣
↪transmission mode
```

```
 * @FE_CAN_BANDWIDTH_AUTO:                  Can auto-detect bandwidth
 * @FE_CAN_GUARD_INTERVAL_AUTO:             Can auto-detect guard␣
↪interval
 * @FE_CAN_HIERARCHY_AUTO:                  Can auto-detect hierarchy
 * @FE_CAN_8VSB:                            Supports 8-VSB modulation
 * @FE_CAN_16VSB:                           Supporta 16-VSB modulation
 * @FE_HAS_EXTENDED_CAPS:                   Unused
 * @FE_CAN_MULTISTREAM:                     Supports multistream␣
↪filtering
 * @FE_CAN_TURBO_FEC:                       Supports "turbo FEC"␣
↪modulation
 * @FE_CAN_2G_MODULATION:                   Supports "2nd generation"␣
↪modulation,
 *                                          e. g. DVB-S2, DVB-T2, DVB-C2
 * @FE_NEEDS_BENDING:                       Unused
 * @FE_CAN_RECOVER:                         Can recover from a cable␣
↪unplug
 *                                          automatically
 * @FE_CAN_MUTE_TS:                         Can stop spurious TS data␣
↪output
 */
enum fe_caps {
        FE_IS_STUPID                        = 0,
        FE_CAN_INVERSION_AUTO               = 0x1,
        FE_CAN_FEC_1_2                      = 0x2,
        FE_CAN_FEC_2_3                      = 0x4,
        FE_CAN_FEC_3_4                      = 0x8,
        FE_CAN_FEC_4_5                      = 0x10,
        FE_CAN_FEC_5_6                      = 0x20,
        FE_CAN_FEC_6_7                      = 0x40,
        FE_CAN_FEC_7_8                      = 0x80,
        FE_CAN_FEC_8_9                      = 0x100,
        FE_CAN_FEC_AUTO                     = 0x200,
        FE_CAN_QPSK                         = 0x400,
        FE_CAN_QAM_16                       = 0x800,
        FE_CAN_QAM_32                       = 0x1000,
        FE_CAN_QAM_64                       = 0x2000,
        FE_CAN_QAM_128                      = 0x4000,
        FE_CAN_QAM_256                      = 0x8000,
        FE_CAN_QAM_AUTO                     = 0x10000,
        FE_CAN_TRANSMISSION_MODE_AUTO       = 0x20000,
        FE_CAN_BANDWIDTH_AUTO               = 0x40000,
        FE_CAN_GUARD_INTERVAL_AUTO          = 0x80000,
        FE_CAN_HIERARCHY_AUTO               = 0x100000,
        FE_CAN_8VSB                         = 0x200000,
        FE_CAN_16VSB                        = 0x400000,
        FE_HAS_EXTENDED_CAPS                = 0x800000,
        FE_CAN_MULTISTREAM                  = 0x4000000,
        FE_CAN_TURBO_FEC                    = 0x8000000,
        FE_CAN_2G_MODULATION                = 0x10000000,
        FE_NEEDS_BENDING                    = 0x20000000,
```

```
        FE_CAN_RECOVER                  = 0x40000000,
        FE_CAN_MUTE_TS                  = 0x80000000
};

/*
 * DEPRECATED: Should be kept just due to backward compatibility.
 */
enum fe_type {
        FE_QPSK,
        FE_QAM,
        FE_OFDM,
        FE_ATSC
};

/**
 * struct dvb_frontend_info - Frontend properties and capabilities
 *
 * @name:                      Name of the frontend
 * @type:                      ****DEPRECATED****.
 *                             Should not be used on modern␣
↪programs,
 *                             as a frontend may have more than␣
↪one type.
 *                             In order to get the support types␣
↪of a given
 *                             frontend, use :c:type:`DTV_ENUM_
↪DELSYS`
 *                             instead.
 * @frequency_min:             Minimal frequency supported by the␣
↪frontend.
 * @frequency_max:             Minimal frequency supported by the␣
↪frontend.
 * @frequency_stepsize:        All frequencies are multiple of␣
↪this value.
 * @frequency_tolerance:       Frequency tolerance.
 * @symbol_rate_min:           Minimal symbol rate, in bauds
 *                             (for Cable/Satellite systems).
 * @symbol_rate_max:           Maximal symbol rate, in bauds
 *                             (for Cable/Satellite systems).
 * @symbol_rate_tolerance:     Maximal symbol rate tolerance, in␣
↪ppm
 *                             (for Cable/Satellite systems).
 * @notifier_delay:            ****DEPRECATED****. Not used by any␣
↪driver.
 * @caps:                      Capabilities supported by the␣
↪frontend,
 *                             as specified in &enum fe_caps.
 *
 * .. note:
 *
 *    #. The frequencies are specified in Hz for Terrestrial and␣
```

```
↪Cable
 *       systems.
 *    #. The frequencies are specified in kHz for Satellite systems.
 */
struct dvb_frontend_info {
        char       name[128];
        enum fe_type type;      /* DEPRECATED. Use DTV_ENUM_DELSYS␣
↪instead */
        __u32      frequency_min;
        __u32      frequency_max;
        __u32      frequency_stepsize;
        __u32      frequency_tolerance;
        __u32      symbol_rate_min;
        __u32      symbol_rate_max;
        __u32      symbol_rate_tolerance;
        __u32      notifier_delay;              /* DEPRECATED */
        enum fe_caps caps;
};

/**
 * struct dvb_diseqc_master_cmd - DiSEqC master command
 *
 * @msg:
 *      DiSEqC message to be sent. It contains a 3 bytes header␣
↪with:
 *      framing + address + command, and an optional argument
 *      of up to 3 bytes of data.
 * @msg_len:
 *      Length of the DiSEqC message. Valid values are 3 to 6.
 *
 * Check out the DiSEqC bus spec available on http://www.eutelsat.
↪org/ for
 * the possible messages that can be used.
 */
struct dvb_diseqc_master_cmd {
        __u8 msg[6];
        __u8 msg_len;
};

/**
 * struct dvb_diseqc_slave_reply - DiSEqC received data
 *
 * @msg:
 *      DiSEqC message buffer to store a message received via␣
↪DiSEqC.
 *      It contains one byte header with: framing and
 *      an optional argument of up to 3 bytes of data.
 * @msg_len:
 *      Length of the DiSEqC message. Valid values are 0 to 4,
 *      where 0 means no message.
 * @timeout:
```

```
 *      Return from ioctl after timeout ms with errorcode when
 *      no message was received.
 *
 * Check out the DiSEqC bus spec available on http://www.eutelsat.
↪org/ for
 * the possible messages that can be used.
 */
struct dvb_diseqc_slave_reply {
        __u8 msg[4];
        __u8 msg_len;
        int  timeout;
};

/**
 * enum fe_sec_voltage - DC Voltage used to feed the LNBf
 *
 * @SEC_VOLTAGE_13:     Output 13V to the LNBf
 * @SEC_VOLTAGE_18:     Output 18V to the LNBf
 * @SEC_VOLTAGE_OFF:    Don't feed the LNBf with a DC voltage
 */
enum fe_sec_voltage {
        SEC_VOLTAGE_13,
        SEC_VOLTAGE_18,
        SEC_VOLTAGE_OFF
};

/**
 * enum fe_sec_tone_mode - Type of tone to be send to the LNBf.
 * @SEC_TONE_ON:        Sends a 22kHz tone burst to the antenna.
 * @SEC_TONE_OFF:       Don't send a 22kHz tone to the antenna␣
↪(except
 *                      if the ``FE_DISEQC_*`` ioctls are called).
 */
enum fe_sec_tone_mode {
        SEC_TONE_ON,
        SEC_TONE_OFF
};

/**
 * enum fe_sec_mini_cmd - Type of mini burst to be sent
 *
 * @SEC_MINI_A:         Sends a mini-DiSEqC 22kHz '0' Tone Burst to␣
↪select
 *                      satellite-A
 * @SEC_MINI_B:         Sends a mini-DiSEqC 22kHz '1' Data Burst to␣
↪select
 *                      satellite-B
 */
enum fe_sec_mini_cmd {
        SEC_MINI_A,
        SEC_MINI_B
```

```
};

/**
 * enum fe_status - Enumerates the possible frontend status.
 * @FE_NONE:           The frontend doesn't have any kind of lock.
 *                     That's the initial frontend status
 * @FE_HAS_SIGNAL:     Has found something above the noise level.
 * @FE_HAS_CARRIER:    Has found a signal.
 * @FE_HAS_VITERBI:    FEC inner coding (Viterbi, LDPC or other␣
↪inner code).
 *                     is stable.
 * @FE_HAS_SYNC:       Synchronization bytes was found.
 * @FE_HAS_LOCK:       Digital TV were locked and everything is␣
↪working.
 * @FE_TIMEDOUT:       Fo lock within the last about 2 seconds.
 * @FE_REINIT:         Frontend was reinitialized, application is␣
↪recommended
 *                     to reset DiSEqC, tone and parameters.
 */
enum fe_status {
        FE_NONE                 = 0x00,
        FE_HAS_SIGNAL           = 0x01,
        FE_HAS_CARRIER          = 0x02,
        FE_HAS_VITERBI          = 0x04,
        FE_HAS_SYNC             = 0x08,
        FE_HAS_LOCK             = 0x10,
        FE_TIMEDOUT             = 0x20,
        FE_REINIT               = 0x40,
};

/**
 * enum fe_spectral_inversion - Type of inversion band
 *
 * @INVERSION_OFF:     Don't do spectral band inversion.
 * @INVERSION_ON:      Do spectral band inversion.
 * @INVERSION_AUTO:    Autodetect spectral band inversion.
 *
 * This parameter indicates if spectral inversion should be␣
↪presumed or
 * not. In the automatic setting (``INVERSION_AUTO``) the hardware␣
↪will try
 * to figure out the correct setting by itself. If the hardware␣
↪doesn't
 * support, the %dvb_frontend will try to lock at the carrier first␣
↪with
 * inversion off. If it fails, it will try to enable inversion.
 */
enum fe_spectral_inversion {
        INVERSION_OFF,
        INVERSION_ON,
        INVERSION_AUTO
```

```
};

/**
 * enum fe_code_rate - Type of Forward Error Correction (FEC)
 *
 *
 * @FEC_NONE: No Forward Error Correction Code
 * @FEC_1_2:  Forward Error Correction Code 1/2
 * @FEC_2_3:  Forward Error Correction Code 2/3
 * @FEC_3_4:  Forward Error Correction Code 3/4
 * @FEC_4_5:  Forward Error Correction Code 4/5
 * @FEC_5_6:  Forward Error Correction Code 5/6
 * @FEC_6_7:  Forward Error Correction Code 6/7
 * @FEC_7_8:  Forward Error Correction Code 7/8
 * @FEC_8_9:  Forward Error Correction Code 8/9
 * @FEC_AUTO: Autodetect Error Correction Code
 * @FEC_3_5:  Forward Error Correction Code 3/5
 * @FEC_9_10: Forward Error Correction Code 9/10
 * @FEC_2_5:  Forward Error Correction Code 2/5
 *
 * Please note that not all FEC types are supported by a given␣
↪standard.
 */
enum fe_code_rate {
        FEC_NONE = 0,
        FEC_1_2,
        FEC_2_3,
        FEC_3_4,
        FEC_4_5,
        FEC_5_6,
        FEC_6_7,
        FEC_7_8,
        FEC_8_9,
        FEC_AUTO,
        FEC_3_5,
        FEC_9_10,
        FEC_2_5,
};

/**
 * enum fe_modulation - Type of modulation/constellation
 * @QPSK:       QPSK modulation
 * @QAM_16:     16-QAM modulation
 * @QAM_32:     32-QAM modulation
 * @QAM_64:     64-QAM modulation
 * @QAM_128:    128-QAM modulation
 * @QAM_256:    256-QAM modulation
 * @QAM_AUTO:   Autodetect QAM modulation
 * @VSB_8:      8-VSB modulation
 * @VSB_16:     16-VSB modulation
 * @PSK_8:      8-PSK modulation
```

```
 * @APSK_16:     16-APSK modulation
 * @APSK_32:     32-APSK modulation
 * @DQPSK:       DQPSK modulation
 * @QAM_4_NR:    4-QAM-NR modulation
 *
 * Please note that not all modulations are supported by a given␣
↪standard.
 *
 */
enum fe_modulation {
        QPSK,
        QAM_16,
        QAM_32,
        QAM_64,
        QAM_128,
        QAM_256,
        QAM_AUTO,
        VSB_8,
        VSB_16,
        PSK_8,
        APSK_16,
        APSK_32,
        DQPSK,
        QAM_4_NR,
};

/**
 * enum fe_transmit_mode - Transmission mode
 *
 * @TRANSMISSION_MODE_AUTO:
 *      Autodetect transmission mode. The hardware will try to find␣
↪the
 *      correct FFT-size (if capable) to fill in the missing␣
↪parameters.
 * @TRANSMISSION_MODE_1K:
 *      Transmission mode 1K
 * @TRANSMISSION_MODE_2K:
 *      Transmission mode 2K
 * @TRANSMISSION_MODE_8K:
 *      Transmission mode 8K
 * @TRANSMISSION_MODE_4K:
 *      Transmission mode 4K
 * @TRANSMISSION_MODE_16K:
 *      Transmission mode 16K
 * @TRANSMISSION_MODE_32K:
 *      Transmission mode 32K
 * @TRANSMISSION_MODE_C1:
 *      Single Carrier (C=1) transmission mode (DTMB only)
 * @TRANSMISSION_MODE_C3780:
 *      Multi Carrier (C=3780) transmission mode (DTMB only)
 *
```

```
 * Please note that not all transmission modes are supported by a␣
↪given
 * standard.
 */
enum fe_transmit_mode {
        TRANSMISSION_MODE_2K,
        TRANSMISSION_MODE_8K,
        TRANSMISSION_MODE_AUTO,
        TRANSMISSION_MODE_4K,
        TRANSMISSION_MODE_1K,
        TRANSMISSION_MODE_16K,
        TRANSMISSION_MODE_32K,
        TRANSMISSION_MODE_C1,
        TRANSMISSION_MODE_C3780,
};

/**
 * enum fe_guard_interval - Guard interval
 *
 * @GUARD_INTERVAL_AUTO:        Autodetect the guard interval
 * @GUARD_INTERVAL_1_128:       Guard interval 1/128
 * @GUARD_INTERVAL_1_32:        Guard interval 1/32
 * @GUARD_INTERVAL_1_16:        Guard interval 1/16
 * @GUARD_INTERVAL_1_8:         Guard interval 1/8
 * @GUARD_INTERVAL_1_4:         Guard interval 1/4
 * @GUARD_INTERVAL_19_128:      Guard interval 19/128
 * @GUARD_INTERVAL_19_256:      Guard interval 19/256
 * @GUARD_INTERVAL_PN420:       PN length 420 (1/4)
 * @GUARD_INTERVAL_PN595:       PN length 595 (1/6)
 * @GUARD_INTERVAL_PN945:       PN length 945 (1/9)
 *
 * Please note that not all guard intervals are supported by a␣
↪given standard.
 */
enum fe_guard_interval {
        GUARD_INTERVAL_1_32,
        GUARD_INTERVAL_1_16,
        GUARD_INTERVAL_1_8,
        GUARD_INTERVAL_1_4,
        GUARD_INTERVAL_AUTO,
        GUARD_INTERVAL_1_128,
        GUARD_INTERVAL_19_128,
        GUARD_INTERVAL_19_256,
        GUARD_INTERVAL_PN420,
        GUARD_INTERVAL_PN595,
        GUARD_INTERVAL_PN945,
};

/**
 * enum fe_hierarchy - Hierarchy
 * @HIERARCHY_NONE:     No hierarchy
```

```
 * @HIERARCHY_AUTO:     Autodetect hierarchy (if supported)
 * @HIERARCHY_1:        Hierarchy 1
 * @HIERARCHY_2:        Hierarchy 2
 * @HIERARCHY_4:        Hierarchy 4
 *
 * Please note that not all hierarchy types are supported by a␣
↪given standard.
 */
enum fe_hierarchy {
        HIERARCHY_NONE,
        HIERARCHY_1,
        HIERARCHY_2,
        HIERARCHY_4,
        HIERARCHY_AUTO
};

/**
 * enum fe_interleaving - Interleaving
 * @INTERLEAVING_NONE:  No interleaving.
 * @INTERLEAVING_AUTO:  Auto-detect interleaving.
 * @INTERLEAVING_240:   Interleaving of 240 symbols.
 * @INTERLEAVING_720:   Interleaving of 720 symbols.
 *
 * Please note that, currently, only DTMB uses it.
 */
enum fe_interleaving {
        INTERLEAVING_NONE,
        INTERLEAVING_AUTO,
        INTERLEAVING_240,
        INTERLEAVING_720,
};

/* DVBv5 property Commands */

#define DTV_UNDEFINED           0
#define DTV_TUNE                1
#define DTV_CLEAR               2
#define DTV_FREQUENCY           3
#define DTV_MODULATION          4
#define DTV_BANDWIDTH_HZ        5
#define DTV_INVERSION           6
#define DTV_DISEQC_MASTER       7
#define DTV_SYMBOL_RATE         8
#define DTV_INNER_FEC           9
#define DTV_VOLTAGE             10
#define DTV_TONE                11
#define DTV_PILOT               12
#define DTV_ROLLOFF             13
#define DTV_DISEQC_SLAVE_REPLY  14

/* Basic enumeration set for querying unlimited capabilities */
```

```
#define DTV_FE_CAPABILITY_COUNT 15
#define DTV_FE_CAPABILITY       16
#define DTV_DELIVERY_SYSTEM     17

/* ISDB-T and ISDB-Tsb */
#define DTV_ISDBT_PARTIAL_RECEPTION     18
#define DTV_ISDBT_SOUND_BROADCASTING    19

#define DTV_ISDBT_SB_SUBCHANNEL_ID      20
#define DTV_ISDBT_SB_SEGMENT_IDX        21
#define DTV_ISDBT_SB_SEGMENT_COUNT      22

#define DTV_ISDBT_LAYERA_FEC                    23
#define DTV_ISDBT_LAYERA_MODULATION             24
#define DTV_ISDBT_LAYERA_SEGMENT_COUNT          25
#define DTV_ISDBT_LAYERA_TIME_INTERLEAVING      26

#define DTV_ISDBT_LAYERB_FEC                    27
#define DTV_ISDBT_LAYERB_MODULATION             28
#define DTV_ISDBT_LAYERB_SEGMENT_COUNT          29
#define DTV_ISDBT_LAYERB_TIME_INTERLEAVING      30

#define DTV_ISDBT_LAYERC_FEC                    31
#define DTV_ISDBT_LAYERC_MODULATION             32
#define DTV_ISDBT_LAYERC_SEGMENT_COUNT          33
#define DTV_ISDBT_LAYERC_TIME_INTERLEAVING      34

#define DTV_API_VERSION         35

#define DTV_CODE_RATE_HP        36
#define DTV_CODE_RATE_LP        37
#define DTV_GUARD_INTERVAL      38
#define DTV_TRANSMISSION_MODE   39
#define DTV_HIERARCHY           40

#define DTV_ISDBT_LAYER_ENABLED 41

#define DTV_STREAM_ID           42
#define DTV_ISDBS_TS_ID_LEGACY  DTV_STREAM_ID
#define DTV_DVBT2_PLP_ID_LEGACY 43

#define DTV_ENUM_DELSYS         44

/* ATSC-MH */
#define DTV_ATSCMH_FIC_VER              45
#define DTV_ATSCMH_PARADE_ID            46
#define DTV_ATSCMH_NOG                  47
#define DTV_ATSCMH_TNOG                 48
#define DTV_ATSCMH_SGN                  49
#define DTV_ATSCMH_PRC                  50
#define DTV_ATSCMH_RS_FRAME_MODE        51
```

```
#define DTV_ATSCMH_RS_FRAME_ENSEMBLE    52
#define DTV_ATSCMH_RS_CODE_MODE_PRI     53
#define DTV_ATSCMH_RS_CODE_MODE_SEC     54
#define DTV_ATSCMH_SCCC_BLOCK_MODE      55
#define DTV_ATSCMH_SCCC_CODE_MODE_A     56
#define DTV_ATSCMH_SCCC_CODE_MODE_B     57
#define DTV_ATSCMH_SCCC_CODE_MODE_C     58
#define DTV_ATSCMH_SCCC_CODE_MODE_D     59

#define DTV_INTERLEAVING                        60
#define DTV_LNA                                 61

/* Quality parameters */
#define DTV_STAT_SIGNAL_STRENGTH        62
#define DTV_STAT_CNR                    63
#define DTV_STAT_PRE_ERROR_BIT_COUNT    64
#define DTV_STAT_PRE_TOTAL_BIT_COUNT    65
#define DTV_STAT_POST_ERROR_BIT_COUNT   66
#define DTV_STAT_POST_TOTAL_BIT_COUNT   67
#define DTV_STAT_ERROR_BLOCK_COUNT      68
#define DTV_STAT_TOTAL_BLOCK_COUNT      69

/* Physical layer scrambling */
#define DTV_SCRAMBLING_SEQUENCE_INDEX   70

#define DTV_MAX_COMMAND         DTV_SCRAMBLING_SEQUENCE_INDEX

/**
 * enum fe_pilot - Type of pilot tone
 *
 * @PILOT_ON:   Pilot tones enabled
 * @PILOT_OFF:  Pilot tones disabled
 * @PILOT_AUTO: Autodetect pilot tones
 */
enum fe_pilot {
        PILOT_ON,
        PILOT_OFF,
        PILOT_AUTO,
};

/**
 * enum fe_rolloff - Rolloff factor
 * @ROLLOFF_35:         Roloff factor: α=35%
 * @ROLLOFF_20:         Roloff factor: α=20%
 * @ROLLOFF_25:         Roloff factor: α=25%
 * @ROLLOFF_AUTO:       Auto-detect the roloff factor.
 *
 * .. note:
 *
 *    Roloff factor of 35% is implied on DVB-S. On DVB-S2, it is␣
↪default.
```

```
 */
enum fe_rolloff {
        ROLLOFF_35,
        ROLLOFF_20,
        ROLLOFF_25,
        ROLLOFF_AUTO,
};

/**
 * enum fe_delivery_system - Type of the delivery system
 *
 * @SYS_UNDEFINED:
 *      Undefined standard. Generally, indicates an error
 * @SYS_DVBC_ANNEX_A:
 *      Cable TV: DVB-C following ITU-T J.83 Annex A spec
 * @SYS_DVBC_ANNEX_B:
 *      Cable TV: DVB-C following ITU-T J.83 Annex B spec (ClearQAM)
 * @SYS_DVBC_ANNEX_C:
 *      Cable TV: DVB-C following ITU-T J.83 Annex C spec
 * @SYS_ISDBC:
 *      Cable TV: ISDB-C (no drivers yet)
 * @SYS_DVBT:
 *      Terrestrial TV: DVB-T
 * @SYS_DVBT2:
 *      Terrestrial TV: DVB-T2
 * @SYS_ISDBT:
 *      Terrestrial TV: ISDB-T
 * @SYS_ATSC:
 *      Terrestrial TV: ATSC
 * @SYS_ATSCMH:
 *      Terrestrial TV (mobile): ATSC-M/H
 * @SYS_DTMB:
 *      Terrestrial TV: DTMB
 * @SYS_DVBS:
 *      Satellite TV: DVB-S
 * @SYS_DVBS2:
 *      Satellite TV: DVB-S2
 * @SYS_TURBO:
 *      Satellite TV: DVB-S Turbo
 * @SYS_ISDBS:
 *      Satellite TV: ISDB-S
 * @SYS_DAB:
 *      Digital audio: DAB (not fully supported)
 * @SYS_DSS:
 *      Satellite TV: DSS (not fully supported)
 * @SYS_CMMB:
 *      Terrestrial TV (mobile): CMMB (not fully supported)
 * @SYS_DVBH:
 *      Terrestrial TV (mobile): DVB-H (standard deprecated)
 */
enum fe_delivery_system {
```

```
        SYS_UNDEFINED,
        SYS_DVBC_ANNEX_A,
        SYS_DVBC_ANNEX_B,
        SYS_DVBT,
        SYS_DSS,
        SYS_DVBS,
        SYS_DVBS2,
        SYS_DVBH,
        SYS_ISDBT,
        SYS_ISDBS,
        SYS_ISDBC,
        SYS_ATSC,
        SYS_ATSCMH,
        SYS_DTMB,
        SYS_CMMB,
        SYS_DAB,
        SYS_DVBT2,
        SYS_TURBO,
        SYS_DVBC_ANNEX_C,
};

/* backward compatibility definitions for delivery systems */
#define SYS_DVBC_ANNEX_AC       SYS_DVBC_ANNEX_A
#define SYS_DMBTH               SYS_DTMB /* DMB-TH is legacy name,␣
↪use DTMB */

/* ATSC-MH specific parameters */

/**
 * enum atscmh_sccc_block_mode - Type of Series Concatenated␣
↪Convolutional
 *                              Code Block Mode.
 *
 * @ATSCMH_SCCC_BLK_SEP:
 *      Separate SCCC: the SCCC outer code mode shall be set␣
↪independently
 *      for each Group Region (A, B, C, D)
 * @ATSCMH_SCCC_BLK_COMB:
 *      Combined SCCC: all four Regions shall have the same SCCC␣
↪outer
 *      code mode.
 * @ATSCMH_SCCC_BLK_RES:
 *      Reserved. Shouldn't be used.
 */
enum atscmh_sccc_block_mode {
        ATSCMH_SCCC_BLK_SEP      = 0,
        ATSCMH_SCCC_BLK_COMB     = 1,
        ATSCMH_SCCC_BLK_RES      = 2,
};

/**
```

```
 * enum atscmh_sccc_code_mode - Type of Series Concatenated␣
↪Convolutional
 *                              Code Rate.
 *
 * @ATSCMH_SCCC_CODE_HLF:
 *      The outer code rate of a SCCC Block is 1/2 rate.
 * @ATSCMH_SCCC_CODE_QTR:
 *      The outer code rate of a SCCC Block is 1/4 rate.
 * @ATSCMH_SCCC_CODE_RES:
 *      Reserved. Should not be used.
 */
enum atscmh_sccc_code_mode {
        ATSCMH_SCCC_CODE_HLF     = 0,
        ATSCMH_SCCC_CODE_QTR     = 1,
        ATSCMH_SCCC_CODE_RES     = 2,
};

/**
 * enum atscmh_rs_frame_ensemble - Reed Solomon(RS) frame ensemble.
 *
 * @ATSCMH_RSFRAME_ENS_PRI:     Primary Ensemble.
 * @ATSCMH_RSFRAME_ENS_SEC:     Secondary Ensemble.
 */
enum atscmh_rs_frame_ensemble {
        ATSCMH_RSFRAME_ENS_PRI   = 0,
        ATSCMH_RSFRAME_ENS_SEC   = 1,
};

/**
 * enum atscmh_rs_frame_mode - Reed Solomon (RS) frame mode.
 *
 * @ATSCMH_RSFRAME_PRI_ONLY:
 *      Single Frame: There is only a primary RS Frame for all Group
 *      Regions.
 * @ATSCMH_RSFRAME_PRI_SEC:
 *      Dual Frame: There are two separate RS Frames: Primary RS␣
↪Frame for
 *      Group Region A and B and Secondary RS Frame for Group␣
↪Region C and
 *      D.
 * @ATSCMH_RSFRAME_RES:
 *      Reserved. Shouldn't be used.
 */
enum atscmh_rs_frame_mode {
        ATSCMH_RSFRAME_PRI_ONLY  = 0,
        ATSCMH_RSFRAME_PRI_SEC   = 1,
        ATSCMH_RSFRAME_RES       = 2,
};

/**
 * enum atscmh_rs_code_mode
```

```
 * @ATSCMH_RSCODE_211_187:      Reed Solomon code (211,187).
 * @ATSCMH_RSCODE_223_187:      Reed Solomon code (223,187).
 * @ATSCMH_RSCODE_235_187:      Reed Solomon code (235,187).
 * @ATSCMH_RSCODE_RES:          Reserved. Shouldn't be used.
 */
enum atscmh_rs_code_mode {
        ATSCMH_RSCODE_211_187    = 0,
        ATSCMH_RSCODE_223_187    = 1,
        ATSCMH_RSCODE_235_187    = 2,
        ATSCMH_RSCODE_RES        = 3,
};

#define NO_STREAM_ID_FILTER     (~0U)
#define LNA_AUTO                (~0U)

/**
 * enum fecap_scale_params - scale types for the quality parameters.
 *
 * @FE_SCALE_NOT_AVAILABLE: That QoS measure is not available. That
 *                          could indicate a temporary or a␣
↪permanent
 *                          condition.
 * @FE_SCALE_DECIBEL: The scale is measured in 0.001 dB steps,␣
↪typically
 *                    used on signal measures.
 * @FE_SCALE_RELATIVE: The scale is a relative percentual measure,
 *                     ranging from 0 (0%) to 0xffff (100%).
 * @FE_SCALE_COUNTER: The scale counts the occurrence of an event,␣
↪like
 *                    bit error, block error, lapsed time.
 */
enum fecap_scale_params {
        FE_SCALE_NOT_AVAILABLE = 0,
        FE_SCALE_DECIBEL,
        FE_SCALE_RELATIVE,
        FE_SCALE_COUNTER
};

/**
 * struct dtv_stats - Used for reading a DTV status property
 *
 * @scale:
 *      Filled with enum fecap_scale_params - the scale in usage
 *      for that parameter
 *
 * @svalue:
 *      integer value of the measure, for %FE_SCALE_DECIBEL,
 *      used for dB measures. The unit is 0.001 dB.
 *
 * @uvalue:
 *      unsigned integer value of the measure, used when @scale is
```

```
 *      either %FE_SCALE_RELATIVE or %FE_SCALE_COUNTER.
 *
 * For most delivery systems, this will return a single value for␣
↪each
 * parameter.
 *
 * It should be noticed, however, that new OFDM delivery systems␣
↪like
 * ISDB can use different modulation types for each group of␣
↪carriers.
 * On such standards, up to 8 groups of statistics can be provided,␣
↪one
 * for each carrier group (called "layer" on ISDB).
 *
 * In order to be consistent with other delivery systems, the first
 * value refers to the entire set of carriers ("global").
 *
 * @scale should use the value %FE_SCALE_NOT_AVAILABLE when
 * the value for the entire group of carriers or from one specific␣
↪layer
 * is not provided by the hardware.
 *
 * @len should be filled with the latest filled status + 1.
 *
 * In other words, for ISDB, those values should be filled like::
 *
 *      u.st.stat.svalue[0] = global statistics;
 *      u.st.stat.scale[0] = FE_SCALE_DECIBEL;
 *      u.st.stat.value[1] = layer A statistics;
 *      u.st.stat.scale[1] = FE_SCALE_NOT_AVAILABLE (if not␣
↪available);
 *      u.st.stat.svalue[2] = layer B statistics;
 *      u.st.stat.scale[2] = FE_SCALE_DECIBEL;
 *      u.st.stat.svalue[3] = layer C statistics;
 *      u.st.stat.scale[3] = FE_SCALE_DECIBEL;
 *      u.st.len = 4;
 */
struct dtv_stats {
        __u8 scale;     /* enum fecap_scale_params type */
        union {
                __u64 uvalue;   /* for counters and relative scales␣
↪*/
                __s64 svalue;   /* for 0.001 dB measures */
        };
} __attribute__ ((packed));

#define MAX_DTV_STATS   4

/**
 * struct dtv_fe_stats - store Digital TV frontend statistics
 *
```

```
 * @len:        length of the statistics - if zero, stats is␣
↪disabled.
 * @stat:       array with digital TV statistics.
 *
 * On most standards, @len can either be 0 or 1. However, for ISDB,␣
↪each
 * layer is modulated in separate. So, each layer may have its own␣
↪set
 * of statistics. If so, stat[0] carries on a global value for the␣
↪property.
 * Indexes 1 to 3 means layer A to B.
 */
struct dtv_fe_stats {
        __u8 len;
        struct dtv_stats stat[MAX_DTV_STATS];
} __attribute__ ((packed));

/**
 * struct dtv_property - store one of frontend command and its value
 *
 * @cmd:                Digital TV command.
 * @reserved:           Not used.
 * @u:                  Union with the values for the command.
 * @u.data:             A unsigned 32 bits integer with command␣
↪value.
 * @u.buffer:           Struct to store bigger properties.
 *                      Currently unused.
 * @u.buffer.data:      an unsigned 32-bits array.
 * @u.buffer.len:       number of elements of the buffer.
 * @u.buffer.reserved1: Reserved.
 * @u.buffer.reserved2: Reserved.
 * @u.st:               a &struct dtv_fe_stats array of statistics.
 * @result:             Currently unused.
 *
 */
struct dtv_property {
        __u32 cmd;
        __u32 reserved[3];
        union {
                __u32 data;
                struct dtv_fe_stats st;
                struct {
                        __u8 data[32];
                        __u32 len;
                        __u32 reserved1[3];
                        void *reserved2;
                } buffer;
        } u;
        int result;
} __attribute__ ((packed));
```

```
/* num of properties cannot exceed DTV_IOCTL_MAX_MSGS per ioctl */
#define DTV_IOCTL_MAX_MSGS 64

/**
 * struct dtv_properties - a set of command/value pairs.
 *
 * @num:        amount of commands stored at the struct.
 * @props:      a pointer to &struct dtv_property.
 */
struct dtv_properties {
        __u32 num;
        struct dtv_property *props;
};

/*
 * When set, this flag will disable any zigzagging or other␣
↪"normal" tuning
 * behavior. Additionally, there will be no automatic monitoring of␣
↪the lock
 * status, and hence no frontend events will be generated. If a␣
↪frontend device
 * is closed, this flag will be automatically turned off when the␣
↪device is
 * reopened read-write.
 */
#define FE_TUNE_MODE_ONESHOT 0x01

/* Digital TV Frontend API calls */

#define FE_GET_INFO                _IOR('o', 61, struct dvb_
↪frontend_info)

#define FE_DISEQC_RESET_OVERLOAD   _IO('o', 62)
#define FE_DISEQC_SEND_MASTER_CMD  _IOW('o', 63, struct dvb_diseqc_
↪master_cmd)
#define FE_DISEQC_RECV_SLAVE_REPLY _IOR('o', 64, struct dvb_diseqc_
↪slave_reply)
#define FE_DISEQC_SEND_BURST       _IO('o', 65)  /* fe_sec_mini_cmd_
↪t */

#define FE_SET_TONE                _IO('o', 66)  /* fe_sec_tone_
↪mode_t */
#define FE_SET_VOLTAGE             _IO('o', 67)  /* fe_sec_voltage_
↪t */
#define FE_ENABLE_HIGH_LNB_VOLTAGE _IO('o', 68)  /* int */

#define FE_READ_STATUS             _IOR('o', 69, fe_status_t)
#define FE_READ_BER                _IOR('o', 70, __u32)
#define FE_READ_SIGNAL_STRENGTH    _IOR('o', 71, __u16)
#define FE_READ_SNR                _IOR('o', 72, __u16)
#define FE_READ_UNCORRECTED_BLOCKS _IOR('o', 73, __u32)
```

```
#define FE_SET_FRONTEND_TUNE_MODE  _IO('o', 81) /* unsigned int */
#define FE_GET_EVENT               _IOR('o', 78, struct dvb_
↪frontend_event)

#define FE_DISHNETWORK_SEND_LEGACY_CMD _IO('o', 80) /* unsigned int␣
↪*/

#define FE_SET_PROPERTY            _IOW('o', 82, struct dtv_
↪properties)
#define FE_GET_PROPERTY            _IOR('o', 83, struct dtv_
↪properties)

#if defined(__DVB_CORE__) || !defined(__KERNEL__)

/*
 * DEPRECATED: Everything below is deprecated in favor of DVBv5 API
 *
 * The DVBv3 only ioctls, structs and enums should not be used on
 * newer programs, as it doesn't support the second generation of
 * digital TV standards, nor supports newer delivery systems.
 * They also don't support modern frontends with usually support␣
↪multiple
 * delivery systems.
 *
 * Drivers shouldn't use them.
 *
 * New applications should use DVBv5 delivery system instead
 */

/*
 */

enum fe_bandwidth {
        BANDWIDTH_8_MHZ,
        BANDWIDTH_7_MHZ,
        BANDWIDTH_6_MHZ,
        BANDWIDTH_AUTO,
        BANDWIDTH_5_MHZ,
        BANDWIDTH_10_MHZ,
        BANDWIDTH_1_712_MHZ,
};

/* This is kept for legacy userspace support */
typedef enum fe_sec_voltage fe_sec_voltage_t;
typedef enum fe_caps fe_caps_t;
typedef enum fe_type fe_type_t;
typedef enum fe_sec_tone_mode fe_sec_tone_mode_t;
typedef enum fe_sec_mini_cmd fe_sec_mini_cmd_t;
typedef enum fe_status fe_status_t;
typedef enum fe_spectral_inversion fe_spectral_inversion_t;
```

```
typedef enum fe_code_rate fe_code_rate_t;
typedef enum fe_modulation fe_modulation_t;
typedef enum fe_transmit_mode fe_transmit_mode_t;
typedef enum fe_bandwidth fe_bandwidth_t;
typedef enum fe_guard_interval fe_guard_interval_t;
typedef enum fe_hierarchy fe_hierarchy_t;
typedef enum fe_pilot fe_pilot_t;
typedef enum fe_rolloff fe_rolloff_t;
typedef enum fe_delivery_system fe_delivery_system_t;

/* DVBv3 structs */

struct dvb_qpsk_parameters {
        __u32           symbol_rate;  /* symbol rate in Symbols per␣
↪second */
        fe_code_rate_t  fec_inner;    /* forward error correction␣
↪(see above) */
};

struct dvb_qam_parameters {
        __u32           symbol_rate; /* symbol rate in Symbols per␣
↪second */
        fe_code_rate_t  fec_inner;   /* forward error correction␣
↪(see above) */
        fe_modulation_t modulation;  /* modulation type (see above)␣
↪*/
};

struct dvb_vsb_parameters {
        fe_modulation_t modulation;  /* modulation type (see above)␣
↪*/
};

struct dvb_ofdm_parameters {
        fe_bandwidth_t      bandwidth;
        fe_code_rate_t      code_rate_HP;  /* high priority stream␣
↪code rate */
        fe_code_rate_t      code_rate_LP;  /* low priority stream␣
↪code rate */
        fe_modulation_t     constellation; /* modulation type (see␣
↪above) */
        fe_transmit_mode_t  transmission_mode;
        fe_guard_interval_t guard_interval;
        fe_hierarchy_t      hierarchy_information;
};

struct dvb_frontend_parameters {
        __u32 frequency;  /* (absolute) frequency in Hz for DVB-C/
↪DVB-T/ATSC */
                          /* intermediate frequency in kHz for␣
↪DVB-S */
```

```
        fe_spectral_inversion_t inversion;
        union {
                struct dvb_qpsk_parameters qpsk;        /* DVB-S */
                struct dvb_qam_parameters  qam;         /* DVB-C */
                struct dvb_ofdm_parameters ofdm;        /* DVB-T */
                struct dvb_vsb_parameters vsb;          /* ATSC */
        } u;
};

struct dvb_frontend_event {
        fe_status_t status;
        struct dvb_frontend_parameters parameters;
};

/* DVBv3 API calls */

#define FE_SET_FRONTEND            _IOW('o', 76, struct dvb_
↪frontend_parameters)
#define FE_GET_FRONTEND            _IOR('o', 77, struct dvb_
↪frontend_parameters)

#endif

#endif /*_DVBFRONTEND_H_*/
```

dmx.h

```
/* SPDX-License-Identifier: LGPL-2.1+ WITH Linux-syscall-note */
/*
 * dmx.h
 *
 * Copyright (C) 2000 Marcus Metzler <marcus@convergence.de>
 *                  & Ralph  Metzler <ralph@convergence.de>
 *                    for convergence integrated media GmbH
 *
 * This program is free software; you can redistribute it and/or
 * modify it under the terms of the GNU Lesser General Public␣
↪License
 * as published by the Free Software Foundation; either version 2.1
 * of the License, or (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU Lesser General Public␣
↪License
 * along with this program; if not, write to the Free Software
 * Foundation, Inc., 59 Temple Place - Suite 330, Boston, MA ␣
```

```
↪02111-1307, USA.
 *
 */

#ifndef _UAPI_DVBDMX_H_
#define _UAPI_DVBDMX_H_

#include <linux/types.h>
#ifndef __KERNEL__
#include <time.h>
#endif

#define DMX_FILTER_SIZE 16

/**
 * enum dmx_output - Output for the demux.
 *
 * @:c:type:DMX_OUT_DECODER <dmx_output>:
 *      Streaming directly to decoder.
 * @:c:type:DMX_OUT_TAP <dmx_output>:
 *      Output going to a memory buffer (to be retrieved via the␣
↪read command).
 *      Delivers the stream output to the demux device on which the␣
↪ioctl
 *      is called.
 * @:c:type:DMX_OUT_TS_TAP <dmx_output>:
 *      Output multiplexed into a new TS (to be retrieved by␣
↪reading from the
 *      logical DVR device). Routes output to the logical DVR device
 *      ``/dev/dvb/adapter?/dvr?``, which delivers a TS multiplexed␣
↪from all
 *      filters for which @:c:type:DMX_OUT_TS_TAP <dmx_output> was␣
↪specified.
 * @:c:type:DMX_OUT_TSDEMUX_TAP <dmx_output>:
 *      Like @:c:type:DMX_OUT_TS_TAP <dmx_output> but retrieved␣
↪from the DMX device.
 */
enum dmx_output {
        DMX_OUT_DECODER,
        DMX_OUT_TAP,
        DMX_OUT_TS_TAP,
        DMX_OUT_TSDEMUX_TAP
};

/**
 * dmx_input - Input from the demux.
 *
 * @:c:type:DMX_IN_FRONTEND <dmx_input>:    Input from a front-end␣
↪device.
 * @:c:type:DMX_IN_DVR <dmx_input>:         Input from the logical␣
↪DVR device.
```

```
 */
dmx_input {
        DMX_IN_FRONTEND,
        DMX_IN_DVR
};

/**
 * dmx_ts_pes - type of the PES filter.
 *
 * @:c:type:DMX_PES_AUDIO0 <dmx_pes_type>:     first audio PID.␣
↪Also referred as @DMX_PES_AUDIO.
 * @:c:type:DMX_PES_VIDEO0 <dmx_pes_type>:     first video PID.␣
↪Also referred as @DMX_PES_VIDEO.
 * @:c:type:DMX_PES_TELETEXT0 <dmx_pes_type>:  first teletext PID.␣
↪Also referred as @DMX_PES_TELETEXT.
 * @:c:type:DMX_PES_SUBTITLE0 <dmx_pes_type>:  first subtitle PID.␣
↪Also referred as @DMX_PES_SUBTITLE.
 * @:c:type:DMX_PES_PCR0 <dmx_pes_type>:       first Program Clock␣
↪Reference PID.
 *                       Also referred as @DMX_PES_PCR.
 *
 * @:c:type:DMX_PES_AUDIO1 <dmx_pes_type>:     second audio PID.
 * @:c:type:DMX_PES_VIDEO1 <dmx_pes_type>:     second video PID.
 * @:c:type:DMX_PES_TELETEXT1 <dmx_pes_type>:  second teletext PID.
 * @:c:type:DMX_PES_SUBTITLE1 <dmx_pes_type>:  second subtitle PID.
 * @:c:type:DMX_PES_PCR1 <dmx_pes_type>:       second Program Clock␣
↪Reference PID.
 *
 * @:c:type:DMX_PES_AUDIO2 <dmx_pes_type>:     third audio PID.
 * @:c:type:DMX_PES_VIDEO2 <dmx_pes_type>:     third video PID.
 * @:c:type:DMX_PES_TELETEXT2 <dmx_pes_type>:  third teletext PID.
 * @:c:type:DMX_PES_SUBTITLE2 <dmx_pes_type>:  third subtitle PID.
 * @:c:type:DMX_PES_PCR2 <dmx_pes_type>:       third Program Clock␣
↪Reference PID.
 *
 * @:c:type:DMX_PES_AUDIO3 <dmx_pes_type>:     fourth audio PID.
 * @:c:type:DMX_PES_VIDEO3 <dmx_pes_type>:     fourth video PID.
 * @:c:type:DMX_PES_TELETEXT3 <dmx_pes_type>:  fourth teletext PID.
 * @:c:type:DMX_PES_SUBTITLE3 <dmx_pes_type>:  fourth subtitle PID.
 * @:c:type:DMX_PES_PCR3 <dmx_pes_type>:       fourth Program Clock␣
↪Reference PID.
 *
 * @:c:type:DMX_PES_OTHER <dmx_pes_type>:      any other PID.
 */

dmx_ts_pes {
        DMX_PES_AUDIO0,
        DMX_PES_VIDEO0,
        DMX_PES_TELETEXT0,
        DMX_PES_SUBTITLE0,
        DMX_PES_PCR0,
```

```
        DMX_PES_AUDIO1,
        DMX_PES_VIDEO1,
        DMX_PES_TELETEXT1,
        DMX_PES_SUBTITLE1,
        DMX_PES_PCR1,

        DMX_PES_AUDIO2,
        DMX_PES_VIDEO2,
        DMX_PES_TELETEXT2,
        DMX_PES_SUBTITLE2,
        DMX_PES_PCR2,

        DMX_PES_AUDIO3,
        DMX_PES_VIDEO3,
        DMX_PES_TELETEXT3,
        DMX_PES_SUBTITLE3,
        DMX_PES_PCR3,

        DMX_PES_OTHER
};

#define DMX_PES_AUDIO    DMX_PES_AUDIO0
#define DMX_PES_VIDEO    DMX_PES_VIDEO0
#define DMX_PES_TELETEXT DMX_PES_TELETEXT0
#define DMX_PES_SUBTITLE DMX_PES_SUBTITLE0
#define DMX_PES_PCR      DMX_PES_PCR0

/**
 * struct dmx_filter - Specifies a section header filter.
 *
 * @filter: bit array with bits to be matched at the section header.
 * @mask: bits that are valid at the filter bit array.
 * @mode: mode of match: if bit is zero, it will match if equal␣
↪(positive
 *        match); if bit is one, it will match if the bit is␣
↪negated.
 *
 * Note: All arrays in this struct have a size of DMX_FILTER_SIZE␣
↪(16 bytes).
 */
struct dmx_filter {
        __u8  filter[DMX_FILTER_SIZE];
        __u8  mask[DMX_FILTER_SIZE];
        __u8  mode[DMX_FILTER_SIZE];
};

/**
 * struct dmx_sct_filter_params - Specifies a section filter.
 *
 * @pid: PID to be filtered.
```

```
 * @filter: section header filter, as defined by &struct dmx_filter.
 * @timeout: maximum time to filter, in milliseconds.
 * @flags: extra flags for the section filter.
 *
 * Carries the configuration for a MPEG-TS section filter.
 *
 * The @flags can be:
 *
 *      - %DMX_CHECK_CRC - only deliver sections where the CRC␣
↪check succeeded;
 *      - %DMX_ONESHOT - disable the section filter after one␣
↪section
 *        has been delivered;
 *      - %DMX_IMMEDIATE_START - Start filter immediately without␣
↪requiring a
 *        :ref:`DMX_START`.
 */
struct dmx_sct_filter_params {
        __u16             pid;
        struct dmx_filter filter;
        __u32             timeout;
        __u32             flags;
#define DMX_CHECK_CRC       1
#define DMX_ONESHOT         2
#define DMX_IMMEDIATE_START 4
};

/**
 * struct dmx_pes_filter_params - Specifies Packetized Elementary␣
↪Stream (PES)
 *      filter parameters.
 *
 * @pid:        PID to be filtered.
 * @input:      Demux input, as specified by &enum dmx_input.
 * @output:     Demux output, as specified by &enum dmx_output.
 * @pes_type:   Type of the pes filter, as specified by &enum dmx_
↪pes_type.
 * @flags:      Demux PES flags.
 */
struct dmx_pes_filter_params {
        __u16           pid;
        dmx_input  input;
        enum dmx_output output;
        dmx_ts_pes pes_type;
        __u32           flags;
};

/**
 * struct dmx_stc - Stores System Time Counter (STC) information.
 *
 * @num: input data: number of the STC, from 0 to N.
```

```
 * @base: output: divisor for STC to get 90 kHz clock.
 * @stc: output: stc in @base * 90 kHz units.
 */
struct dmx_stc {
        unsigned int num;
        unsigned int base;
        __u64 stc;
};

/**
 * enum dmx_buffer_flags - DMX memory-mapped buffer flags
 *
 * @:c:type:DMX_BUFFER_FLAG_HAD_CRC32_DISCARD <dmx_buffer_flags>:
 *      Indicates that the Kernel discarded one or more frames due␣
↪to wrong
 *      CRC32 checksum.
 * @:c:type:DMX_BUFFER_FLAG_TEI <dmx_buffer_flags>:
 *      Indicates that the Kernel has detected a Transport Error␣
↪indicator
 *      (TEI) on a filtered pid.
 * @:c:type:DMX_BUFFER_PKT_COUNTER_MISMATCH <dmx_buffer_flags>:
 *      Indicates that the Kernel has detected a packet counter␣
↪mismatch
 *      on a filtered pid.
 * @:c:type:DMX_BUFFER_FLAG_DISCONTINUITY_DETECTED <dmx_buffer_
↪flags>:
 *      Indicates that the Kernel has detected one or more frame␣
↪discontinuity.
 * @:c:type:DMX_BUFFER_FLAG_DISCONTINUITY_INDICATOR <dmx_buffer_
↪flags>:
 *      Received at least one packet with a frame discontinuity␣
↪indicator.
 */

enum dmx_buffer_flags {
        DMX_BUFFER_FLAG_HAD_CRC32_DISCARD               = 1 << 0,
        DMX_BUFFER_FLAG_TEI                             = 1 << 1,
        DMX_BUFFER_PKT_COUNTER_MISMATCH                 = 1 << 2,
        DMX_BUFFER_FLAG_DISCONTINUITY_DETECTED          = 1 << 3,
        DMX_BUFFER_FLAG_DISCONTINUITY_INDICATOR         = 1 << 4,
};

/**
 * struct dmx_buffer - dmx buffer info
 *
 * @index:      id number of the buffer
 * @bytesused:  number of bytes occupied by data in the buffer␣
↪(payload);
 * @offset:     for buffers with memory == DMX_MEMORY_MMAP;
 *              offset from the start of the device memory for this␣
↪plane,
```

```
 *               (or a "cookie" that should be passed to mmap() as␣
↪offset)
 * @length:      size in bytes of the buffer
 * @flags:       bit array of buffer flags as defined by &enum dmx_
↪buffer_flags.
 *               Filled only at &DMX_DQBUF.
 * @count:       monotonic counter for filled buffers. Helps to␣
↪identify
 *               data stream loses. Filled only at &DMX_DQBUF.
 *
 * Contains data exchanged by application and driver using one of␣
↪the streaming
 * I/O methods.
 *
 * Please notice that, for &DMX_QBUF, only @index should be filled.
 * On &DMX_DQBUF calls, all fields will be filled by the Kernel.
 */
struct dmx_buffer {
        __u32                   index;
        __u32                   bytesused;
        __u32                   offset;
        __u32                   length;
        __u32                   flags;
        __u32                   count;
};

/**
 * struct dmx_requestbuffers - request dmx buffer information
 *
 * @count:       number of requested buffers,
 * @size:        size in bytes of the requested buffer
 *
 * Contains data used for requesting a dmx buffer.
 * All reserved fields must be set to zero.
 */
struct dmx_requestbuffers {
        __u32                   count;
        __u32                   size;
};

/**
 * struct dmx_exportbuffer - export of dmx buffer as DMABUF file␣
↪descriptor
 *
 * @index:       id number of the buffer
 * @flags:       flags for newly created file, currently only O_
↪CLOEXEC is
 *               supported, refer to manual of open syscall for more␣
↪details
 * @fd:          file descriptor associated with DMABUF (set by␣
↪driver)
```

```
 *
 * Contains data used for exporting a dmx buffer as DMABUF file␣
↪descriptor.
 * The buffer is identified by a 'cookie' returned by DMX_QUERYBUF
 * (identical to the cookie used to mmap() the buffer to userspace).
↪ All
 * reserved fields must be set to zero. The field reserved0 is␣
↪expected to
 * become a structure 'type' allowing an alternative layout of the␣
↪structure
 * content. Therefore this field should not be used for any other␣
↪extensions.
 */
struct dmx_exportbuffer {
        __u32           index;
        __u32           flags;
        __s32           fd;
};

#define DMX_START                _IO('o', 41)
#define DMX_STOP                 _IO('o', 42)
#define DMX_SET_FILTER           _IOW('o', 43, struct dmx_sct_
↪filter_params)
#define DMX_SET_PES_FILTER       _IOW('o', 44, struct dmx_pes_
↪filter_params)
#define DMX_SET_BUFFER_SIZE      _IO('o', 45)
#define DMX_GET_PES_PIDS         _IOR('o', 47, __u16[5])
#define DMX_GET_STC              _IOWR('o', 50, struct dmx_stc)
#define DMX_ADD_PID              _IOW('o', 51, __u16)
#define DMX_REMOVE_PID           _IOW('o', 52, __u16)

#if !defined(__KERNEL__)

/* This is needed for legacy userspace support */
typedef enum dmx_output dmx_output_t;
typedef dmx_input dmx_input_t;
typedef dmx_ts_pes dmx_pes_type_t;
typedef struct dmx_filter dmx_filter_t;

#endif

#define DMX_REQBUFS              _IOWR('o', 60, struct dmx_
↪requestbuffers)
#define DMX_QUERYBUF             _IOWR('o', 61, struct dmx_buffer)
#define DMX_EXPBUF               _IOWR('o', 62, struct dmx_
↪exportbuffer)
#define DMX_QBUF                 _IOWR('o', 63, struct dmx_buffer)
#define DMX_DQBUF                _IOWR('o', 64, struct dmx_buffer)

#endif /* _DVBDMX_H_ */
```

ca.h

```
/* SPDX-License-Identifier: LGPL-2.1+ WITH Linux-syscall-note */
/*
 * ca.h
 *
 * Copyright (C) 2000 Ralph  Metzler <ralph@convergence.de>
 *                    & Marcus Metzler <marcus@convergence.de>
 *                      for convergence integrated media GmbH
 *
 * This program is free software; you can redistribute it and/or
 * modify it under the terms of the GNU General Lesser Public␣
↪License
 * as published by the Free Software Foundation; either version 2.1
 * of the License, or (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU Lesser General Public␣
↪License
 * along with this program; if not, write to the Free Software
 * Foundation, Inc., 59 Temple Place - Suite 330, Boston, MA ␣
↪02111-1307, USA.
 *
 */

#ifndef _DVBCA_H_
#define _DVBCA_H_

/**
 * struct ca_slot_info - CA slot interface types and info.
 *
 * @num:        slot number.
 * @type:       slot type.
 * @flags:      flags applicable to the slot.
 *
 * This struct stores the CA slot information.
 *
 * @type can be:
 *
 *      - %CA_CI - CI high level interface;
 *      - %CA_CI_LINK - CI link layer level interface;
 *      - %CA_CI_PHYS - CI physical layer level interface;
 *      - %CA_DESCR - built-in descrambler;
 *      - %CA_SC -simple smart card interface.
 *
 * @flags can be:
 *
```

```
 *      - %CA_CI_MODULE_PRESENT - module (or card) inserted;
 *      - %CA_CI_MODULE_READY - module is ready for usage.
 */

struct ca_slot_info {
        int num;
        int type;
#define CA_CI            1
#define CA_CI_LINK       2
#define CA_CI_PHYS       4
#define CA_DESCR         8
#define CA_SC          128

        unsigned int flags;
#define CA_CI_MODULE_PRESENT 1
#define CA_CI_MODULE_READY   2
};

/**
 * struct ca_descr_info - descrambler types and info.
 *
 * @num:        number of available descramblers (keys).
 * @type:       type of supported scrambling system.
 *
 * Identifies the number of descramblers and their type.
 *
 * @type can be:
 *
 *      - %CA_ECD - European Common Descrambler (ECD) hardware;
 *      - %CA_NDS - Videoguard (NDS) hardware;
 *      - %CA_DSS - Distributed Sample Scrambling (DSS) hardware.
 */
struct ca_descr_info {
        unsigned int num;
        unsigned int type;
#define CA_ECD           1
#define CA_NDS           2
#define CA_DSS           4
};

/**
 * struct ca_caps - CA slot interface capabilities.
 *
 * @slot_num:   total number of CA card and module slots.
 * @slot_type:  bitmap with all supported types as defined at
 *              &struct ca_slot_info (e. g. %CA_CI, %CA_CI_LINK,␣
↪etc).
 * @descr_num:  total number of descrambler slots (keys)
 * @descr_type: bitmap with all supported types as defined at
 *              &struct ca_descr_info (e. g. %CA_ECD, %CA_NDS, etc).
 */
```

```
struct ca_caps {
        unsigned int slot_num;
        unsigned int slot_type;
        unsigned int descr_num;
        unsigned int descr_type;
};

/**
 * struct ca_msg - a message to/from a CI-CAM
 *
 * @index:      unused
 * @type:       unused
 * @length:     length of the message
 * @msg:        message
 *
 * This struct carries a message to be send/received from a CI CA␣
↪module.
 */
struct ca_msg {
        unsigned int index;
        unsigned int type;
        unsigned int length;
        unsigned char msg[256];
};

/**
 * struct ca_descr - CA descrambler control words info
 *
 * @index: CA Descrambler slot
 * @parity: control words parity, where 0 means even and 1 means odd
 * @cw: CA Descrambler control words
 */
struct ca_descr {
        unsigned int index;
        unsigned int parity;
        unsigned char cw[8];
};

#define CA_RESET          _IO('o', 128)
#define CA_GET_CAP        _IOR('o', 129, struct ca_caps)
#define CA_GET_SLOT_INFO  _IOR('o', 130, struct ca_slot_info)
#define CA_GET_DESCR_INFO _IOR('o', 131, struct ca_descr_info)
#define CA_GET_MSG        _IOR('o', 132, struct ca_msg)
#define CA_SEND_MSG       _IOW('o', 133, struct ca_msg)
#define CA_SET_DESCR      _IOW('o', 134, struct ca_descr)

#if !defined(__KERNEL__)

/* This is needed for legacy userspace support */
typedef struct ca_slot_info ca_slot_info_t;
typedef struct ca_descr_info  ca_descr_info_t;
```

```
typedef struct ca_caps  ca_caps_t;
typedef struct ca_msg ca_msg_t;
typedef struct ca_descr ca_descr_t;

#endif

#endif
```

net.h

```
/* SPDX-License-Identifier: LGPL-2.1+ WITH Linux-syscall-note */
/*
 * net.h
 *
 * Copyright (C) 2000 Marcus Metzler <marcus@convergence.de>
 *                    & Ralph  Metzler <ralph@convergence.de>
 *                      for convergence integrated media GmbH
 *
 * This program is free software; you can redistribute it and/or
 * modify it under the terms of the GNU Lesser General Public␣
↪License
 * as published by the Free Software Foundation; either version 2.1
 * of the License, or (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU Lesser General Public␣
↪License
 * along with this program; if not, write to the Free Software
 * Foundation, Inc., 59 Temple Place - Suite 330, Boston, MA ␣
↪02111-1307, USA.
 *
 */

#ifndef _DVBNET_H_
#define _DVBNET_H_

#include <linux/types.h>

/**
 * struct dvb_net_if - describes a DVB network interface
 *
 * @pid: Packet ID (PID) of the MPEG-TS that contains data
 * @if_num: number of the Digital TV interface.
 * @feedtype: Encapsulation type of the feed.
 *
 * A MPEG-TS stream may contain packet IDs with IP packages on it.
```

```
 * This struct describes it, and the type of encoding.
 *
 * @feedtype can be:
 *
 *      - %DVB_NET_FEEDTYPE_MPE for MPE encoding
 *      - %DVB_NET_FEEDTYPE_ULE for ULE encoding.
 */
struct dvb_net_if {
        __u16 pid;
        __u16 if_num;
        __u8  feedtype;
#define DVB_NET_FEEDTYPE_MPE 0  /* multi protocol encapsulation */
#define DVB_NET_FEEDTYPE_ULE 1  /* ultra lightweight encapsulation␣
↪*/
};

#define NET_ADD_IF    _IOWR('o', 52, struct dvb_net_if)
#define NET_REMOVE_IF _IO('o', 53)
#define NET_GET_IF    _IOWR('o', 54, struct dvb_net_if)

/* binary compatibility cruft: */
struct __dvb_net_if_old {
        __u16 pid;
        __u16 if_num;
};
#define __NET_ADD_IF_OLD _IOWR('o', 52, struct __dvb_net_if_old)
#define __NET_GET_IF_OLD _IOWR('o', 54, struct __dvb_net_if_old)

#endif /*_DVBNET_H_*/
```

Legacy uAPI

audio.h

```
/* SPDX-License-Identifier: LGPL-2.1+ WITH Linux-syscall-note */
/*
 * audio.h - DEPRECATED MPEG-TS audio decoder API
 *
 * NOTE: should not be used on future drivers
 *
 * Copyright (C) 2000 Ralph  Metzler <ralph@convergence.de>
 *                  & Marcus Metzler <marcus@convergence.de>
 *                    for convergence integrated media GmbH
 *
 * This program is free software; you can redistribute it and/or
 * modify it under the terms of the GNU General Lesser Public␣
↪License
 * as published by the Free Software Foundation; either version 2.1
 * of the License, or (at your option) any later version.
 *
```

```
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU Lesser General Public␣
↪License
 * along with this program; if not, write to the Free Software
 * Foundation, Inc., 59 Temple Place - Suite 330, Boston, MA ␣
↪02111-1307, USA.
 *
 */

#ifndef _DVBAUDIO_H_
#define _DVBAUDIO_H_

#include <linux/types.h>

typedef enum {
        AUDIO_SOURCE_DEMUX, /* Select the demux as the main source␣
↪*/
        AUDIO_SOURCE_MEMORY /* Select internal memory as the main␣
↪source */
} audio_stream_source_t;

typedef enum {
        AUDIO_STOPPED,      /* Device is stopped */
        AUDIO_PLAYING,      /* Device is currently playing */
        AUDIO_PAUSED        /* Device is paused */
} audio_play_state_t;

typedef enum {
        AUDIO_STEREO,
        AUDIO_MONO_LEFT,
        AUDIO_MONO_RIGHT,
        AUDIO_MONO,
        AUDIO_STEREO_SWAPPED
} audio_channel_select_t;

typedef struct audio_mixer {
        unsigned int volume_left;
        unsigned int volume_right;
  /* what else do we need? bass, pass-through, ... */
} audio_mixer_t;

typedef struct audio_status {
        int                    AV_sync_state;  /* sync audio and␣
↪video? */
        int                    mute_state;     /* audio is muted */
        audio_play_state_t     play_state;     /* current playback␣
↪state */
```

```
        audio_stream_source_t  stream_source;  /* current stream␣
↪source */
        audio_channel_select_t channel_select; /* currently␣
↪selected channel */
        int                    bypass_mode;    /* pass on audio␣
↪data to */
        audio_mixer_t          mixer_state;    /* current mixer␣
↪state */
} audio_status_t;                              /* separate decoder␣
↪hardware */

/* for GET_CAPABILITIES and SET_FORMAT, the latter should only set␣
↪one bit */
#define AUDIO_CAP_DTS    1
#define AUDIO_CAP_LPCM   2
#define AUDIO_CAP_MP1    4
#define AUDIO_CAP_MP2    8
#define AUDIO_CAP_MP3   16
#define AUDIO_CAP_AAC   32
#define AUDIO_CAP_OGG   64
#define AUDIO_CAP_SDDS 128
#define AUDIO_CAP_AC3  256

#define AUDIO_STOP                 _IO('o', 1)
#define AUDIO_PLAY                 _IO('o', 2)
#define AUDIO_PAUSE                _IO('o', 3)
#define AUDIO_CONTINUE             _IO('o', 4)
#define AUDIO_SELECT_SOURCE        _IO('o', 5)
#define AUDIO_SET_MUTE             _IO('o', 6)
#define AUDIO_SET_AV_SYNC          _IO('o', 7)
#define AUDIO_SET_BYPASS_MODE      _IO('o', 8)
#define AUDIO_CHANNEL_SELECT       _IO('o', 9)
#define AUDIO_GET_STATUS           _IOR('o', 10, audio_status_t)

#define AUDIO_GET_CAPABILITIES     _IOR('o', 11, unsigned int)
#define AUDIO_CLEAR_BUFFER         _IO('o',  12)
#define AUDIO_SET_ID               _IO('o', 13)
#define AUDIO_SET_MIXER            _IOW('o', 14, audio_mixer_t)
#define AUDIO_SET_STREAMTYPE       _IO('o', 15)
#define AUDIO_BILINGUAL_CHANNEL_SELECT _IO('o', 20)

#endif /* _DVBAUDIO_H_ */
```

video.h

```
/* SPDX-License-Identifier: LGPL-2.1+ WITH Linux-syscall-note */
/*
 * video.h - DEPRECATED MPEG-TS video decoder API
 *
 * NOTE: should not be used on future drivers
 *
 * Copyright (C) 2000 Marcus Metzler <marcus@convergence.de>
 *                  & Ralph  Metzler <ralph@convergence.de>
 *                    for convergence integrated media GmbH
 *
 * This program is free software; you can redistribute it and/or
 * modify it under the terms of the GNU Lesser General Public␣
↪License
 * as published by the Free Software Foundation; either version 2.1
 * of the License, or (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU Lesser General Public␣
↪License
 * along with this program; if not, write to the Free Software
 * Foundation, Inc., 59 Temple Place - Suite 330, Boston, MA ␣
↪02111-1307, USA.
 *
 */

#ifndef _UAPI_DVBVIDEO_H_
#define _UAPI_DVBVIDEO_H_

#include <linux/types.h>
#ifndef __KERNEL__
#include <time.h>
#endif

typedef enum {
        VIDEO_FORMAT_4_3,     /* Select 4:3 format */
        VIDEO_FORMAT_16_9,    /* Select 16:9 format. */
        VIDEO_FORMAT_221_1    /* 2.21:1 */
} video_format_t;

typedef enum {
        VIDEO_PAN_SCAN,       /* use pan and scan format */
        VIDEO_LETTER_BOX,     /* use letterbox format */
        VIDEO_CENTER_CUT_OUT  /* use center cut out format */
} video_displayformat_t;
```

```
typedef struct {
        int w;
        int h;
        video_format_t aspect_ratio;
} video_size_t;

typedef enum {
        VIDEO_SOURCE_DEMUX, /* Select the demux as the main source␣
↪*/
        VIDEO_SOURCE_MEMORY /* If this source is selected, the␣
↪stream
                               comes from the user through the write
                               system call */
} video_stream_source_t;

typedef enum {
        VIDEO_STOPPED, /* Video is stopped */
        VIDEO_PLAYING, /* Video is currently playing */
        VIDEO_FREEZED  /* Video is freezed */
} video_play_state_t;

/* Decoder commands */
#define VIDEO_CMD_PLAY        (0)
#define VIDEO_CMD_STOP        (1)
#define VIDEO_CMD_FREEZE      (2)
#define VIDEO_CMD_CONTINUE    (3)

/* Flags for VIDEO_CMD_FREEZE */
#define VIDEO_CMD_FREEZE_TO_BLACK       (1 << 0)

/* Flags for VIDEO_CMD_STOP */
#define VIDEO_CMD_STOP_TO_BLACK         (1 << 0)
#define VIDEO_CMD_STOP_IMMEDIATELY      (1 << 1)

/* Play input formats: */
/* The decoder has no special format requirements */
#define VIDEO_PLAY_FMT_NONE         (0)
/* The decoder requires full GOPs */
#define VIDEO_PLAY_FMT_GOP          (1)

/* The structure must be zeroed before use by the application
   This ensures it can be extended safely in the future. */
struct video_command {
        __u32 cmd;
        __u32 flags;
        union {
                struct {
                        __u64 pts;
                } stop;

                struct {
```

```
                        /* 0 or 1000 specifies normal speed,
                           1 specifies forward single stepping,
                           -1 specifies backward single stepping,
                           >1: playback at speed/1000 of the normal␣
↪speed,
                           <-1: reverse playback at (-speed/1000)␣
↪of the normal speed. */
                        __s32 speed;
                        __u32 format;
                } play;

                struct {
                        __u32 data[16];
                } raw;
        };
};

/* FIELD_UNKNOWN can be used if the hardware does not know whether
   the Vsync is for an odd, even or progressive (i.e.␣
↪non-interlaced)
   field. */
#define VIDEO_VSYNC_FIELD_UNKNOWN       (0)
#define VIDEO_VSYNC_FIELD_ODD           (1)
#define VIDEO_VSYNC_FIELD_EVEN          (2)
#define VIDEO_VSYNC_FIELD_PROGRESSIVE   (3)

struct video_event {
        __s32 type;
#define VIDEO_EVENT_SIZE_CHANGED        1
#define VIDEO_EVENT_FRAME_RATE_CHANGED  2
#define VIDEO_EVENT_DECODER_STOPPED     3
#define VIDEO_EVENT_VSYNC               4
        /* unused, make sure to use atomic time for y2038 if it␣
↪ever gets used */
        long timestamp;
        union {
                video_size_t size;
                unsigned int frame_rate;        /* in frames per␣
↪1000sec */
                unsigned char vsync_field;      /* unknown/odd/even/
↪progressive */
        } u;
};

struct video_status {
        int                     video_blank;   /* blank video on␣
↪freeze? */
        video_play_state_t      play_state;    /* current state of␣
↪playback */
        video_stream_source_t stream_source; /* current source␣
↪(demux/memory) */
```

```
        video_format_t       video_format;  /* current aspect␣
↪ratio of stream*/
        video_displayformat_t display_format;/* selected cropping␣
↪mode */
};

struct video_still_picture {
        char __user *iFrame;        /* pointer to a single iframe␣
↪in memory */
        __s32 size;
};

typedef __u16 video_attributes_t;
/*   bits: descr. */
/*   15-14 Video compression mode (0=MPEG-1, 1=MPEG-2) */
/*   13-12 TV system (0=525/60, 1=625/50) */
/*   11-10 Aspect ratio (0=4:3, 3=16:9) */
/*    9- 8 permitted display mode on 4:3 monitor (0=both, 1=only␣
↪pan-sca */
/*    7    line 21-1 data present in GOP (1=yes, 0=no) */
/*    6    line 21-2 data present in GOP (1=yes, 0=no) */
/*    5- 3 source resolution (0=720x480/576, 1=704x480/576,␣
↪2=352x480/57 */
/*    2    source letterboxed (1=yes, 0=no) */
/*    0    film/camera mode (0=
 *camera, 1=film (625/50 only)) */

/* bit definitions for capabilities: */
/* can the hardware decode MPEG1 and/or MPEG2? */
#define VIDEO_CAP_MPEG1   1
#define VIDEO_CAP_MPEG2   2
/* can you send a system and/or program stream to video device?
   (you still have to open the video and the audio device but only
    send the stream to the video device) */
#define VIDEO_CAP_SYS     4
#define VIDEO_CAP_PROG    8
/* can the driver also handle SPU, NAVI and CSS encoded data?
   (CSS API is not present yet) */
#define VIDEO_CAP_SPU    16
#define VIDEO_CAP_NAVI   32
#define VIDEO_CAP_CSS    64

#define VIDEO_STOP                 _IO('o', 21)
#define VIDEO_PLAY                 _IO('o', 22)
#define VIDEO_FREEZE               _IO('o', 23)
#define VIDEO_CONTINUE             _IO('o', 24)
#define VIDEO_SELECT_SOURCE        _IO('o', 25)
#define VIDEO_SET_BLANK            _IO('o', 26)
#define VIDEO_GET_STATUS           _IOR('o', 27, struct video_
↪status)
#define VIDEO_GET_EVENT            _IOR('o', 28, struct video_event)
```

```
#define VIDEO_SET_DISPLAY_FORMAT   _IO('o', 29)
#define VIDEO_STILLPICTURE         _IOW('o', 30, struct video_still_
↪picture)
#define VIDEO_FAST_FORWARD         _IO('o', 31)
#define VIDEO_SLOWMOTION           _IO('o', 32)
#define VIDEO_GET_CAPABILITIES     _IOR('o', 33, unsigned int)
#define VIDEO_CLEAR_BUFFER         _IO('o',  34)
#define VIDEO_SET_STREAMTYPE       _IO('o', 36)
#define VIDEO_SET_FORMAT           _IO('o', 37)
#define VIDEO_GET_SIZE             _IOR('o', 55, video_size_t)

/**
 * VIDEO_GET_PTS
 *
 * Read the 33 bit presentation time stamp as defined
 * in ITU T-REC-H.222.0 / ISO/IEC 13818-1.
 *
 * The PTS should belong to the currently played
 * frame if possible, but may also be a value close to it
 * like the PTS of the last decoded frame or the last PTS
 * extracted by the PES parser.
 */
#define VIDEO_GET_PTS              _IOR('o', 57, __u64)

/* Read the number of displayed frames since the decoder was␣
↪started */
#define VIDEO_GET_FRAME_COUNT      _IOR('o', 58, __u64)

#define VIDEO_COMMAND              _IOWR('o', 59, struct video_
↪command)
#define VIDEO_TRY_COMMAND          _IOWR('o', 60, struct video_
↪command)

#endif /* _UAPI_DVBVIDEO_H_ */
```

8.3.9 Revision and Copyright

Authors:

- J. K. Metzler, Ralph <rjkm@metzlerbros.de>
- Original author of the Digital TV API documentation.
- O. C. Metzler, Marcus <rjkm@metzlerbros.de>
- Original author of the Digital TV API documentation.
- Carvalho Chehab, Mauro <mchehab+samsung@kernel.org>
- Ported document to Docbook XML, addition of DVBv5 API, documentation gaps fix.

8.3.10 Revision History

revision
2.2.0 / 2017-09-01 (*mcc*)

Most gaps between the uAPI document and the Kernel implementation got fixed for the non-legacy API.

revision
2.1.0 / 2015-05-29 (*mcc*)

DocBook improvements and cleanups, in order to document the system calls on a more standard way and provide more description about the current Digital TV API.

revision
2.0.4 / 2011-05-06 (*mcc*)

Add more information about DVBv5 API, better describing the frontend GET/SET props ioctl's.

revision
2.0.3 / 2010-07-03 (*mcc*)

Add some frontend capabilities flags, present on kernel, but missing at the specs.

revision
2.0.2 / 2009-10-25 (*mcc*)

documents FE_SET_FRONTEND_TUNE_MODE and FE_DISHETWORK_SEND_LEGACY_CMD ioctls.

revision
2.0.1 / 2009-09-16 (*mcc*)

Added ISDB-T test originally written by Patrick Boettcher

revision
2.0.0 / 2009-09-06 (*mcc*)

Conversion from LaTex to DocBook XML. The contents is the same as the original LaTex version.

revision
1.0.0 / 2003-07-24 (*rjkm*)

Initial revision on LaTEX.

8.4 Part III - Remote Controller API

8.4.1 Introduction

Currently, most analog and digital devices have a Infrared input for remote controllers. Each manufacturer has their own type of control. It is not rare for the same manufacturer to ship different types of controls, depending on the device.

A Remote Controller interface is mapped as a normal evdev/input interface, just like a keyboard or a mouse. So, it uses all ioctls already defined for any other input devices.

However, remove controllers are more flexible than a normal input device, as the IR receiver (and/or transmitter) can be used in conjunction with a wide variety of different IR remotes.

In order to allow flexibility, the Remote Controller subsystem allows controlling the RC-specific attributes via *the sysfs class nodes*.

8.4.2 Remote Controller's sysfs nodes

As defined at `Documentation/ABI/testing/sysfs-class-rc`, those are the sysfs nodes that control the Remote Controllers:

/sys/class/rc/

The `/sys/class/rc/` class sub-directory belongs to the Remote Controller core and provides a sysfs interface for configuring infrared remote controller receivers.

/sys/class/rc/rcN/

A `/sys/class/rc/rcN` directory is created for each remote control receiver device where N is the number of the receiver.

/sys/class/rc/rcN/protocols

Reading this file returns a list of available protocols, something like:

```
rc5 [rc6] nec jvc [sony]
```

Enabled protocols are shown in [] brackets.

Writing "+proto" will add a protocol to the list of enabled protocols.

Writing "-proto" will remove a protocol from the list of enabled protocols.

Writing "proto" will enable only "proto".

Writing "none" will disable all protocols.

Write fails with `EINVAL` if an invalid protocol combination or unknown protocol name is used.

/sys/class/rc/rcN/filter

Sets the scancode filter expected value.

Use in combination with `/sys/class/rc/rcN/filter_mask` to set the expected value of the bits set in the filter mask. If the hardware supports it then scancodes which do not match the filter will be ignored. Otherwise the write will fail with an error.

This value may be reset to 0 if the current protocol is altered.

/sys/class/rc/rcN/filter_mask

Sets the scancode filter mask of bits to compare. Use in combination with `/sys/class/rc/rcN/filter` to set the bits of the scancode which should be compared against the expected value. A value of 0 disables the filter to allow all valid scancodes to be processed.

If the hardware supports it then scancodes which do not match the filter will be ignored. Otherwise the write will fail with an error.

This value may be reset to 0 if the current protocol is altered.

/sys/class/rc/rcN/wakeup_protocols

Reading this file returns a list of available protocols to use for the wakeup filter, something like:

```
rc-5 nec nec-x rc-6-0 rc-6-6a-24 [rc-6-6a-32] rc-6-mce
```

Note that protocol variants are listed, so `nec`, `sony`, `rc-5`, `rc-6` have their different bit length encodings listed if available.

Note that all protocol variants are listed.

The enabled wakeup protocol is shown in [] brackets.

Only one protocol can be selected at a time.

Writing "proto" will use "proto" for wakeup events.

Writing "none" will disable wakeup.

Write fails with `EINVAL` if an invalid protocol combination or unknown protocol name is used, or if wakeup is not supported by the hardware.

/sys/class/rc/rcN/wakeup_filter

Sets the scancode wakeup filter expected value. Use in combination with `/sys/class/rc/rcN/wakeup_filter_mask` to set the expected value of the bits set in the wakeup filter mask to trigger a system wake event.

If the hardware supports it and wakeup_filter_mask is not 0 then scancodes which match the filter will wake the system from e.g. suspend to RAM or power off. Otherwise the write will fail with an error.

This value may be reset to 0 if the wakeup protocol is altered.

/sys/class/rc/rcN/wakeup_filter_mask

Sets the scancode wakeup filter mask of bits to compare. Use in combination with `/sys/class/rc/rcN/wakeup_filter` to set the bits of the scancode which should be compared against the expected value to trigger a system wake event.

If the hardware supports it and wakeup_filter_mask is not 0 then scancodes which match the filter will wake the system from e.g. suspend to RAM or power off. Otherwise the write will fail with an error.

This value may be reset to 0 if the wakeup protocol is altered.

8.4.3 Remote Controller Protocols and Scancodes

IR is encoded as a series of pulses and spaces, using a protocol. These protocols can encode e.g. an address (which device should respond) and a command: what it should do. The values for these are not always consistent across different devices for a given protocol.

Therefore out the output of the IR decoder is a scancode; a single u32 value. Using keymap tables this can be mapped to linux key codes.

Other things can be encoded too. Some IR protocols encode a toggle bit; this is to distinguish whether the same button is being held down, or has been released and pressed again. If has been released and pressed again, the toggle bit will invert from one IR message to the next.

Some remotes have a pointer-type device which can used to control the mouse; some air conditioning systems can have their target temperature target set in IR.

The following are the protocols the kernel knows about and also lists how scancodes are encoded for each protocol.

rc-5 (RC_PROTO_RC5)

This IR protocol uses manchester encoding to encode 14 bits. There is a detailed description here https://www.sbprojects.net/knowledge/ir/rc5.php.

The scancode encoding is *not* consistent with the lirc daemon (lircd) rc5 protocol, or the manchester BPF decoder.

Table 227: rc5 bits scancode mapping

rc-5 bit	scancode bit	description
1	none	Start bit, always set
1	6 (inverted)	2nd start bit in rc5, re-used as 6th command bit
1	none	Toggle bit
5	8 to 13	Address
6	0 to 5	Command

There is a variant of rc5 called either rc5x or extended rc5 where there the second stop bit is the 6th commmand bit, but inverted. This is done so it the scancodes and encoding is compatible with existing schemes. This bit is stored in bit 6 of the scancode, inverted. This is done to keep it compatible with plain rc-5 where there are two start bits.

rc-5-sz (RC_PROTO_RC5_SZ)

This is much like rc-5 but one bit longer. The scancode is encoded differently.

Table 228: rc-5-sz bits scancode mapping

rc-5-sz bits	scancode bit	description
1	none	Start bit, always set
1	13	Address bit
1	none	Toggle bit
6	6 to 11	Address
6	0 to 5	Command

rc-5x-20 (RC_PROTO_RC5X_20)

This rc-5 extended to encoded 20 bits. The is a 3555 microseconds space after the 8th bit.

Table 229: rc-5x-20 bits scancode mapping

rc-5-sz bits	scancode bit	description
1	none	Start bit, always set
1	14	Address bit
1	none	Toggle bit
5	16 to 20	Address
6	8 to 13	Address
6	0 to 5	Command

jvc (RC_PROTO_JVC)

The jvc protocol is much like nec, without the inverted values. It is described here https://www.sbprojects.net/knowledge/ir/jvc.php.

The scancode is a 16 bits value, where the address is the lower 8 bits and the command the higher 8 bits; this is reversed from IR order.

sony-12 (RC_PROTO_SONY12)

The sony protocol is a pulse-width encoding. There are three variants, which just differ in number of bits and scancode encoding.

Table 230: sony-12 bits scancode mapping

sony-12 bits	scancode bit	description
5	16 to 20	device
7	0 to 6	function

sony-15 (RC_PROTO_SONY15)

The sony protocol is a pulse-width encoding. There are three variants, which just differ in number of bits and scancode encoding.

Table 231: sony-12 bits scancode mapping

sony-12 bits	scancode bit	description
8	16 to 23	device
7	0 to 6	function

sony-20 (RC_PROTO_SONY20)

The sony protocol is a pulse-width encoding. There are three variants, which just differ in number of bits and scancode encoding.

Table 232: sony-20 bits scancode mapping

sony-20 bits	scancode bit	description
5	16 to 20	device
7	0 to 7	device
8	8 to 15	extended bits

nec (RC_PROTO_NEC)

The nec protocol encodes an 8 bit address and an 8 bit command. It is described here https://www.sbprojects.net/knowledge/ir/nec.php. Note that the protocol sends least significant bit first.

As a check, the nec protocol sends the address and command twice; the second time it is inverted. This is done for verification.

A plain nec IR message has 16 bits; the high 8 bits are the address and the low 8 bits are the command.

nec-x (RC_PROTO_NECX)

Extended nec has a 16 bit address and a 8 bit command. This is encoded as a 24 bit value as you would expect, with the lower 8 bits the command and the upper 16 bits the address.

nec-32 (RC_PROTO_NEC32)

nec-32 does not send an inverted address or an inverted command; the entire message, all 32 bits, are used.

For this to be decoded correctly, the second 8 bits must not be the inverted value of the first, and also the last 8 bits must not be the inverted value of the third 8 bit value.

The scancode has a somewhat unusual encoding.

Table 233: nec-32 bits scancode mapping

nec-32 bits	scancode bit
First 8 bits	16 to 23
Second 8 bits	24 to 31
Third 8 bits	0 to 7
Fourth 8 bits	8 to 15

sanyo (RC_PROTO_SANYO)

The sanyo protocol is like the nec protocol, but with 13 bits address rather than 8 bits. Both the address and the command are followed by their inverted versions, but these are not present in the scancodes.

Bis 8 to 20 of the scancode is the 13 bits address, and the lower 8 bits are the command.

mcir2-kbd (RC_PROTO_MCIR2_KBD)

This protocol is generated by the Microsoft MCE keyboard for keyboard events. Refer to the ir-mce_kbd-decoder.c to see how it is encoded.

mcir2-mse (RC_PROTO_MCIR2_MSE)

This protocol is generated by the Microsoft MCE keyboard for pointer events. Refer to the ir-mce_kbd-decoder.c to see how it is encoded.

rc-6-0 (RC_PROTO_RC6_0)

This is the rc-6 in mode 0. rc-6 is described here https://www.sbprojects.net/knowledge/ir/rc6.php. The scancode is the exact 16 bits as in the protocol. There is also a toggle bit.

rc-6-6a-20 (RC_PROTO_RC6_6A_20)

This is the rc-6 in mode 6a, 20 bits. rc-6 is described here https://www.sbprojects.net/knowledge/ir/rc6.php. The scancode is the exact 20 bits as in the protocol. There is also a toggle bit.

rc-6-6a-24 (RC_PROTO_RC6_6A_24)

This is the rc-6 in mode 6a, 24 bits. rc-6 is described here https://www.sbprojects.net/knowledge/ir/rc6.php. The scancode is the exact 24 bits as in the protocol. There is also a toggle bit.

rc-6-6a-32 (RC_PROTO_RC6_6A_32)

This is the rc-6 in mode 6a, 32 bits. rc-6 is described here https://www.sbprojects.net/knowledge/ir/rc6.php. The upper 16 bits are the vendor, and the lower 16 bits are the vendor-specific bits. This protocol is for the non-Microsoft MCE variant (vendor != 0x800f).

rc-6-mce (RC_PROTO_RC6_MCE)

This is the rc-6 in mode 6a, 32 bits. The upper 16 bits are the vendor, and the lower 16 bits are the vendor-specific bits. This protocol is for the Microsoft MCE variant (vendor = 0x800f). The toggle bit in the protocol itself is ignored, and the 16th bit should be takes as the toggle bit.

sharp (RC_PROTO_SHARP)

This is a protocol used by Sharp VCRs, is described here https://www.sbprojects.net/knowledge/ir/sharp.php. There is a very long (40ms) space between the normal and inverted values, and some IR receivers cannot decode this.

There is a 5 bit address and a 8 bit command. In the scancode the address is in bits 8 to 12, and the command in bits 0 to 7.

xmp (RC_PROTO_XMP)

This protocol has several versions and only version 1 is supported. Refer to the decoder (ir-xmp-decoder.c) to see how it is encoded.

cec (RC_PROTO_CEC)

This is not an IR protocol, this is a protocol over CEC. The CEC infrastructure uses rc-core for handling CEC commands, so that they can easily be remapped.

imon (RC_PROTO_IMON)

This protocol is used by Antec Veris/SoundGraph iMON remotes.

The protocol describes both button presses and pointer movements. The protocol encodes 31 bits, and the scancode is simply the 31 bits with the top bit always 0.

rc-mm-12 (RC_PROTO_RCMM12)

The rc-mm protocol is described here https://www.sbprojects.net/knowledge/ir/rcmm.php. The scancode is simply the 12 bits.

rc-mm-24 (RC_PROTO_RCMM24)

The rc-mm protocol is described here https://www.sbprojects.net/knowledge/ir/rcmm.php. The scancode is simply the 24 bits.

rc-mm-32 (RC_PROTO_RCMM32)

The rc-mm protocol is described here https://www.sbprojects.net/knowledge/ir/rcmm.php. The scancode is simply the 32 bits.

xbox-dvd (RC_PROTO_XBOX_DVD)

This protocol is used by XBox DVD Remote, which was made for the original XBox. There is no in-kernel decoder or encoder for this protocol. The usb device decodes the protocol. There is a BPF decoder available in v4l-utils.

8.4.4 Remote controller tables

Unfortunately, for several years, there was no effort to create uniform IR keycodes for different devices. This caused the same IR keyname to be mapped completely differently on different IR devices. This resulted that the same IR keyname to be mapped completely different on different IR's. Due to that, V4L2 API now specifies a standard for mapping Media keys on IR.

This standard should be used by both V4L/DVB drivers and userspace applications

The modules register the remote as keyboard within the linux input layer. This means that the IR key strokes will look like normal keyboard key strokes (if CONFIG_INPUT_KEYBOARD is enabled). Using the event devices (CONFIG_INPUT_EVDEV) it is possible for applications to access the remote via /dev/input/event devices.

Table 234: IR default keymapping

Key code	Meaning	Key examples on IR
Numeric keys		
`KEY_NUMERIC_0`	Keyboard digit 0	0
`KEY_NUMERIC_1`	Keyboard digit 1	1
`KEY_NUMERIC_2`	Keyboard digit 2	2
`KEY_NUMERIC_3`	Keyboard digit 3	3
`KEY_NUMERIC_4`	Keyboard digit 4	4
`KEY_NUMERIC_5`	Keyboard digit 5	5
`KEY_NUMERIC_6`	Keyboard digit 6	6
`KEY_NUMERIC_7`	Keyboard digit 7	7
`KEY_NUMERIC_8`	Keyboard digit 8	8
`KEY_NUMERIC_9`	Keyboard digit 9	9
Movie play control		
`KEY_FORWARD`	Instantly advance in time	>> / FORWARD
`KEY_BACK`	Instantly go back in time	<<< / BACK
`KEY_FASTFORWARD`	Play movie faster	`>>> / FORWARD`
`KEY_REWIND`	Play movie back	REWIND / BACKWARD
`KEY_NEXT`	Select next chapter / sub-chapter / interval	NEXT / SKIP
`KEY_PREVIOUS`	Select previous chapter / sub-chapter / interval	<< / PREV / PREVIOUS

continues on next page

Table 234 – continued from previous page

KEY_AGAIN	Repeat the video or a video interval	REPEAT / LOOP / RECALL
KEY_PAUSE	Pause stream	PAUSE / FREEZE
KEY_PLAY	Play movie at the normal timeshift	NORMAL TIMESHIFT / LIVE / >
KEY_PLAYPAUSE	Alternate between play and pause	PLAY / PAUSE
KEY_STOP	Stop stream	STOP
KEY_RECORD	Start/stop recording stream	CAPTURE / REC / RECORD/PAUSE
KEY_CAMERA	Take a picture of the image	CAMERA ICON / CAPTURE / SNAPSHOT
KEY_SHUFFLE	Enable shuffle mode	SHUFFLE
KEY_TIME	Activate time shift mode	TIME SHIFT
KEY_TITLE	Allow changing the chapter	CHAPTER
KEY_SUBTITLE	Allow changing the subtitle	SUBTITLE
Image control		
KEY_BRIGHTNESSDOWN	Decrease Brightness	BRIGHTNESS DECREASE
KEY_BRIGHTNESSUP	Increase Brightness	BRIGHTNESS INCREASE
KEY_ANGLE	Switch video camera angle (on videos with more than one angle stored)	ANGLE / SWAP
KEY_EPG	Open the Elecrowonic Play Guide (EPG)	EPG / GUIDE
KEY_TEXT	Activate/change closed caption mode	CLOSED CAPTION/TELETEXT / DVD TEXT / TELETEXT / TTX
Audio control		
KEY_AUDIO	Change audio source	AUDIO SOURCE / AUDIO / MUSIC
KEY_MUTE	Mute/unmute audio	MUTE / DEMUTE / UNMUTE
KEY_VOLUMEDOWN	Decrease volume	VOLUME- / VOLUME DOWN
KEY_VOLUMEUP	Increase volume	VOLUME+ / VOLUME UP
KEY_MODE	Change sound mode	MONO/STEREO
KEY_LANGUAGE	Select Language	1ST / 2ND LANGUAGE / DVD LANG / MTS/SAP / MTS SEL
Channel control		
KEY_CHANNEL	Go to the next favorite channel	ALT / CHANNEL / CH SURFING / SURF / FAV
KEY_CHANNELDOWN	Decrease channel sequentially	CHANNEL - / CHANNEL DOWN / DOWN
KEY_CHANNELUP	Increase channel sequentially	CHANNEL + / CHANNEL UP / UP
KEY_DIGITS	Use more than one digit for channel	PLUS / 100/ 1xx / xxx / -/– / Single Double Triple Digit
KEY_SEARCH	Start channel autoscan	SCAN / AUTOSCAN

continues on next page

Table 234 – continued from previous page

Colored keys		
KEY_BLUE	IR Blue key	BLUE
KEY_GREEN	IR Green Key	GREEN
KEY_RED	IR Red key	RED
KEY_YELLOW	IR Yellow key	YELLOW
Media selection		
KEY_CD	Change input source to Compact Disc	CD
KEY_DVD	Change input to DVD	DVD / DVD MENU
KEY_EJECTCLOSECD	Open/close the CD/DVD player	->) / CLOSE / OPEN
KEY_MEDIA	Turn on/off Media application	PC/TV / TURN ON/OFF APP
KEY_PC	Selects from TV to PC	PC
KEY_RADIO	Put into AM/FM radio mode	RADIO / TV/FM / TV/RADIO / FM / FM/RADIO
KEY_TV	Select tv mode	TV / LIVE TV
KEY_TV2	Select Cable mode	AIR/CBL
KEY_VCR	Select VCR mode	VCR MODE / DTR
KEY_VIDEO	Alternate between input modes	SOURCE / SELECT / DISPLAY / SWITCH INPUTS / VIDEO
Power control		
KEY_POWER	Turn on/off computer	SYSTEM POWER / COMPUTER POWER
KEY_POWER2	Turn on/off application	TV ON/OFF / POWER
KEY_SLEEP	Activate sleep timer	SLEEP / SLEEP TIMER
KEY_SUSPEND	Put computer into suspend mode	STANDBY / SUSPEND
Window control		
KEY_CLEAR	Stop stream and return to default input video/audio	CLEAR / RESET / BOSS KEY
KEY_CYCLEWINDOWS	Minimize windows and move to the next one	ALT-TAB / MINIMIZE / DESKTOP
KEY_FAVORITES	Open the favorites stream window	TV WALL / Favorites
KEY_MENU	Call application menu	2ND CONTROLS (USA: MENU) / DVD/MENU / SHOW/HIDE CTRL
KEY_NEW	Open/Close Picture in Picture	PIP
KEY_OK	Send a confirmation code to application	OK / ENTER / RETURN
KEY_ASPECT_RATIO	Select screen aspect ratio	4:3 16:9 SELECT
KEY_FULL_SCREEN	Put device into zoom/full screen mode	ZOOM / FULL SCREEN / ZOOM+ / HIDE PANNEL / SWITCH
Navigation keys		
KEY_ESC	Cancel current operation	CANCEL / BACK
KEY_HELP	Open a Help window	HELP

continues on next page

Table 234 – continued from previous page

KEY_HOMEPAGE	Navigate to Home-page	HOME
KEY_INFO	Open On Screen Display	DISPLAY INFORMATION / OSD
KEY_WWW	Open the default browser	WEB
KEY_UP	Up key	UP
KEY_DOWN	Down key	DOWN
KEY_LEFT	Left key	LEFT
KEY_RIGHT	Right key	RIGHT
Miscellaneous keys		
KEY_DOT	Return a dot	.
KEY_FN	Select a function	FUNCTION

It should be noted that, sometimes, there some fundamental missing keys at some cheaper IR' s. Due to that, it is recommended to:

Table 235: Notes

On simpler IR' s, without separate channel keys, you need to map UP as KEY_CHANNELUP
On simpler IR' s, without separate channel keys, you need to map DOWN as KEY_CHANNELDOWN
On simpler IR' s, without separate volume keys, you need to map LEFT as KEY_VOLUMEDOWN
On simpler IR' s, without separate volume keys, you need to map RIGHT as KEY_VOLUMEUP

8.4.5 Changing default Remote Controller mappings

The event interface provides two ioctls to be used against the /dev/input/event device, to allow changing the default keymapping.

This program demonstrates how to replace the keymap tables.

file: uapi/v4l/keytable.c

```
/* keytable.c - This program allows checking/replacing keys at IR

   Copyright (C) 2006-2009 Mauro Carvalho Chehab <mchehab@kernel.
↪org>

   This program is free software; you can redistribute it and/or␣
↪modify
   it under the terms of the GNU General Public License as␣
↪published by
   the Free Software Foundation, version 2 of the License.

```

(continues on next page)

(continued from previous page)

```
   This program is distributed in the hope that it will be useful,
   but WITHOUT ANY WARRANTY; without even the implied warranty of
   MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
   GNU General Public License for more details.
 */

#include <ctype.h>
#include <errno.h>
#include <fcntl.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <linux/input.h>
#include <sys/ioctl.h>

#include "parse.h"

void prtcode (int *codes)
{
        struct parse_key *p;

        for (p=keynames;p->name!=NULL;p++) {
                if (p->value == (unsigned)codes[1]) {
                        printf("scancode 0x%04x = %s (0x%02x)\\n",␣
↪codes[0], p->name, codes[1]);
                        return;
                }
        }

        if (isprint (codes[1]))
                printf("scancode %d = '%c' (0x%02x)\\n", codes[0],␣
↪codes[1], codes[1]);
        else
                printf("scancode %d = 0x%02x\\n", codes[0],␣
↪codes[1]);
}

int parse_code(char *string)
{
        struct parse_key *p;

        for (p=keynames;p->name!=NULL;p++) {
                if (!strcasecmp(p->name, string)) {
                        return p->value;
                }
        }
        return -1;
}
```

(continues on next page)

(continued from previous page)

```
int main (int argc, char *argv[])
{
        int fd;
        unsigned int i, j;
        int codes[2];

        if (argc<2 || argc>4) {
                printf ("usage: %s <device> to get table; or\\n"
                        "       %s <device> <scancode> <keycode>\\n"
                        "       %s <device> <keycode_file>n",*argv,
↪*argv,*argv);
                return -1;
        }

        if ((fd = open(argv[1], O_RDONLY)) < 0) {
                perror("Couldn't open input device");
                return(-1);
        }

        if (argc==4) {
                int value;

                value=parse_code(argv[3]);

                if (value==-1) {
                        value = strtol(argv[3], NULL, 0);
                        if (errno)
                                perror("value");
                }

                codes [0] = (unsigned) strtol(argv[2], NULL, 0);
                codes [1] = (unsigned) value;

                if(ioctl(fd, EVIOCSKEYCODE, codes))
                        perror ("EVIOCSKEYCODE");

                if(ioctl(fd, EVIOCGKEYCODE, codes)==0)
                        prtcode(codes);
                return 0;
        }

        if (argc==3) {
                FILE *fin;
                int value;
                char *scancode, *keycode, s[2048];

                fin=fopen(argv[2],"r");
                if (fin==NULL) {
                        perror ("opening keycode file");
```

(continues on next page)

(continued from previous page)

```
                        return -1;
                }

                /* Clears old table */
                for (j = 0; j < 256; j++) {
                        for (i = 0; i < 256; i++) {
                                codes[0] = (j << 8) | i;
                                codes[1] = KEY_RESERVED;
                                ioctl(fd, EVIOCSKEYCODE, codes);
                        }
                }

                while (fgets(s,sizeof(s),fin)) {
                        scancode=strtok(s,"\\n\\t =:");
                        if (!scancode) {
                                perror ("parsing input file scancode
↪");
                                return -1;
                        }
                        if (!strcasecmp(scancode, "scancode")) {
                                scancode = strtok(NULL,"\\n\\t =:");
                                if (!scancode) {
                                        perror ("parsing input file␣
↪scancode");
                                        return -1;
                                }
                        }

                        keycode=strtok(NULL,"\\n\\t =:(");
                        if (!keycode) {
                                perror ("parsing input file keycode
↪");
                                return -1;
                        }

                        // printf ("parsing %s=%s:", scancode,␣
↪keycode);
                        value=parse_code(keycode);
                        // printf ("\\tvalue=%d\\n",value);

                        if (value==-1) {
                                value = strtol(keycode, NULL, 0);
                                if (errno)
                                        perror("value");
                        }

                        codes [0] = (unsigned) strtol(scancode,␣
↪NULL, 0);
                        codes [1] = (unsigned) value;
```

(continues on next page)

(continued from previous page)

```
                        // printf("\\t%04x=%04x\\n",codes[0],␣
↪codes[1]);
                        if(ioctl(fd, EVIOCSKEYCODE, codes)) {
                                fprintf(stderr, "Setting scancode 0x
↪%04x with 0x%04x via ",codes[0], codes[1]);
                                perror ("EVIOCSKEYCODE");
                        }

                        if(ioctl(fd, EVIOCGKEYCODE, codes)==0)
                                prtcode(codes);
                }
                return 0;
        }

        /* Get scancode table */
        for (j = 0; j < 256; j++) {
                for (i = 0; i < 256; i++) {
                        codes[0] = (j << 8) | i;
                        if (!ioctl(fd, EVIOCGKEYCODE, codes) &&␣
↪codes[1] != KEY_RESERVED)
                                prtcode(codes);
                }
        }
        return 0;
}
```

8.4.6 LIRC Device Interface

Introduction

LIRC stands for Linux Infrared Remote Control. The LIRC device interface is a bi-directional interface for transporting raw IR and decoded scancodes data between userspace and kernelspace. Fundamentally, it is just a chardev (/dev/lircX, for X = 0, 1, 2, ⋯), with a number of standard struct file_operations defined on it. With respect to transporting raw IR and decoded scancodes to and fro, the essential fops are read, write and ioctl.

It is also possible to attach a BPF program to a LIRC device for decoding raw IR into scancodes.

Example dmesg output upon a driver registering w/LIRC:

```
$ dmesg |grep lirc_dev
rc rc0: lirc_dev: driver mceusb registered at minor = 0, raw IR␣
↪receiver, raw IR transmitter
```

What you should see for a chardev:

```
$ ls -l /dev/lirc*
crw-rw---- 1 root root 248, 0 Jul 2 22:20 /dev/lirc0
```

Note that the package v4l-utils contains tools for working with LIRC devices:

- ir-ctl: can receive raw IR and transmit IR, as well as query LIRC device features.
- ir-keytable: can load keymaps; allows you to set IR kernel protocols; load BPF IR decoders and test IR decoding. Some BPF IR decoders are also provided.

LIRC modes

LIRC supports some modes of receiving and sending IR codes, as shown on the following table.

`LIRC_MODE_SCANCODE`

This mode is for both sending and receiving IR.

For transmitting (aka sending), create a `struct lirc_scancode` with the desired scancode set in the `scancode` member, `rc_proto` set to the *IR protocol*, and all other members set to 0. Write this struct to the lirc device.

For receiving, you read `struct lirc_scancode` from the LIRC device. The `scancode` field is set to the received scancode and the *IR protocol* is set in `rc_proto`. If the scancode maps to a valid key code, this is set in the `keycode` field, else it is set to `KEY_RESERVED`.

The `flags` can have `LIRC_SCANCODE_FLAG_TOGGLE` set if the toggle bit is set in protocols that support it (e.g. rc-5 and rc-6), or `LIRC_SCANCODE_FLAG_REPEAT` for when a repeat is received for protocols that support it (e.g. nec).

In the Sanyo and NEC protocol, if you hold a button on remote, rather than repeating the entire scancode, the remote sends a shorter message with no scancode, which just means button is held, a "repeat". When this is received, the `LIRC_SCANCODE_FLAG_REPEAT` is set and the scancode and keycode is repeated.

With nec, there is no way to distinguish "button hold" from "repeatedly pressing the same button". The rc-5 and rc-6 protocols have a toggle bit. When a button is released and pressed again, the toggle bit is inverted. If the toggle bit is set, the `LIRC_SCANCODE_FLAG_TOGGLE` is set.

The `timestamp` field is filled with the time nanoseconds (in `CLOCK_MONOTONIC`) when the scancode was decoded.

`LIRC_MODE_MODE2`

The driver returns a sequence of pulse and space codes to userspace, as a series of u32 values.

This mode is used only for IR receive.

The upper 8 bits determine the packet type, and the lower 24 bits the payload. Use `LIRC_VALUE()` macro to get the payload, and the macro `LIRC_MODE2()` will give you the type, which is one of:

`LIRC_MODE2_PULSE`

Signifies the presence of IR in microseconds.

`LIRC_MODE2_SPACE`

Signifies absence of IR in microseconds.

`LIRC_MODE2_FREQUENCY`

If measurement of the carrier frequency was enabled with *ioctl LIRC_SET_MEASURE_CARRIER_MODE* then this packet gives you the carrier frequency in Hertz.

`LIRC_MODE2_TIMEOUT`

If timeout reports are enabled with *ioctl LIRC_SET_REC_TIMEOUT_REPORTS*, when the timeout set with *ioctl LIRC_GET_REC_TIMEOUT and LIRC_SET_REC_TIMEOUT* expires due to no IR being detected, this packet will be sent, with the number of microseconds with no IR.

`LIRC_MODE_PULSE`

In pulse mode, a sequence of pulse/space integer values are written to the lirc device using *LIRC write()*.

The values are alternating pulse and space lengths, in microseconds. The first and last entry must be a pulse, so there must be an odd number of entries.

This mode is used only for IR send.

BPF based IR decoder

The kernel has support for decoding the most common *IR protocols*, but there are many protocols which are not supported. To support these, it is possible to load an BPF program which does the decoding. This can only be done on LIRC devices which support reading raw IR.

First, using the bpf(2) syscall with the `BPF_LOAD_PROG` argument, program must be loaded of type `BPF_PROG_TYPE_LIRC_MODE2`. Once attached to the LIRC device, this program will be called for each pulse, space or timeout event on the LIRC device. The context for the BPF program is a pointer to a unsigned int, which is a *LIRC_MODE_MODE2* value. When the program has decoded the scancode, it can be submitted using the BPF functions `bpf_rc_keydown()` or `bpf_rc_repeat()`. Mouse or pointer movements can be reported using `bpf_rc_pointer_rel()`.

Once you have the file descriptor for the `BPF_PROG_TYPE_LIRC_MODE2` BPF program, it can be attached to the LIRC device using the bpf(2) syscall. The target must be the file descriptor for the LIRC device, and the attach type must be `BPF_LIRC_MODE2`. No more than 64 BPF programs can be attached to a single LIRC device at a time.

LIRC Function Reference

LIRC read()

Name

lirc-read - Read from a LIRC device

Synopsis

```
#include <unistd.h>
```

ssize_t **read**(int fd, void *buf, size_t count)

Arguments

fd
File descriptor returned by `open()`.

buf
Buffer to be filled

count
Max number of bytes to read

Description

read() attempts to read up to `count` bytes from file descriptor `fd` into the buffer starting at `buf`. If `count` is zero, *read()* returns zero and has no other results. If `count` is greater than `SSIZE_MAX`, the result is unspecified.

The exact format of the data depends on what *LIRC modes* a driver uses. Use *ioctl LIRC_GET_FEATURES* to get the supported mode, and use *ioctls LIRC_GET_REC_MODE and LIRC_SET_REC_MODE* set the current active mode.

The mode *LIRC_MODE_MODE2* is for raw IR, in which packets containing an unsigned int value describing an IR signal are read from the chardev.

Alternatively, *LIRC_MODE_SCANCODE* can be available, in this mode scancodes which are either decoded by software decoders, or by hardware decoders. The `rc_proto` member is set to the *IR protocol* used for transmission, and `scancode` to the decoded scancode, and the `keycode` set to the keycode or `KEY_RESERVED`.

Return Value

On success, the number of bytes read is returned. It is not an error if this number is smaller than the number of bytes requested, or the amount of data required for one frame. On error, -1 is returned, and the `errno` variable is set appropriately.

LIRC write()

Name

lirc-write - Write to a LIRC device

Synopsis

```
#include <unistd.h>
```

ssize_t **write**(int fd, void *buf, size_t count)

Arguments

fd
: File descriptor returned by `open()`.

buf
: Buffer with data to be written

count
: Number of bytes at the buffer

Description

write() writes up to `count` bytes to the device referenced by the file descriptor `fd` from the buffer starting at `buf`.

The exact format of the data depends on what mode a driver is in, use *ioctl LIRC_GET_FEATURES* to get the supported modes and use *ioctls LIRC_GET_SEND_MODE and LIRC_SET_SEND_MODE* set the mode.

When in *LIRC_MODE_PULSE* mode, the data written to the chardev is a pulse/space sequence of integer values. Pulses and spaces are only marked implicitly by their position. The data must start and end with a pulse, therefore, the data must always include an uneven number of samples. The write function blocks until the data has been transmitted by the hardware. If more data is provided than the hardware can send, the driver returns `EINVAL`.

When in *LIRC_MODE_SCANCODE* mode, one `struct lirc_scancode` must be written to the chardev at a time, else `EINVAL` is returned. Set the desired scancode in the `scancode` member, and the *IR protocol* in the `rc_proto`: member. All other members must be set to 0, else `EINVAL` is returned. If there is no protocol encoder for the protocol or the scancode is not valid for the specified protocol, EINVAL is returned. The write function blocks until the scancode is transmitted by the hardware.

Return Value

On success, the number of bytes written is returned. It is not an error if this number is smaller than the number of bytes requested, or the amount of data required for one frame. On error, -1 is returned, and the errno variable is set appropriately. The generic error codes are described at the *Generic Error Codes* chapter.

ioctl LIRC_GET_FEATURES

Name

LIRC_GET_FEATURES - Get the underlying hardware device's features

Synopsis

LIRC_GET_FEATURES

`int ioctl(int fd, LIRC_GET_FEATURES, __u32 *features)`

Arguments

fd
: File descriptor returned by open().

features
: Bitmask with the LIRC features.

Description

Get the underlying hardware device's features. If a driver does not announce support of certain features, calling of the corresponding ioctls is undefined.

LIRC features

`LIRC_CAN_REC_RAW`

Unused. Kept just to avoid breaking uAPI.

`LIRC_CAN_REC_PULSE`

Unused. Kept just to avoid breaking uAPI. *LIRC_MODE_PULSE* can only be used for transmitting.

`LIRC_CAN_REC_MODE2`

This is raw IR driver for receiving. This means that *LIRC_MODE_MODE2* is used. This also implies that *LIRC_MODE_SCANCODE* is also supported, as long as the kernel is recent enough. Use the *ioctls LIRC_GET_REC_MODE and LIRC_SET_REC_MODE* to switch modes.

`LIRC_CAN_REC_LIRCCODE`

Unused. Kept just to avoid breaking uAPI.

`LIRC_CAN_REC_SCANCODE`

This is a scancode driver for receiving. This means that *LIRC_MODE_SCANCODE* is used.

`LIRC_CAN_SET_SEND_CARRIER`

The driver supports changing the modulation frequency via *ioctl LIRC_SET_SEND_CARRIER*.

`LIRC_CAN_SET_SEND_DUTY_CYCLE`

The driver supports changing the duty cycle using *ioctl LIRC_SET_SEND_DUTY_CYCLE*.

`LIRC_CAN_SET_TRANSMITTER_MASK`

The driver supports changing the active transmitter(s) using *ioctl LIRC_SET_TRANSMITTER_MASK*.

`LIRC_CAN_SET_REC_CARRIER`

The driver supports setting the receive carrier frequency using *ioctl LIRC_SET_REC_CARRIER*.

`LIRC_CAN_SET_REC_DUTY_CYCLE_RANGE`

Unused. Kept just to avoid breaking uAPI.

`LIRC_CAN_SET_REC_CARRIER_RANGE`

The driver supports *ioctl LIRC_SET_REC_CARRIER_RANGE*.

`LIRC_CAN_GET_REC_RESOLUTION`

The driver supports *ioctl LIRC_GET_REC_RESOLUTION*.

`LIRC_CAN_SET_REC_TIMEOUT`

The driver supports *ioctl LIRC_SET_REC_TIMEOUT*.

`LIRC_CAN_SET_REC_FILTER`

Unused. Kept just to avoid breaking uAPI.

`LIRC_CAN_MEASURE_CARRIER`

The driver supports measuring of the modulation frequency using *ioctl LIRC_SET_MEASURE_CARRIER_MODE*.

`LIRC_CAN_USE_WIDEBAND_RECEIVER`

The driver supports learning mode using *ioctl LIRC_SET_WIDEBAND_RECEIVER*.

`LIRC_CAN_NOTIFY_DECODE`

Unused. Kept just to avoid breaking uAPI.

`LIRC_CAN_SEND_RAW`

Unused. Kept just to avoid breaking uAPI.

`LIRC_CAN_SEND_PULSE`

The driver supports sending (also called as IR blasting or IR TX) using *LIRC_MODE_PULSE*. This implies that *LIRC_MODE_SCANCODE* is also supported for transmit, as long as the kernel is recent enough. Use the *ioctls LIRC_GET_SEND_MODE and LIRC_SET_SEND_MODE* to switch modes.

`LIRC_CAN_SEND_MODE2`

Unused. Kept just to avoid breaking uAPI. *LIRC_MODE_MODE2* can only be used for receiving.

`LIRC_CAN_SEND_LIRCCODE`

Unused. Kept just to avoid breaking uAPI.

Return Value

On success 0 is returned, on error -1 and the `errno` variable is set appropriately. The generic error codes are described at the *Generic Error Codes* chapter.

ioctls LIRC_GET_SEND_MODE and LIRC_SET_SEND_MODE

Name

LIRC_GET_SEND_MODE/LIRC_SET_SEND_MODE - Get/set current transmit mode.

Synopsis

`LIRC_GET_SEND_MODE`

`int ioctl(int fd, LIRC_GET_SEND_MODE, __u32 *mode)`

`LIRC_SET_SEND_MODE`

`int ioctl(int fd, LIRC_SET_SEND_MODE, __u32 *mode)`

Arguments

fd
File descriptor returned by open().

mode
The mode used for transmitting.

Description

Get/set current transmit mode.

Only *LIRC_MODE_PULSE* and *LIRC_MODE_SCANCODE* are supported by for IR send, depending on the driver. Use *ioctl LIRC_GET_FEATURES* to find out which modes the driver supports.

Return Value

ENODEV	Device not available.
ENOTTY	Device does not support transmitting.
EINVAL	Invalid mode or invalid mode for this device.

ioctls LIRC_GET_REC_MODE and LIRC_SET_REC_MODE

Name

LIRC_GET_REC_MODE/LIRC_SET_REC_MODE - Get/set current receive mode.

Synopsis

LIRC_GET_REC_MODE

`int ioctl(int fd, LIRC_GET_REC_MODE, __u32 *mode)`

LIRC_SET_REC_MODE

`int ioctl(int fd, LIRC_SET_REC_MODE, __u32 *mode)`

Arguments

fd
File descriptor returned by open().

mode
Mode used for receive.

Description

Get and set the current receive mode. Only *LIRC_MODE_MODE2* and *LIRC_MODE_SCANCODE* are supported. Use *ioctl LIRC_GET_FEATURES* to find out which modes the driver supports.

Return Value

ENODEV	Device not available.
ENOTTY	Device does not support receiving.
EINVAL	Invalid mode or invalid mode for this device.

ioctl LIRC_GET_REC_RESOLUTION

Name

LIRC_GET_REC_RESOLUTION - Obtain the value of receive resolution, in microseconds.

Synopsis

LIRC_GET_REC_RESOLUTION

```
int ioctl(int fd, LIRC_GET_REC_RESOLUTION, __u32 *microseconds)
```

Arguments

fd
File descriptor returned by open().

microseconds
Resolution, in microseconds.

Description

Some receivers have maximum resolution which is defined by internal sample rate or data format limitations. E.g. it's common that signals can only be reported in 50 microsecond steps.

This ioctl returns the integer value with such resolution, with can be used by userspace applications like lircd to automatically adjust the tolerance value.

Return Value

On success 0 is returned, on error -1 and the `errno` variable is set appropriately. The generic error codes are described at the *Generic Error Codes* chapter.

ioctl LIRC_SET_SEND_DUTY_CYCLE

Name

LIRC_SET_SEND_DUTY_CYCLE - Set the duty cycle of the carrier signal for IR transmit.

Synopsis

`LIRC_SET_SEND_DUTY_CYCLE`

`int ioctl(int fd, LIRC_SET_SEND_DUTY_CYCLE, __u32 *duty_cycle)`

Arguments

`fd`
File descriptor returned by open().

`duty_cycle`
Duty cicle, describing the pulse width in percent (from 1 to 99) of the total cycle. Values 0 and 100 are reserved.

Description

Get/set the duty cycle of the carrier signal for IR transmit.

Currently, no special meaning is defined for 0 or 100, but this could be used to switch off carrier generation in the future, so these values should be reserved.

Return Value

On success 0 is returned, on error -1 and the `errno` variable is set appropriately. The generic error codes are described at the *Generic Error Codes* chapter.

ioctls LIRC_GET_MIN_TIMEOUT and LIRC_GET_MAX_TIMEOUT

Name

LIRC_GET_MIN_TIMEOUT / LIRC_GET_MAX_TIMEOUT - Obtain the possible timeout range for IR receive.

Synopsis

LIRC_GET_MIN_TIMEOUT

`int ioctl(int fd, LIRC_GET_MIN_TIMEOUT, __u32 *timeout)`

LIRC_GET_MAX_TIMEOUT

`int ioctl(int fd, LIRC_GET_MAX_TIMEOUT, __u32 *timeout)`

Arguments

fd
File descriptor returned by open().

timeout
Timeout, in microseconds.

Description

Some devices have internal timers that can be used to detect when there's no IR activity for a long time. This can help lircd in detecting that a IR signal is finished and can speed up the decoding process. Returns an integer value with the minimum/maximum timeout that can be set.

Note: Some devices have a fixed timeout, in that case both ioctls will return the same value even though the timeout cannot be changed via *ioctl LIRC_GET_REC_TIMEOUT and LIRC_SET_REC_TIMEOUT*.

Return Value

On success 0 is returned, on error -1 and the `errno` variable is set appropriately. The generic error codes are described at the *Generic Error Codes* chapter.

ioctl LIRC_GET_REC_TIMEOUT and LIRC_SET_REC_TIMEOUT

Name

LIRC_GET_REC_TIMEOUT/LIRC_SET_REC_TIMEOUT - Get/set the integer value for IR inactivity timeout.

Synopsis

LIRC_GET_REC_TIMEOUT

`int ioctl(int fd, LIRC_GET_REC_TIMEOUT, __u32 *timeout)`

LIRC_SET_REC_TIMEOUT

`int ioctl(int fd, LIRC_SET_REC_TIMEOUT, __u32 *timeout)`

Arguments

fd
File descriptor returned by open().

timeout
Timeout, in microseconds.

Description

Get and set the integer value for IR inactivity timeout.

If supported by the hardware, setting it to 0 disables all hardware timeouts and data should be reported as soon as possible. If the exact value cannot be set, then the next possible value _greater_ than the given value should be set.

Note: The range of supported timeout is given by *ioctls LIRC_GET_MIN_TIMEOUT and LIRC_GET_MAX_TIMEOUT*.

Return Value

On success 0 is returned, on error -1 and the `errno` variable is set appropriately. The generic error codes are described at the *Generic Error Codes* chapter.

ioctl LIRC_SET_REC_CARRIER

Name

LIRC_SET_REC_CARRIER - Set carrier used to modulate IR receive.

Synopsis

LIRC_SET_REC_CARRIER

`int ioctl(int fd, LIRC_SET_REC_CARRIER, __u32 *frequency)`

Arguments

fd
: File descriptor returned by open().

frequency
: Frequency of the carrier that modulates PWM data, in Hz.

Description

Set receive carrier used to modulate IR PWM pulses and spaces.

Note: If called together with *ioctl LIRC_SET_REC_CARRIER_RANGE*, this ioctl sets the upper bound frequency that will be recognized by the device.

Return Value

On success 0 is returned, on error -1 and the `errno` variable is set appropriately. The generic error codes are described at the *Generic Error Codes* chapter.

ioctl LIRC_SET_REC_CARRIER_RANGE

Name

LIRC_SET_REC_CARRIER_RANGE - Set lower bound of the carrier used to modulate IR receive.

Synopsis

LIRC_SET_REC_CARRIER_RANGE

`int ioctl(int fd, LIRC_SET_REC_CARRIER_RANGE, __u32 *frequency)`

Arguments

fd
: File descriptor returned by open().

frequency
: Frequency of the carrier that modulates PWM data, in Hz.

Description

This ioctl sets the upper range of carrier frequency that will be recognized by the IR receiver.

Note: To set a range use *LIRC_SET_REC_CARRIER_RANGE* with the lower bound first and later call *LIRC_SET_REC_CARRIER* with the upper bound.

Return Value

On success 0 is returned, on error -1 and the `errno` variable is set appropriately. The generic error codes are described at the *Generic Error Codes* chapter.

ioctl LIRC_SET_SEND_CARRIER

Name

LIRC_SET_SEND_CARRIER - Set send carrier used to modulate IR TX.

Synopsis

LIRC_SET_SEND_CARRIER

`int ioctl(int fd, LIRC_SET_SEND_CARRIER, __u32 *frequency)`

Arguments

fd
File descriptor returned by open().

frequency
Frequency of the carrier to be modulated, in Hz.

Description

Set send carrier used to modulate IR PWM pulses and spaces.

Return Value

On success 0 is returned, on error -1 and the `errno` variable is set appropriately. The generic error codes are described at the *Generic Error Codes* chapter.

ioctl LIRC_SET_TRANSMITTER_MASK

Name

LIRC_SET_TRANSMITTER_MASK - Enables send codes on a given set of transmitters

Synopsis

LIRC_SET_TRANSMITTER_MASK

`int ioctl(int fd, LIRC_SET_TRANSMITTER_MASK, __u32 *mask)`

Arguments

fd
File descriptor returned by open().

mask
Mask with channels to enable tx. Channel 0 is the least significant bit.

Description

Some IR TX devices have multiple output channels, in such case, *LIRC_CAN_SET_TRANSMITTER_MASK* is returned via *ioctl LIRC_GET_FEATURES* and this ioctl sets what channels will send IR codes.

This ioctl enables the given set of transmitters. The first transmitter is encoded by the least significant bit and so on.

When an invalid bit mask is given, i.e. a bit is set, even though the device does not have so many transitters, then this ioctl returns the number of available transitters and does nothing otherwise.

Return Value

On success 0 is returned, on error -1 and the `errno` variable is set appropriately. The generic error codes are described at the *Generic Error Codes* chapter.

ioctl LIRC_SET_REC_TIMEOUT_REPORTS

Name

LIRC_SET_REC_TIMEOUT_REPORTS - enable or disable timeout reports for IR receive

Synopsis

LIRC_SET_REC_TIMEOUT_REPORTS

`int ioctl(int fd, LIRC_SET_REC_TIMEOUT_REPORTS, __u32 *enable)`

Arguments

fd
File descriptor returned by open().

enable
enable = 1 means enable timeout report, enable = 0 means disable timeout reports.

Description

Enable or disable timeout reports for IR receive. By default, timeout reports should be turned off.

Note: This ioctl is only valid for *LIRC_MODE_MODE2*.

Return Value

On success 0 is returned, on error -1 and the `errno` variable is set appropriately. The generic error codes are described at the *Generic Error Codes* chapter.

ioctl LIRC_SET_MEASURE_CARRIER_MODE

Name

LIRC_SET_MEASURE_CARRIER_MODE - enable or disable measure mode

Synopsis

LIRC_SET_MEASURE_CARRIER_MODE

`int ioctl(int fd, LIRC_SET_MEASURE_CARRIER_MODE, __u32 *enable)`

Arguments

fd
: File descriptor returned by open().

enable
: enable = 1 means enable measure mode, enable = 0 means disable measure mode.

Description

Enable or disable measure mode. If enabled, from the next key press on, the driver will send `LIRC_MODE2_FREQUENCY` packets. By default this should be turned off.

Return Value

On success 0 is returned, on error -1 and the `errno` variable is set appropriately. The generic error codes are described at the *Generic Error Codes* chapter.

ioctl LIRC_SET_WIDEBAND_RECEIVER

Name

LIRC_SET_WIDEBAND_RECEIVER - enable wide band receiver.

Synopsis

LIRC_SET_WIDEBAND_RECEIVER

`int ioctl(int fd, LIRC_SET_WIDEBAND_RECEIVER, __u32 *enable)`

Arguments

fd
File descriptor returned by open().

enable
enable = 1 means enable wideband receiver, enable = 0 means disable wideband receiver.

Description

Some receivers are equipped with special wide band receiver which is intended to be used to learn output of existing remote. This ioctl allows enabling or disabling it.

This might be useful of receivers that have otherwise narrow band receiver that prevents them to be used with some remotes. Wide band receiver might also be more precise. On the other hand its disadvantage it usually reduced range of reception.

Note: Wide band receiver might be implictly enabled if you enable carrier reports. In that case it will be disabled as soon as you disable carrier reports. Trying to disable wide band receiver while carrier reports are active will do nothing.

Return Value

On success 0 is returned, on error -1 and the `errno` variable is set appropriately. The generic error codes are described at the *Generic Error Codes* chapter.

LIRC Header File

lirc.h

```
/* SPDX-License-Identifier: GPL-2.0 WITH Linux-syscall-note */
/*
 * lirc.h - linux infrared remote control header file
 * last modified 2010/07/13 by Jarod Wilson
 */

#ifndef _LINUX_LIRC_H
#define _LINUX_LIRC_H

#include <linux/types.h>
#include <linux/ioctl.h>

#define PULSE_BIT       0x01000000
#define PULSE_MASK      0x00FFFFFF

#define LIRC_MODE2_SPACE     0x00000000
#define LIRC_MODE2_PULSE     0x01000000
#define LIRC_MODE2_FREQUENCY 0x02000000
#define LIRC_MODE2_TIMEOUT   0x03000000

#define LIRC_VALUE_MASK      0x00FFFFFF
#define LIRC_MODE2_MASK      0xFF000000

#define LIRC_SPACE(val) (((val)&LIRC_VALUE_MASK) | LIRC_MODE2_SPACE)
#define LIRC_PULSE(val) (((val)&LIRC_VALUE_MASK) | LIRC_MODE2_PULSE)
#define LIRC_FREQUENCY(val) (((val)&LIRC_VALUE_MASK) | LIRC_MODE2_
↪FREQUENCY)
#define LIRC_TIMEOUT(val) (((val)&LIRC_VALUE_MASK) | LIRC_MODE2_
↪TIMEOUT)

#define LIRC_VALUE(val) ((val)&LIRC_VALUE_MASK)
#define LIRC_MODE2(val) ((val)&LIRC_MODE2_MASK)

#define LIRC_IS_SPACE(val) (LIRC_MODE2(val) == LIRC_MODE2_SPACE)
#define LIRC_IS_PULSE(val) (LIRC_MODE2(val) == LIRC_MODE2_PULSE)
#define LIRC_IS_FREQUENCY(val) (LIRC_MODE2(val) == LIRC_MODE2_
↪FREQUENCY)
#define LIRC_IS_TIMEOUT(val) (LIRC_MODE2(val) == LIRC_MODE2_TIMEOUT)

/* used heavily by lirc userspace */
#define lirc_t int
```

```
/*** lirc compatible hardware features ***/

#define LIRC_MODE2SEND(x) (x)
#define LIRC_SEND2MODE(x) (x)
#define LIRC_MODE2REC(x) ((x) << 16)
#define LIRC_REC2MODE(x) ((x) >> 16)

#define LIRC_MODE_RAW                  0x00000001
#define LIRC_MODE_PULSE                0x00000002
#define LIRC_MODE_MODE2                0x00000004
#define LIRC_MODE_SCANCODE             0x00000008
#define LIRC_MODE_LIRCCODE             0x00000010

#define LIRC_CAN_SEND_RAW              LIRC_MODE2SEND(LIRC_MODE_RAW)
#define LIRC_CAN_SEND_PULSE            LIRC_MODE2SEND(LIRC_MODE_
↪PULSE)
#define LIRC_CAN_SEND_MODE2            LIRC_MODE2SEND(LIRC_MODE_
↪MODE2)
#define LIRC_CAN_SEND_LIRCCODE         LIRC_MODE2SEND(LIRC_MODE_
↪LIRCCODE)

#define LIRC_CAN_SEND_MASK             0x0000003f

#define LIRC_CAN_SET_SEND_CARRIER      0x00000100
#define LIRC_CAN_SET_SEND_DUTY_CYCLE   0x00000200
#define LIRC_CAN_SET_TRANSMITTER_MASK  0x00000400

#define LIRC_CAN_REC_RAW               LIRC_MODE2REC(LIRC_MODE_RAW)
#define LIRC_CAN_REC_PULSE             LIRC_MODE2REC(LIRC_MODE_
↪PULSE)
#define LIRC_CAN_REC_MODE2             LIRC_MODE2REC(LIRC_MODE_
↪MODE2)
#define LIRC_CAN_REC_SCANCODE          LIRC_MODE2REC(LIRC_MODE_
↪SCANCODE)
#define LIRC_CAN_REC_LIRCCODE          LIRC_MODE2REC(LIRC_MODE_
↪LIRCCODE)

#define LIRC_CAN_REC_MASK              LIRC_MODE2REC(LIRC_CAN_SEND_
↪MASK)

#define LIRC_CAN_SET_REC_CARRIER       (LIRC_CAN_SET_SEND_CARRIER <
↪< 16)
#define LIRC_CAN_SET_REC_DUTY_CYCLE    (LIRC_CAN_SET_SEND_DUTY_
↪CYCLE << 16)

#define LIRC_CAN_SET_REC_DUTY_CYCLE_RANGE 0x40000000
#define LIRC_CAN_SET_REC_CARRIER_RANGE    0x80000000
#define LIRC_CAN_GET_REC_RESOLUTION       0x20000000
#define LIRC_CAN_SET_REC_TIMEOUT          0x10000000
#define LIRC_CAN_SET_REC_FILTER           0x08000000
```

```
#define LIRC_CAN_MEASURE_CARRIER          0x02000000
#define LIRC_CAN_USE_WIDEBAND_RECEIVER    0x04000000

#define LIRC_CAN_SEND(x) ((x)&LIRC_CAN_SEND_MASK)
#define LIRC_CAN_REC(x) ((x)&LIRC_CAN_REC_MASK)

#define LIRC_CAN_NOTIFY_DECODE            0x01000000

/*** IOCTL commands for lirc driver ***/

#define LIRC_GET_FEATURES               _IOR('i', 0x00000000, __u32)

#define LIRC_GET_SEND_MODE              _IOR('i', 0x00000001, __u32)
#define LIRC_GET_REC_MODE               _IOR('i', 0x00000002, __u32)
#define LIRC_GET_REC_RESOLUTION         _IOR('i', 0x00000007, __u32)

#define LIRC_GET_MIN_TIMEOUT            _IOR('i', 0x00000008, __u32)
#define LIRC_GET_MAX_TIMEOUT            _IOR('i', 0x00000009, __u32)

/* code length in bits, currently only for LIRC_MODE_LIRCCODE */
#define LIRC_GET_LENGTH                 _IOR('i', 0x0000000f, __u32)

#define LIRC_SET_SEND_MODE              _IOW('i', 0x00000011, __u32)
#define LIRC_SET_REC_MODE               _IOW('i', 0x00000012, __u32)
/* Note: these can reset the according pulse_width */
#define LIRC_SET_SEND_CARRIER           _IOW('i', 0x00000013, __u32)
#define LIRC_SET_REC_CARRIER            _IOW('i', 0x00000014, __u32)
#define LIRC_SET_SEND_DUTY_CYCLE        _IOW('i', 0x00000015, __u32)
#define LIRC_SET_TRANSMITTER_MASK       _IOW('i', 0x00000017, __u32)

/*
 * when a timeout != 0 is set the driver will send a
 * LIRC_MODE2_TIMEOUT data packet, otherwise LIRC_MODE2_TIMEOUT is
 * never sent, timeout is disabled by default
 */
#define LIRC_SET_REC_TIMEOUT            _IOW('i', 0x00000018, __u32)

/* 1 enables, 0 disables timeout reports in MODE2 */
#define LIRC_SET_REC_TIMEOUT_REPORTS    _IOW('i', 0x00000019, __u32)

/*
 * if enabled from the next key press on the driver will send
 * LIRC_MODE2_FREQUENCY packets
 */
#define LIRC_SET_MEASURE_CARRIER_MODE    _IOW('i', 0x0000001d, __u32)

/*
 * to set a range use LIRC_SET_REC_CARRIER_RANGE with the
 * lower bound first and later LIRC_SET_REC_CARRIER with the upper␣
↪bound
```

```
 */
#define LIRC_SET_REC_CARRIER_RANGE     _IOW('i', 0x0000001f, __u32)

#define LIRC_SET_WIDEBAND_RECEIVER     _IOW('i', 0x00000023, __u32)

/*
 * Return the recording timeout, which is either set by
 * the ioctl LIRC_SET_REC_TIMEOUT or by the kernel after setting␣
↪the protocols.
 */
#define LIRC_GET_REC_TIMEOUT           _IOR('i', 0x00000024, __u32)

/*
 * struct lirc_scancode - decoded scancode with protocol for use␣
↪with
 *      LIRC_MODE_SCANCODE
 *
 * @timestamp: Timestamp in nanoseconds using CLOCK_MONOTONIC when␣
↪IR
 *      was decoded.
 * @flags: should be 0 for transmit. When receiving scancodes,
 *      LIRC_SCANCODE_FLAG_TOGGLE or LIRC_SCANCODE_FLAG_REPEAT can␣
↪be set
 *      depending on the protocol
 * @rc_proto: see enum rc_proto
 * @keycode: the translated keycode. Set to 0 for transmit.
 * @scancode: the scancode received or to be sent
 */
struct lirc_scancode {
        __u64   timestamp;
        __u16   flags;
        __u16   rc_proto;
        __u32   keycode;
        __u64   scancode;
};

/* Set if the toggle bit of rc-5 or rc-6 is enabled */
#define LIRC_SCANCODE_FLAG_TOGGLE      1
/* Set if this is a nec or sanyo repeat */
#define LIRC_SCANCODE_FLAG_REPEAT      2

/**
 * enum rc_proto - the Remote Controller protocol
 *
 * @RC_PROTO_UNKNOWN: Protocol not known
 * @RC_PROTO_OTHER: Protocol known but proprietary
 * @RC_PROTO_RC5: Philips RC5 protocol
 * @RC_PROTO_RC5X_20: Philips RC5x 20 bit protocol
 * @RC_PROTO_RC5_SZ: StreamZap variant of RC5
 * @RC_PROTO_JVC: JVC protocol
 * @RC_PROTO_SONY12: Sony 12 bit protocol
```

```
 * @RC_PROTO_SONY15: Sony 15 bit protocol
 * @RC_PROTO_SONY20: Sony 20 bit protocol
 * @RC_PROTO_NEC: NEC protocol
 * @RC_PROTO_NECX: Extended NEC protocol
 * @RC_PROTO_NEC32: NEC 32 bit protocol
 * @RC_PROTO_SANYO: Sanyo protocol
 * @RC_PROTO_MCIR2_KBD: RC6-ish MCE keyboard
 * @RC_PROTO_MCIR2_MSE: RC6-ish MCE mouse
 * @RC_PROTO_RC6_0: Philips RC6-0-16 protocol
 * @RC_PROTO_RC6_6A_20: Philips RC6-6A-20 protocol
 * @RC_PROTO_RC6_6A_24: Philips RC6-6A-24 protocol
 * @RC_PROTO_RC6_6A_32: Philips RC6-6A-32 protocol
 * @RC_PROTO_RC6_MCE: MCE (Philips RC6-6A-32 subtype) protocol
 * @RC_PROTO_SHARP: Sharp protocol
 * @RC_PROTO_XMP: XMP protocol
 * @RC_PROTO_CEC: CEC protocol
 * @RC_PROTO_IMON: iMon Pad protocol
 * @RC_PROTO_RCMM12: RC-MM protocol 12 bits
 * @RC_PROTO_RCMM24: RC-MM protocol 24 bits
 * @RC_PROTO_RCMM32: RC-MM protocol 32 bits
 * @RC_PROTO_XBOX_DVD: Xbox DVD Movie Playback Kit protocol
 */
enum rc_proto {
        RC_PROTO_UNKNOWN        = 0,
        RC_PROTO_OTHER          = 1,
        RC_PROTO_RC5            = 2,
        RC_PROTO_RC5X_20        = 3,
        RC_PROTO_RC5_SZ         = 4,
        RC_PROTO_JVC            = 5,
        RC_PROTO_SONY12         = 6,
        RC_PROTO_SONY15         = 7,
        RC_PROTO_SONY20         = 8,
        RC_PROTO_NEC            = 9,
        RC_PROTO_NECX           = 10,
        RC_PROTO_NEC32          = 11,
        RC_PROTO_SANYO          = 12,
        RC_PROTO_MCIR2_KBD      = 13,
        RC_PROTO_MCIR2_MSE      = 14,
        RC_PROTO_RC6_0          = 15,
        RC_PROTO_RC6_6A_20      = 16,
        RC_PROTO_RC6_6A_24      = 17,
        RC_PROTO_RC6_6A_32      = 18,
        RC_PROTO_RC6_MCE        = 19,
        RC_PROTO_SHARP          = 20,
        RC_PROTO_XMP            = 21,
        RC_PROTO_CEC            = 22,
        RC_PROTO_IMON           = 23,
        RC_PROTO_RCMM12         = 24,
        RC_PROTO_RCMM24         = 25,
        RC_PROTO_RCMM32         = 26,
        RC_PROTO_XBOX_DVD       = 27,
```

```
};

#endif
```

8.4.7 Revision and Copyright

Authors:

- Carvalho Chehab, Mauro <mchehab@kernel.org>
- Initial version.

8.4.8 Revision History

revision
3.15 / 2014-02-06 (*mcc*)

Added the interface description and the RC sysfs class description.

revision
1.0 / 2009-09-06 (*mcc*)

Initial revision

8.5 Part IV - Media Controller API

8.5.1 Introduction

Media devices increasingly handle multiple related functions. Many USB cameras include microphones, video capture hardware can also output video, or SoC camera interfaces also perform memory-to-memory operations similar to video codecs.

Independent functions, even when implemented in the same hardware, can be modelled as separate devices. A USB camera with a microphone will be presented to userspace applications as V4L2 and ALSA capture devices. The devices' relationships (when using a webcam, end-users shouldn't have to manually select the associated USB microphone), while not made available directly to applications by the drivers, can usually be retrieved from sysfs.

With more and more advanced SoC devices being introduced, the current approach will not scale. Device topologies are getting increasingly complex and can't always be represented by a tree structure. Hardware blocks are shared between different functions, creating dependencies between seemingly unrelated devices.

Kernel abstraction APIs such as V4L2 and ALSA provide means for applications to access hardware parameters. As newer hardware expose an increasingly high number of those parameters, drivers need to guess what applications really require based on limited information, thereby implementing policies that belong to userspace.

The media controller API aims at solving those problems.

8.5.2 Media device model

Discovering a device internal topology, and configuring it at runtime, is one of the goals of the media controller API. To achieve this, hardware devices and Linux Kernel interfaces are modelled as graph objects on an oriented graph. The object types that constitute the graph are:

- An **entity** is a basic media hardware or software 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.
- An **interface** is a graph representation of a Linux Kernel userspace API interface, like a device node or a sysfs file that controls one or more entities in the graph.
- A **pad** is a data 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 **data 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.
- An **interface link** is a point-to-point bidirectional control connection between a Linux Kernel interface and an entity.

8.5.3 Types and flags used to represent the media graph elements

Table 236: Media entity functions

MEDIA_ENT_F_UNKNOWN and MEDIA_ENT_F_V4L2_SUBDEV_UNKNOWN	Unknown entity. That generally indicates that driver didn't initialize properly the entity, which is Kernel bug
MEDIA_ENT_F_IO_V4L	Data streaming input and/or output entity.
MEDIA_ENT_F_IO_VBI	V4L VBI streaming input or output entity
MEDIA_ENT_F_IO_SWRADIO	V4L Software Digital Radio (SDR) streaming input o output entity
MEDIA_ENT_F_IO_DTV	DVB Digital TV streaming input or output entity
MEDIA_ENT_F_DTV_DEMOD	Digital TV demodulator entity.
MEDIA_ENT_F_TS_DEMUX	MPEG Transport stream demux entity. Could b implemented on hardware or in Kernelspace by th Linux DVB subsystem.
MEDIA_ENT_F_DTV_CA	Digital TV Conditional Access module (CAM) entity
MEDIA_ENT_F_DTV_NET_DECAP	Digital TV network ULE/MLE desencapsulation en tity. Could be implemented on hardware or in Ker nelspace
MEDIA_ENT_F_CONN_RF	Connector for a Radio Frequency (RF) signal.
MEDIA_ENT_F_CONN_SVIDEO	Connector for a S-Video signal.
MEDIA_ENT_F_CONN_COMPOSITE	Connector for a RGB composite signal.

continues on next pag

Table 236 – continued from previous page

MEDIA_ENT_F_CAM_SENSOR	Camera video sensor entity.
MEDIA_ENT_F_FLASH	Flash controller entity.
MEDIA_ENT_F_LENS	Lens controller entity.
MEDIA_ENT_F_ATV_DECODER	Analog video decoder, the basic function of the vide decoder is to accept analogue video from a wide vari ety of sources such as broadcast, DVD players, cam eras and video cassette recorders, in either NTSC PAL, SECAM or HD format, separating the strean into its component parts, luminance and chrom nance, and output it in some digital video standard with appropriate timing signals.
MEDIA_ENT_F_TUNER	Digital TV, analog TV, radio and/or software radi tuner, with consists on a PLL tuning stage that con verts radio frequency (RF) signal into an Intermed ate Frequency (IF). Modern tuners have internally IF PLL decoders for audio and video, but older model have those stages implemented on separate entities
MEDIA_ENT_F_IF_VID_DECODER	IF-PLL video decoder. It receives the IF from a PL and decodes the analog TV video signal. This is com monly found on some very old analog tuners, lik Philips MK3 designs. They all contain a tda9887 (o some software compatible similar chip, like tda9885 Those devices use a different I2C address than th tuner PLL.
MEDIA_ENT_F_IF_AUD_DECODER	IF-PLL sound decoder. It receives the IF from a PL and decodes the analog TV audio signal. This is com monly found on some very old analog hardware, lik Micronas msp3400, Philips tda9840, tda985x, etc Those devices use a different I2C address than th tuner PLL and should be controlled together with th IF-PLL video decoder.
MEDIA_ENT_F_AUDIO_CAPTURE	Audio Capture Function Entity.
MEDIA_ENT_F_AUDIO_PLAYBACK	Audio Playback Function Entity.
MEDIA_ENT_F_AUDIO_MIXER	Audio Mixer Function Entity.
MEDIA_ENT_F_PROC_VIDEO_COMPOSER	Video composer (blender). An entity capable of vide composing must have at least two sink pads and on source pad, and composes input video frames ont output video frames. Composition can be performe using alpha blending, color keying, raster operation (ROP), stitching or any other means.
MEDIA_ENT_F_PROC_VIDEO_PIXEL_FORMATT	Video pixel formatter. An entity capable of pixel for matting must have at least one sink pad and on source pad. Read pixel formatters read pixels from memory and perform a subset of unpacking, crop ping, color keying, alpha multiplication and pixel en coding conversion. Write pixel formatters perform a subset of dithering, pixel encoding conversion an packing and write pixels to memory.

continues on next pag

Table 236 – continued from previous page

MEDIA_ENT_F_PROC_VIDEO_PIXEL_ENC_CON	Video pixel encoding converter. An entity capable o pixel encoding conversion must have at least one sin pad and one source pad, and convert the encoding o pixels received on its sink pad(s) to a different en coding output on its source pad(s). Pixel encodin conversion includes but isn't limited to RGB to/from HSV, RGB to/from YUV and CFA (Bayer) to RGB con versions.
MEDIA_ENT_F_PROC_VIDEO_LUT	Video look-up table. An entity capable of video looku table processing must have one sink pad and on source pad. It uses the values of the pixels receive on its sink pad to look up entries in internal tables an output them on its source pad. The lookup process ing can be performed on all components separately o combine them for multi-dimensional table lookups.
MEDIA_ENT_F_PROC_VIDEO_SCALER	Video scaler. An entity capable of video scaling mus have at least one sink pad and one source pad, an scale the video frame(s) received on its sink pad(s) t a different resolution output on its source pad(s). Th range of supported scaling ratios is entity-specifi and can differ between the horizontal and vertica directions (in particular scaling can be supported i one direction only). Binning and sub-sampling (occa sionally also referred to as skipping) are considere as scaling.
MEDIA_ENT_F_PROC_VIDEO_STATISTICS	Video statistics computation (histogram, 3A, etc.). A entity capable of statistics computation must hav one sink pad and one source pad. It computes statis tics over the frames received on its sink pad and ou puts the statistics data on its source pad.
MEDIA_ENT_F_PROC_VIDEO_ENCODER	Video (MPEG, HEVC, VPx, etc.) encoder. An entit capable of compressing video frames. Must have on sink pad and at least one source pad.
MEDIA_ENT_F_PROC_VIDEO_DECODER	Video (MPEG, HEVC, VPx, etc.) decoder. An en tity capable of decompressing a compressed vide stream into uncompressed video frames. Must hav one sink pad and at least one source pad.
MEDIA_ENT_F_VID_MUX	Video multiplexer. An entity capable of multiplexin must have at least two sink pads and one source pad and must pass the video frame(s) received from th active sink pad to the source pad.
MEDIA_ENT_F_VID_IF_BRIDGE	Video interface bridge. A video interface bridge en tity must have at least one sink pad and at least on source pad. It receives video frames on its sink pa from an input video bus of one type (HDMI, eDP, MIP CSI-2, etc.), and outputs them on its source pad to a output video bus of another type (eDP, MIPI CSI-2 parallel, etc.).

continues on next pag

Table 236 – continued from previous page

MEDIA_ENT_F_DV_DECODER	Digital video decoder. The basic function of the vide decoder is to accept digital video from a wide variet of sources and output it in some digital video stan dard, with appropriate timing signals.
MEDIA_ENT_F_DV_ENCODER	Digital video encoder. The basic function of the vide encoder is to accept digital video from some digita video standard with appropriate timing signals (usu ally a parallel video bus with sync signals) and outpu this to a digital video output connector such as HDM or DisplayPort.

Table 237: Media entity flags

MEDIA_ENT_FL_DEFAULT	Default entity for its type. Used to discover the default audio, VBI and video devices, the default camera sensor, etc.
MEDIA_ENT_FL_CONNECTOR	The entity represents a connector.

Table 238: Media interface types

MEDIA_INTF_T_DVB_FE	Device node interface for the Digital TV frontend	typically, /dev/dvb/adapter?/frontend?
MEDIA_INTF_T_DVB_DEMUX	Device node interface for the Digital TV demux	typically, /dev/dvb/adapter?/demux?
MEDIA_INTF_T_DVB_DVR	Device node interface for the Digital TV DVR	typically, /dev/dvb/adapter?/dvr?
MEDIA_INTF_T_DVB_CA	Device node interface for the Digital TV Conditional Access	typically, /dev/dvb/adapter?/ca?
MEDIA_INTF_T_DVB_NET	Device node interface for the Digital TV network control	typically, /dev/dvb/adapter?/net?
MEDIA_INTF_T_V4L_VIDEO	Device node interface for video (V4L)	typically, /dev/video?
MEDIA_INTF_T_V4L_VBI	Device node interface for VBI (V4L)	typically, /dev/vbi?
MEDIA_INTF_T_V4L_RADIO	Device node interface for radio (V4L)	typically, /dev/radio?
MEDIA_INTF_T_V4L_SUBDEV	Device node interface for a V4L subdevice	typically, /dev/v4l-subdev?
MEDIA_INTF_T_V4L_SWRADIO	Device node interface for Software Defined Radio (V4L)	typically, /dev/swradio?
MEDIA_INTF_T_V4L_TOUCH	Device node interface for Touch device (V4L)	typically, /dev/v4l-touch?
MEDIA_INTF_T_ALSA_PCM_CAPTUR	Device node interface for ALSA PCM Capture	typically, /dev/snd/pcmC?D?c
MEDIA_INTF_T_ALSA_PCM_PLAYBA	Device node interface for ALSA PCM Playback	typically, /dev/snd/pcmC?D?p
MEDIA_INTF_T_ALSA_CONTROL	Device node interface for ALSA Control	typically, /dev/snd/controlC?
MEDIA_INTF_T_ALSA_COMPRESS	Device node interface for ALSA Compress	typically, /dev/snd/compr?
MEDIA_INTF_T_ALSA_RAWMIDI	Device node interface for ALSA Raw MIDI	typically, /dev/snd/midi?
MEDIA_INTF_T_ALSA_HWDEP	Device node interface for ALSA Hardware Dependent	typically, /dev/snd/hwC?D?
MEDIA_INTF_T_ALSA_SEQUENCER	Device node interface for ALSA Sequencer	typically, /dev/snd/seq
MEDIA_INTF_T_ALSA_TIMER	Device node interface for ALSA Timer	typically, /dev/snd/timer

Table 239: Media pad flags

MEDIA_PAD_FL_SINK	Input pad, relative to the entity. Input pads sink data and are targets of links.
MEDIA_PAD_FL_SOURCE	Output pad, relative to the entity. Output pads source data and are origins of links.
MEDIA_PAD_FL_MUST_CONNEC	If this flag is set and the pad is linked to any other pad, then at least one of those links must be enabled for the entity to be able to stream. There could be temporary reasons (e.g. device configuration dependent) for the pad to need enabled links even when this flag isn't set; the absence of the flag doesn't imply there is none.

One and only one of MEDIA_PAD_FL_SINK and MEDIA_PAD_FL_SOURCE must be set for every pad.

Table 240: Media link flags

MEDIA_LNK_FL_ENABLED	The link is enabled and can be used to transfer media data. When two or more links target a sink pad, only one of them can be enabled at a time.
MEDIA_LNK_FL_IMMUTABLE	The link enabled state can't be modified at runtime. An immutable link is always enabled.
MEDIA_LNK_FL_DYNAMIC	The link enabled state can be modified during streaming. This flag is set by drivers and is read-only for applications.
MEDIA_LNK_FL_LINK_TYPE	This is a bitmask that defines the type of the link. Currently, two types of links are supported: MEDIA_LNK_FL_DATA_LINK if the link is between two pads MEDIA_LNK_FL_INTERFACE_LINK if the link is between an interface and an entity

8.5.4 Request API

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 *ioctl MEDIA_IOC_REQUEST_ALLOC* for the media device node. This returns a file descriptor representing the request. Typically, several such requests will be allocated.

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 *ioctl 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.

Request Submission

Once the configuration and buffers of the request are specified, it can be queued by calling *ioctl 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.

> **Caution:** For *memory-to-memory devices* 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.

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 *VIDIOC_STREAMOFF* is called or the device is *closed*. Attempts to directly queue a buffer when earlier a buffer was queued via a request or vice versa will result in an EBUSY error.

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 `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.

A completed request contains the state of the device after the request was executed. User-space can query that state by calling *ioctl VIDIOC_G_EXT_CTRLS* with the request file descriptor. Calling *ioctl 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.

Recycling and Destruction

Finally, a completed request can either be discarded or be reused. Calling `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.

The *ioctl 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.

Example for a Codec Device

For use-cases such as *codecs*, 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.

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 *VIDIOC_DQBUF*:

```
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 *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.

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

(continues on next page)

(continued from previous page)

```
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 *ioctl MEDIA_REQUEST_IOC_REINIT*…

```
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.

8.5.5 Function Reference

media open()

Name

media-open - Open a media device

Synopsis

```
#include <fcntl.h>
```

int **open**(const char *device_name, int flags)

Arguments

device_name
: Device to be opened.

flags
: Open flags. Access mode must be either `O_RDONLY` or `O_RDWR`. Other flags have no effect.

Description

To open a media device applications call *open()* with the desired device name. The function has no side effects; the device configuration remain unchanged.

When the device is opened in read-only mode, attempts to modify its configuration will result in an error, and `errno` will be set to EBADF.

Return Value

open() returns the new file descriptor on success. On error, -1 is returned, and `errno` is set appropriately. Possible error codes are:

EACCES
: The requested access to the file is not allowed.

EMFILE
: The process already has the maximum number of files open.

ENFILE
: The system limit on the total number of open files has been reached.

ENOMEM
: Insufficient kernel memory was available.

ENXIO
: No device corresponding to this device special file exists.

media close()

Name

media-close - Close a media device

Synopsis

```
#include <unistd.h>
```

int **close**(int fd)

Arguments

fd
File descriptor returned by *open()*.

Description

Closes the media device. Resources associated with the file descriptor are freed. The device configuration remain unchanged.

Return Value

close() returns 0 on success. On error, -1 is returned, and `errno` is set appropriately. Possible error codes are:

EBADF
`fd` is not a valid open file descriptor.

media ioctl()

Name

media-ioctl - Control a media device

Synopsis

```
#include <sys/ioctl.h>
```

`int ioctl(int fd, int request, void *argp)`

Arguments

fd
: File descriptor returned by *open()*.

request
: Media ioctl request code as defined in the media.h header file, for example MEDIA_IOC_SETUP_LINK.

argp
: Pointer to a request-specific structure.

Description

The *ioctl()* function manipulates media device parameters. The argument `fd` must be an open file descriptor.

The ioctl `request` code specifies the media function to be called. It has encoded in it whether the argument is an input, output or read/write parameter, and the size of the argument `argp` in bytes.

Macros and structures definitions specifying media ioctl requests and their parameters are located in the media.h header file. All media ioctl requests, their respective function and parameters are specified in *Function Reference*.

Return Value

On success 0 is returned, on error -1 and the `errno` variable is set appropriately. The generic error codes are described at the *Generic Error Codes* chapter.

Request-specific error codes are listed in the individual requests descriptions.

When an ioctl that takes an output or read/write parameter fails, the parameter remains unmodified.

ioctl MEDIA_IOC_DEVICE_INFO

Name

MEDIA_IOC_DEVICE_INFO - Query device information

Synopsis

MEDIA_IOC_DEVICE_INFO

```
int ioctl(int fd, MEDIA_IOC_DEVICE_INFO, struct media_device_info
*argp)
```

Arguments

fd
File descriptor returned by *open()*.

argp
Pointer to struct *media_device_info*.

Description

All media devices must support the `MEDIA_IOC_DEVICE_INFO` ioctl. To query device information, applications call the ioctl with a pointer to a struct *media_device_info*. The driver fills the structure and returns the information to the application. The ioctl never fails.

type **media_device_info**

Table 241: struct media_device_info

char	driver[16]	Name of the driver implementing the media API as a NUL-terminated ASCII string. The driver version is stored in the `driver_version` field. Driver specific applications can use this information to verify the driver identity. It is also useful to work around known bugs, or to identify drivers in error reports.
char	model[32]	Device model name as a NUL-terminated UTF-8 string. The device version is stored in the `device_version` field and is not be appended to the model name.
char	serial[40]	Serial number as a NUL-terminated ASCII string.
char	bus_info[32]	Location of the device in the system as a NUL-terminated ASCII string. This includes the bus type name (PCI, USB, ⋯) and a bus-specific identifier.
__u32	media_version	Media API version, formatted with the `KERNEL_VERSION()` macro.
__u32	hw_revision	Hardware device revision in a driver-specific format.
__u32	driver_version	Media device driver version, formatted with the `KERNEL_VERSION()` macro. Together with the `driver` field this identifies a particular driver.
__u32	reserved[31]	Reserved for future extensions. Drivers and applications must set this array to zero.

The `serial` and `bus_info` fields can be used to distinguish between multiple instances of otherwise identical hardware. The serial number takes precedence when provided and can be assumed to be unique. If the serial number is an empty string, the `bus_info` field can be used instead. The `bus_info` field is guaranteed to be unique, but can vary across reboots or device unplug/replug.

Return Value

On success 0 is returned, on error -1 and the `errno` variable is set appropriately. The generic error codes are described at the *Generic Error Codes* chapter.

ioctl MEDIA_IOC_G_TOPOLOGY

Name

MEDIA_IOC_G_TOPOLOGY - Enumerate the graph topology and graph element properties

Synopsis

MEDIA_IOC_G_TOPOLOGY

`int ioctl(int fd, MEDIA_IOC_G_TOPOLOGY, struct media_v2_topology *argp)`

Arguments

fd
File descriptor returned by *open()*.

argp
Pointer to struct *media_v2_topology*.

Description

The typical usage of this ioctl is to call it twice. On the first call, the structure defined at struct *media_v2_topology* should be zeroed. At return, if no errors happen, this ioctl will return the `topology_version` and the total number of entities, interfaces, pads and links.

Before the second call, the userspace should allocate arrays to store the graph elements that are desired, putting the pointers to them at the ptr_entities, ptr_interfaces, ptr_links and/or ptr_pads, keeping the other values untouched.

If the `topology_version` remains the same, the ioctl should fill the desired arrays with the media graph elements.

type **media_v2_topology**

Table 242: struct media_v2_topology

__u64	topology_versio	Version of the media graph topology. When the graph is created, this field starts with zero. Every time a graph element is added or removed, this field is incremented.
__u32	num_entities	Number of entities in the graph
__u32	reserved1	Applications and drivers shall set this to 0.
__u64	ptr_entities	A pointer to a memory area where the entities array will be stored, converted to a 64-bits integer. It can be zero. if zero, the ioctl won't store the entities. It will just update `num_entities`
__u32	num_interfaces	Number of interfaces in the graph
__u32	reserved2	Applications and drivers shall set this to 0.
__u64	ptr_interfaces	A pointer to a memory area where the interfaces array will be stored, converted to a 64-bits integer. It can be zero. if zero, the ioctl won't store the interfaces. It will just update `num_interfaces`
__u32	num_pads	Total number of pads in the graph
__u32	reserved3	Applications and drivers shall set this to 0.
__u64	ptr_pads	A pointer to a memory area where the pads array will be stored, converted to a 64-bits integer. It can be zero. if zero, the ioctl won't store the pads. It will just update `num_pads`
__u32	num_links	Total number of data and interface links in the graph
__u32	reserved4	Applications and drivers shall set this to 0.
__u64	ptr_links	A pointer to a memory area where the links array will be stored, converted to a 64-bits integer. It can be zero. if zero, the ioctl won't store the links. It will just update `num_links`

type **media_v2_entity**

Table 243: struct media_v2_entity

__u32	id	Unique ID for the entity. Do not expect that the ID will always be the same for each instance of the device. In other words, do not hardcode entity IDs in an application.
char	name[64]	Entity name as an UTF-8 NULL-terminated string. This name must be unique within the media topology.
__u32	function	Entity main function, see *Media entity functions* for details.
__u32	flags	Entity flags, see *Media entity flags* for details. Only valid if `MEDIA_V2_ENTITY_HAS_FLAGS(media_version)` returns true. The `media_version` is defined in struct *media_device_info* and can be retrieved using *ioctl MEDIA_IOC_DEVICE_INFO*.
__u32	reserved[5]	Reserved for future extensions. Drivers and applications must set this array to zero.

type **media_v2_interface**

Table 244: struct media_v2_interface

__u32	id	Unique ID for the interface. Do not expect that the ID will always be the same for each instance of the device. In other words, do not hardcode interface IDs in an application.
__u32	intf_type	Interface type, see *Media interface types* for details.
__u32	flags	Interface flags. Currently unused.
__u32	reserved[9]	Reserved for future extensions. Drivers and applications must set this array to zero.
struct me-dia_v2_ir	devnode	Used only for device node interfaces. See *media_v2_intf_devnode* for details.

type **media_v2_intf_devnode**

Table 245: struct media_v2_intf_devnode

__u32	major	Device node major number.
__u32	minor	Device node minor number.

type **media_v2_pad**

Table 246: struct media_v2_pad

__u32	id	Unique ID for the pad. Do not expect that the ID will always be the same for each instance of the device. In other words, do not hardcode pad IDs in an application.
__u32	entity_id	Unique ID for the entity where this pad belongs.
__u32	flags	Pad flags, see *Media pad flags* for more details.
__u32	index	Pad index, starts at 0. Only valid if `MEDIA_V2_PAD_HAS_INDEX(media_version)` returns true. The `media_version` is defined in struct *media_device_info* and can be retrieved using *ioctl MEDIA_IOC_DEVICE_INFO*.
__u32	reserved[4]	Reserved for future extensions. Drivers and applications must set this array to zero.

type **media_v2_link**

Table 247: struct media_v2_link

__u32	id	Unique ID for the link. Do not expect that the ID will always be the same for each instance of the device. In other words, do not hardcode link IDs in an application.
__u32	source_id	On pad to pad links: unique ID for the source pad. On interface to entity links: unique ID for the interface.
__u32	sink_id	On pad to pad links: unique ID for the sink pad. On interface to entity links: unique ID for the entity.
__u32	flags	Link flags, see *Media link flags* for more details.
__u32	reserved[6]	Reserved for future extensions. Drivers and applications must set this array to zero.

Return Value

On success 0 is returned, on error -1 and the `errno` variable is set appropriately. The generic error codes are described at the *Generic Error Codes* chapter.

ENOSPC
This is returned when either one or more of the num_entities, num_interfaces, num_links or num_pads are non-zero and are smaller than the actual number of elements inside the graph. This may happen if the `topology_version` changed when compared to the last time this ioctl was called. Userspace should usually free the area for the pointers, zero the struct elements and call this ioctl again.

ioctl MEDIA_IOC_ENUM_ENTITIES

Name

MEDIA_IOC_ENUM_ENTITIES - Enumerate entities and their properties

Synopsis

`MEDIA_IOC_ENUM_ENTITIES`

`int ioctl(int fd, MEDIA_IOC_ENUM_ENTITIES, struct media_entity_desc *argp)`

Arguments

`fd`
File descriptor returned by *`open()`*.

`argp`
Pointer to struct *`media_entity_desc`*.

Description

To query the attributes of an entity, applications set the id field of a struct *`media_entity_desc`* structure and call the MEDIA_IOC_ENUM_ENTITIES ioctl with a pointer to this structure. The driver fills the rest of the structure or returns an EINVAL error code when the id is invalid.

Entities can be enumerated by or'ing the id with the `MEDIA_ENT_ID_FLAG_NEXT` flag. The driver will return information about the entity with the smallest id strictly larger than the requested one ('next entity'), or the `EINVAL` error code if there is none.

Entity IDs can be non-contiguous. Applications must *not* try to enumerate entities by calling MEDIA_IOC_ENUM_ENTITIES with increasing id's until they get an error.

type **media_entity_desc**

Table 248: struct media_entity_desc

__u32	id		Entity ID, set by the application. When the ID is or'ed with MEDIA_ENT_ID_FLAG_NEXT, the driver clears the flag and returns the first entity with a larger ID. Do not expect that the ID will always be the same for each instance of the device. In other words, do not hard-code entity IDs in an application.
char	name[32]		Entity name as an UTF-8 NULL-terminated string. This

Return Value

On success 0 is returned, on error -1 and the `errno` variable is set appropriately. The generic error codes are described at the *Generic Error Codes* chapter.

EINVAL
The struct *media_entity_desc* `id` references a non-existing entity.

ioctl MEDIA_IOC_ENUM_LINKS

Name

MEDIA_IOC_ENUM_LINKS - Enumerate all pads and links for a given entity

Synopsis

MEDIA_IOC_ENUM_LINKS

`int ioctl(int fd, MEDIA_IOC_ENUM_LINKS, struct media_links_enum *argp)`

Arguments

fd
File descriptor returned by *open()*.

argp
Pointer to struct *media_links_enum*.

Description

To enumerate pads and/or links for a given entity, applications set the entity field of a struct *media_links_enum* structure and initialize the struct *media_pad_desc* and struct *media_link_desc* structure arrays pointed by the `pads` and `links` fields. They then call the MEDIA_IOC_ENUM_LINKS ioctl with a pointer to this structure.

If the `pads` field is not NULL, the driver fills the `pads` array with information about the entity's pads. The array must have enough room to store all the entity's pads. The number of pads can be retrieved with *ioctl MEDIA_IOC_ENUM_ENTITIES*.

If the `links` field is not NULL, the driver fills the `links` array with information about the entity's outbound links. The array must have enough room to store all the entity's outbound links. The number of outbound links can be retrieved with *ioctl MEDIA_IOC_ENUM_ENTITIES*.

Only forward links that originate at one of the entity's source pads are returned during the enumeration process.

type **media_links_enum**

Table 249: struct media_links_enum

__u32	entity	Entity id, set by the application.
struct *media_pad_desc*	*pads	Pointer to a pads array allocated by the application. Ignored if NULL.
struct *media_link_desc*	*links	Pointer to a links array allocated by the application. Ignored if NULL.
__u32	reserved[4]	Reserved for future extensions. Drivers and applications must set the array to zero.

type **media_pad_desc**

Table 250: struct media_pad_desc

__u32	entity	ID of the entity this pad belongs to.
__u16	index	Pad index, starts at 0.
__u32	flags	Pad flags, see *Media pad flags* for more details.
__u32	reserved[2]	Reserved for future extensions. Drivers and applications must set the array to zero.

type **media_link_desc**

Table 251: struct media_link_desc

struct *media_pad_desc*	source	Pad at the origin of this link.
struct *media_pad_desc*	sink	Pad at the target of this link.
__u32	flags	Link flags, see *Media link flags* for more details.
__u32	reserved[2]	Reserved for future extensions. Drivers and applications must set the array to zero.

Return Value

On success 0 is returned, on error -1 and the `errno` variable is set appropriately. The generic error codes are described at the *Generic Error Codes* chapter.

EINVAL

The struct *media_links_enum* `id` references a non-existing entity.

ioctl MEDIA_IOC_SETUP_LINK

Name

MEDIA_IOC_SETUP_LINK - Modify the properties of a link

Synopsis

MEDIA_IOC_SETUP_LINK

```
int ioctl(int fd, MEDIA_IOC_SETUP_LINK, struct media_link_desc
*argp)
```

Arguments

fd
File descriptor returned by *open()*.

argp
Pointer to struct *media_link_desc*.

Description

To change link properties applications fill a struct *media_link_desc* with link identification information (source and sink pad) and the new requested link flags. They then call the MEDIA_IOC_SETUP_LINK ioctl with a pointer to that structure.

The only configurable property is the `ENABLED` link flag to enable/disable a link. Links marked with the `IMMUTABLE` link flag can not be enabled or disabled.

Link configuration has no side effect on other links. If an enabled link at the sink pad prevents the link from being enabled, the driver returns with an `EBUSY` error code.

Only links marked with the `DYNAMIC` link flag can be enabled/disabled while streaming media data. Attempting to enable or disable a streaming non-dynamic link will return an `EBUSY` error code.

If the specified link can't be found the driver returns with an `EINVAL` error code.

Return Value

On success 0 is returned, on error -1 and the `errno` variable is set appropriately. The generic error codes are described at the *Generic Error Codes* chapter.

EINVAL
The struct *media_link_desc* references a non-existing link, or the link is immutable and an attempt to modify its configuration was made.

ioctl MEDIA_IOC_REQUEST_ALLOC

Name

MEDIA_IOC_REQUEST_ALLOC - Allocate a request

Synopsis

MEDIA_IOC_REQUEST_ALLOC

```
int ioctl(int fd, MEDIA_IOC_REQUEST_ALLOC, int *argp)
```

Arguments

fd
File descriptor returned by *open()*.

argp
Pointer to an integer.

Description

If the media device supports *requests*, then this ioctl can be used to allocate a request. If it is not supported, then `errno` is set to `ENOTTY`. A request is accessed through a file descriptor that is returned in `*argp`.

If the request was successfully allocated, then the request file descriptor can be passed to the *VIDIOC_QBUF*, *VIDIOC_G_EXT_CTRLS*, *VIDIOC_S_EXT_CTRLS* and *VIDIOC_TRY_EXT_CTRLS* ioctls.

In addition, the request can be queued by calling *ioctl MEDIA_REQUEST_IOC_QUEUE* and re-initialized by calling *ioctl MEDIA_REQUEST_IOC_REINIT*.

Finally, the file descriptor can be *polled* to wait for the request to complete.

The request will remain allocated until all the file descriptors associated with it are closed by *close()* and the driver no longer uses the request internally. See also *here* for more information.

Return Value

On success 0 is returned, on error -1 and the `errno` variable is set appropriately. The generic error codes are described at the *Generic Error Codes* chapter.

ENOTTY
The driver has no support for requests.

request close()

Name

request-close - Close a request file descriptor

Synopsis

```
#include <unistd.h>
```

int **close**(int fd)

Arguments

fd
File descriptor returned by *ioctl MEDIA_IOC_REQUEST_ALLOC*.

Description

Closes the request file descriptor. Resources associated with the request are freed once all file descriptors associated with the request are closed and the driver has completed the request. See *here* for more information.

Return Value

close() returns 0 on success. On error, -1 is returned, and `errno` is set appropriately. Possible error codes are:

EBADF
`fd` is not a valid open file descriptor.

request ioctl()

Name

request-ioctl - Control a request file descriptor

Synopsis

```
#include <sys/ioctl.h>
```

`int ioctl(int fd, int cmd, void *argp)`

Arguments

fd
: File descriptor returned by *ioctl MEDIA_IOC_REQUEST_ALLOC*.

cmd
: The request ioctl command code as defined in the media.h header file, for example *ioctl MEDIA_REQUEST_IOC_QUEUE*.

argp
: Pointer to a request-specific structure.

Description

The *ioctl()* function manipulates request parameters. The argument `fd` must be an open file descriptor.

The ioctl `cmd` code specifies the request function to be called. It has encoded in it whether the argument is an input, output or read/write parameter, and the size of the argument `argp` in bytes.

Macros and structures definitions specifying request ioctl commands and their parameters are located in the media.h header file. All request ioctl commands, their respective function and parameters are specified in *Function Reference*.

Return Value

On success 0 is returned, on error -1 and the `errno` variable is set appropriately. The generic error codes are described at the *Generic Error Codes* chapter.

Command-specific error codes are listed in the individual command descriptions.

When an ioctl that takes an output or read/write parameter fails, the parameter remains unmodified.

request poll()

Name

request-poll - Wait for some event on a file descriptor

Synopsis

```
#include <sys/poll.h>
```

int **poll**(struct pollfd *ufds, unsigned int nfds, int timeout)

Arguments

ufds
List of file descriptor events to be watched

nfds
Number of file descriptor events at the *ufds array

timeout
Timeout to wait for events

Description

With the *poll()* function applications can wait for a request to complete.

On success *poll()* returns the number of file descriptors that have been selected (that is, file descriptors for which the `revents` field of the respective struct `pollfd` is non-zero). Request file descriptor set the `POLLPRI` flag in `revents` when the request was completed. When the function times out it returns a value of zero, on failure it returns -1 and the `errno` variable is set appropriately.

Attempting to poll for a request that is not yet queued will set the `POLLERR` flag in `revents`.

Return Value

On success, *poll()* returns the number of structures which have non-zero `revents` fields, or zero if the call timed out. On error -1 is returned, and the `errno` variable is set appropriately:

EBADF
One or more of the `ufds` members specify an invalid file descriptor.

EFAULT
`ufds` references an inaccessible memory area.

EINTR
The call was interrupted by a signal.

EINVAL
The `nfds` value exceeds the `RLIMIT_NOFILE` value. Use `getrlimit()` to obtain this value.

ioctl MEDIA_REQUEST_IOC_QUEUE

Name

MEDIA_REQUEST_IOC_QUEUE - Queue a request

Synopsis

MEDIA_REQUEST_IOC_QUEUE

`int ioctl(int request_fd, MEDIA_REQUEST_IOC_QUEUE)`

Arguments

request_fd
File descriptor returned by *ioctl MEDIA_IOC_REQUEST_ALLOC*.

Description

If the media device supports *requests*, then this request ioctl can be used to queue a previously allocated request.

If the request was successfully queued, then the file descriptor can be *polled* to wait for the request to complete.

If the request was already queued before, then `EBUSY` is returned. Other errors can be returned if the contents of the request contained invalid or inconsistent data, see the next section for a list of common error codes. On error both the request and driver state are unchanged.

Once a request is queued, then the driver is required to gracefully handle errors that occur when the request is applied to the hardware. The exception is the `EIO` error which signals a fatal error that requires the application to stop streaming to reset the hardware state.

It is not allowed to mix queuing requests with queuing buffers directly (without a request). `EBUSY` will be returned if the first buffer was queued directly and you next try to queue a request, or vice versa.

A request must contain at least one buffer, otherwise this ioctl will return an `ENOENT` error.

Return Value

On success 0 is returned, on error -1 and the `errno` variable is set appropriately. The generic error codes are described at the *Generic Error Codes* chapter.

EBUSY
The request was already queued or the application queued the first buffer directly, but later attempted to use a request. It is not permitted to mix the two APIs.

ENOENT
The request did not contain any buffers. All requests are required to have at least one buffer. This can also be returned if some required configuration is missing in the request.

ENOMEM
Out of memory when allocating internal data structures for this request.

EINVAL
The request has invalid data.

EIO
The hardware is in a bad state. To recover, the application needs to stop streaming to reset the hardware state and then try to restart streaming.

ioctl MEDIA_REQUEST_IOC_REINIT

Name

MEDIA_REQUEST_IOC_REINIT - Re-initialize a request

Synopsis

`MEDIA_REQUEST_IOC_REINIT`

```
int ioctl(int request_fd, MEDIA_REQUEST_IOC_REINIT)
```

Arguments

`request_fd`
File descriptor returned by *ioctl MEDIA_IOC_REQUEST_ALLOC*.

Description

If the media device supports *requests*, then this request ioctl can be used to re-initialize a previously allocated request.

Re-initializing a request will clear any existing data from the request. This avoids having to `close()` a completed request and allocate a new request. Instead the completed request can just be re-initialized and it is ready to be used again.

A request can only be re-initialized if it either has not been queued yet, or if it was queued and completed. Otherwise it will set `errno` to EBUSY. No other error codes can be returned.

Return Value

On success 0 is returned, on error -1 and the `errno` variable is set appropriately.

EBUSY
The request is queued but not yet completed.

8.5.6 Media Controller Header File

media.h

```
/* SPDX-License-Identifier: GPL-2.0 WITH Linux-syscall-note */
/*
 * Multimedia device API
 *
 * Copyright (C) 2010 Nokia Corporation
 *
 * Contacts: Laurent Pinchart <laurent.pinchart@ideasonboard.com>
 *           Sakari Ailus <sakari.ailus@iki.fi>
 *
 * This program is free software; you can redistribute it and/or␣
↪modify
 * it under the terms of the GNU General Public License version 2 as
 * published by the Free Software Foundation.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 */

#ifndef __LINUX_MEDIA_H
#define __LINUX_MEDIA_H

#ifndef __KERNEL__
#include <stdint.h>
#endif
#include <linux/ioctl.h>
```

```
#include <linux/types.h>

struct media_device_info {
        char driver[16];
        char model[32];
        char serial[40];
        char bus_info[32];
        __u32 media_version;
        __u32 hw_revision;
        __u32 driver_version;
        __u32 reserved[31];
};

/*
 * Base number ranges for entity functions
 *
 * NOTE: Userspace should not rely on these ranges to identify a␣
↪group
 * of function types, as newer functions can be added with any name␣
↪within
 * the full u32 range.
 *
 * Some older functions use the MEDIA_ENT_F_OLD_*_BASE range. Do not
 * change this, this is for backwards compatibility. When adding new
 * functions always use MEDIA_ENT_F_BASE.
 */
#define MEDIA_ENT_F_BASE                        0x00000000
#define MEDIA_ENT_F_OLD_BASE                    0x00010000
#define MEDIA_ENT_F_OLD_SUBDEV_BASE             0x00020000

/*
 * Initial value to be used when a new entity is created
 * Drivers should change it to something useful.
 */
#define MEDIA_ENT_F_UNKNOWN                     MEDIA_ENT_F_BASE

/*
 * Subdevs are initialized with MEDIA_ENT_F_V4L2_SUBDEV_UNKNOWN in␣
↪order
 * to preserve backward compatibility. Drivers must change to the␣
↪proper
 * subdev type before registering the entity.
 */
#define MEDIA_ENT_F_V4L2_SUBDEV_UNKNOWN         MEDIA_ENT_F_OLD_
↪SUBDEV_BASE

/*
 * DVB entity functions
 */
#define MEDIA_ENT_F_DTV_DEMOD                   (MEDIA_ENT_F_BASE +␣
↪0x00001)
```

```
#define MEDIA_ENT_F_TS_DEMUX                        (MEDIA_ENT_F_BASE +␣
↪0x00002)
#define MEDIA_ENT_F_DTV_CA                          (MEDIA_ENT_F_BASE +␣
↪0x00003)
#define MEDIA_ENT_F_DTV_NET_DECAP                   (MEDIA_ENT_F_BASE +␣
↪0x00004)

/*
 * I/O entity functions
 */
#define MEDIA_ENT_F_IO_V4L                          (MEDIA_ENT_F_OLD_
↪BASE + 1)
#define MEDIA_ENT_F_IO_DTV                          (MEDIA_ENT_F_BASE +␣
↪0x01001)
#define MEDIA_ENT_F_IO_VBI                          (MEDIA_ENT_F_BASE +␣
↪0x01002)
#define MEDIA_ENT_F_IO_SWRADIO                      (MEDIA_ENT_F_BASE +␣
↪0x01003)

/*
 * Sensor functions
 */
#define MEDIA_ENT_F_CAM_SENSOR                      (MEDIA_ENT_F_OLD_
↪SUBDEV_BASE + 1)
#define MEDIA_ENT_F_FLASH                           (MEDIA_ENT_F_OLD_
↪SUBDEV_BASE + 2)
#define MEDIA_ENT_F_LENS                            (MEDIA_ENT_F_OLD_
↪SUBDEV_BASE + 3)

/*
 * Digital TV, analog TV, radio and/or software defined radio tuner␣
↪functions.
 *
 * It is a responsibility of the master/bridge drivers to add␣
↪connectors
 * and links for MEDIA_ENT_F_TUNER. Please notice that some old␣
↪tuners
 * may require the usage of separate I2C chips to decode analog TV␣
↪signals,
 * when the master/bridge chipset doesn't have its own TV standard␣
↪decoder.
 * On such cases, the IF-PLL staging is mapped via one or two␣
↪entities:
 * MEDIA_ENT_F_IF_VID_DECODER and/or MEDIA_ENT_F_IF_AUD_DECODER.
 */
#define MEDIA_ENT_F_TUNER                           (MEDIA_ENT_F_OLD_
↪SUBDEV_BASE + 5)

/*
 * Analog TV IF-PLL decoder functions
 *
```

```
 * It is a responsibility of the master/bridge drivers to create␣
↪links
 * for MEDIA_ENT_F_IF_VID_DECODER and MEDIA_ENT_F_IF_AUD_DECODER.
 */
#define MEDIA_ENT_F_IF_VID_DECODER              (MEDIA_ENT_F_BASE +␣
↪0x02001)
#define MEDIA_ENT_F_IF_AUD_DECODER              (MEDIA_ENT_F_BASE +␣
↪0x02002)

/*
 * Audio entity functions
 */
#define MEDIA_ENT_F_AUDIO_CAPTURE               (MEDIA_ENT_F_BASE +␣
↪0x03001)
#define MEDIA_ENT_F_AUDIO_PLAYBACK              (MEDIA_ENT_F_BASE +␣
↪0x03002)
#define MEDIA_ENT_F_AUDIO_MIXER                 (MEDIA_ENT_F_BASE +␣
↪0x03003)

/*
 * Processing entity functions
 */
#define MEDIA_ENT_F_PROC_VIDEO_COMPOSER         (MEDIA_ENT_F_BASE +␣
↪0x4001)
#define MEDIA_ENT_F_PROC_VIDEO_PIXEL_FORMATTER  (MEDIA_ENT_F_BASE +␣
↪0x4002)
#define MEDIA_ENT_F_PROC_VIDEO_PIXEL_ENC_CONV   (MEDIA_ENT_F_BASE +␣
↪0x4003)
#define MEDIA_ENT_F_PROC_VIDEO_LUT              (MEDIA_ENT_F_BASE +␣
↪0x4004)
#define MEDIA_ENT_F_PROC_VIDEO_SCALER           (MEDIA_ENT_F_BASE +␣
↪0x4005)
#define MEDIA_ENT_F_PROC_VIDEO_STATISTICS       (MEDIA_ENT_F_BASE +␣
↪0x4006)
#define MEDIA_ENT_F_PROC_VIDEO_ENCODER          (MEDIA_ENT_F_BASE +␣
↪0x4007)
#define MEDIA_ENT_F_PROC_VIDEO_DECODER          (MEDIA_ENT_F_BASE +␣
↪0x4008)

/*
 * Switch and bridge entity functions
 */
#define MEDIA_ENT_F_VID_MUX                     (MEDIA_ENT_F_BASE +␣
↪0x5001)
#define MEDIA_ENT_F_VID_IF_BRIDGE               (MEDIA_ENT_F_BASE +␣
↪0x5002)

/*
 * Video decoder/encoder functions
 */
#define MEDIA_ENT_F_ATV_DECODER                 (MEDIA_ENT_F_OLD_
```

```
↪SUBDEV_BASE + 4)
#define MEDIA_ENT_F_DV_DECODER                  (MEDIA_ENT_F_BASE +␣
↪0x6001)
#define MEDIA_ENT_F_DV_ENCODER                  (MEDIA_ENT_F_BASE +␣
↪0x6002)

/* Entity flags */
#define MEDIA_ENT_FL_DEFAULT                    (1 << 0)
#define MEDIA_ENT_FL_CONNECTOR                  (1 << 1)

/* OR with the entity id value to find the next entity */
#define MEDIA_ENT_ID_FLAG_NEXT                  (1U << 31)

struct media_entity_desc {
        __u32 id;
        char name[32];
        __u32 type;
        __u32 revision;
        __u32 flags;
        __u32 group_id;
        __u16 pads;
        __u16 links;

        __u32 reserved[4];

        union {
                /* Node specifications */
                struct {
                        __u32 major;
                        __u32 minor;
                } dev;

#if !defined(__KERNEL__)
                /*
                 * TODO: this shouldn't have been added without
                 * actual drivers that use this. When the first␣
↪real driver
                 * appears that sets this information, special␣
↪attention
                 * should be given whether this information is 1)␣
↪enough, and
                 * 2) can deal with udev rules that rename devices.␣
↪The struct
                 * dev would not be sufficient for this since that␣
↪does not
                 * contain the subdevice information. In addition,␣
↪struct dev
                 * can only refer to a single device, and not to␣
↪multiple (e.g.
                 * pcm and mixer devices).
                 */
```

```
                struct {
                        __u32 card;
                        __u32 device;
                        __u32 subdevice;
                } alsa;

                /*
                 * DEPRECATED: previous node specifications. Kept␣
↪just to
                 * avoid breaking compilation. Use media_entity_
↪desc.dev
                 * instead.
                 */
                struct {
                        __u32 major;
                        __u32 minor;
                } v4l;
                struct {
                        __u32 major;
                        __u32 minor;
                } fb;
                int dvb;
#endif

                /* Sub-device specifications */
                /* Nothing needed yet */
                __u8 raw[184];
        };
};

#define MEDIA_PAD_FL_SINK                       (1 << 0)
#define MEDIA_PAD_FL_SOURCE                     (1 << 1)
#define MEDIA_PAD_FL_MUST_CONNECT               (1 << 2)

struct media_pad_desc {
        __u32 entity;           /* entity ID */
        __u16 index;            /* pad index */
        __u32 flags;            /* pad flags */
        __u32 reserved[2];
};

#define MEDIA_LNK_FL_ENABLED                    (1 << 0)
#define MEDIA_LNK_FL_IMMUTABLE                  (1 << 1)
#define MEDIA_LNK_FL_DYNAMIC                    (1 << 2)

#define MEDIA_LNK_FL_LINK_TYPE                  (0xf << 28)
#  define MEDIA_LNK_FL_DATA_LINK                (0 << 28)
#  define MEDIA_LNK_FL_INTERFACE_LINK           (1 << 28)

struct media_link_desc {
        struct media_pad_desc source;
```

```
        struct media_pad_desc sink;
        __u32 flags;
        __u32 reserved[2];
};

struct media_links_enum {
        __u32 entity;
        /* Should have enough room for pads elements */
        struct media_pad_desc __user *pads;
        /* Should have enough room for links elements */
        struct media_link_desc __user *links;
        __u32 reserved[4];
};

/* Interface type ranges */

#define MEDIA_INTF_T_DVB_BASE                   0x00000100
#define MEDIA_INTF_T_V4L_BASE                   0x00000200

/* Interface types */

#define MEDIA_INTF_T_DVB_FE                     (MEDIA_INTF_T_DVB_
↪BASE)
#define MEDIA_INTF_T_DVB_DEMUX                  (MEDIA_INTF_T_DVB_
↪BASE + 1)
#define MEDIA_INTF_T_DVB_DVR                    (MEDIA_INTF_T_DVB_
↪BASE + 2)
#define MEDIA_INTF_T_DVB_CA                     (MEDIA_INTF_T_DVB_
↪BASE + 3)
#define MEDIA_INTF_T_DVB_NET                    (MEDIA_INTF_T_DVB_
↪BASE + 4)

#define MEDIA_INTF_T_V4L_VIDEO                  (MEDIA_INTF_T_V4L_
↪BASE)
#define MEDIA_INTF_T_V4L_VBI                    (MEDIA_INTF_T_V4L_
↪BASE + 1)
#define MEDIA_INTF_T_V4L_RADIO                  (MEDIA_INTF_T_V4L_
↪BASE + 2)
#define MEDIA_INTF_T_V4L_SUBDEV                 (MEDIA_INTF_T_V4L_
↪BASE + 3)
#define MEDIA_INTF_T_V4L_SWRADIO                (MEDIA_INTF_T_V4L_
↪BASE + 4)
#define MEDIA_INTF_T_V4L_TOUCH                  (MEDIA_INTF_T_V4L_
↪BASE + 5)

#define MEDIA_INTF_T_ALSA_BASE                  0x00000300
#define MEDIA_INTF_T_ALSA_PCM_CAPTURE           (MEDIA_INTF_T_ALSA_
↪BASE)
#define MEDIA_INTF_T_ALSA_PCM_PLAYBACK          (MEDIA_INTF_T_ALSA_
↪BASE + 1)
#define MEDIA_INTF_T_ALSA_CONTROL               (MEDIA_INTF_T_ALSA_
```

```
↪BASE + 2)

#if defined(__KERNEL__)

/*
 * Connector functions
 *
 * For now these should not be used in userspace, as some␣
↪definitions may
 * change.
 *
 * It is the responsibility of the entity drivers to add connectors␣
↪and links.
 */
#define MEDIA_ENT_F_CONN_RF                     (MEDIA_ENT_F_BASE +␣
↪0x30001)
#define MEDIA_ENT_F_CONN_SVIDEO                 (MEDIA_ENT_F_BASE +␣
↪0x30002)
#define MEDIA_ENT_F_CONN_COMPOSITE              (MEDIA_ENT_F_BASE +␣
↪0x30003)

#endif

/*
 * MC next gen API definitions
 */

/*
 * Appeared in 4.19.0.
 *
 * The media_version argument comes from the media_version field in
 * struct media_device_info.
 */
#define MEDIA_V2_ENTITY_HAS_FLAGS(media_version) \
        ((media_version) >= ((4 << 16) | (19 << 8) | 0))

struct media_v2_entity {
        __u32 id;
        char name[64];
        __u32 function;         /* Main function of the entity */
        __u32 flags;
        __u32 reserved[5];
} __attribute__ ((packed));

/* Should match the specific fields at media_intf_devnode */
struct media_v2_intf_devnode {
        __u32 major;
        __u32 minor;
} __attribute__ ((packed));

struct media_v2_interface {
```

```
        __u32 id;
        __u32 intf_type;
        __u32 flags;
        __u32 reserved[9];

        union {
                struct media_v2_intf_devnode devnode;
                __u32 raw[16];
        };
} __attribute__ ((packed));

/*
 * Appeared in 4.19.0.
 *
 * The media_version argument comes from the media_version field in
 * struct media_device_info.
 */
#define MEDIA_V2_PAD_HAS_INDEX(media_version) \
        ((media_version) >= ((4 << 16) | (19 << 8) | 0))

struct media_v2_pad {
        __u32 id;
        __u32 entity_id;
        __u32 flags;
        __u32 index;
        __u32 reserved[4];
} __attribute__ ((packed));

struct media_v2_link {
        __u32 id;
        __u32 source_id;
        __u32 sink_id;
        __u32 flags;
        __u32 reserved[6];
} __attribute__ ((packed));

struct media_v2_topology {
        __u64 topology_version;

        __u32 num_entities;
        __u32 reserved1;
        __u64 ptr_entities;

        __u32 num_interfaces;
        __u32 reserved2;
        __u64 ptr_interfaces;

        __u32 num_pads;
        __u32 reserved3;
        __u64 ptr_pads;
```

```
        __u32 num_links;
        __u32 reserved4;
        __u64 ptr_links;
} __attribute__ ((packed));

/* ioctls */

#define MEDIA_IOC_DEVICE_INFO   _IOWR('|', 0x00, struct media_
↪device_info)
#define MEDIA_IOC_ENUM_ENTITIES _IOWR('|', 0x01, struct media_
↪entity_desc)
#define MEDIA_IOC_ENUM_LINKS    _IOWR('|', 0x02, struct media_links_
↪enum)
#define MEDIA_IOC_SETUP_LINK    _IOWR('|', 0x03, struct media_link_
↪desc)
#define MEDIA_IOC_G_TOPOLOGY    _IOWR('|', 0x04, struct media_v2_
↪topology)
#define MEDIA_IOC_REQUEST_ALLOC _IOR ('|', 0x05, int)

/*
 * These ioctls are called on the request file descriptor as␣
↪returned
 * by MEDIA_IOC_REQUEST_ALLOC.
 */
#define MEDIA_REQUEST_IOC_QUEUE         _IO('|',  0x80)
#define MEDIA_REQUEST_IOC_REINIT        _IO('|',  0x81)

#ifndef __KERNEL__

/*
 * Legacy symbols used to avoid userspace compilation breakages.
 * Do not use any of this in new applications!
 *
 * Those symbols map the entity function into types and should be
 * used only on legacy programs for legacy hardware. Don't rely
 * on those for MEDIA_IOC_G_TOPOLOGY.
 */
#define MEDIA_ENT_TYPE_SHIFT                    16
#define MEDIA_ENT_TYPE_MASK                     0x00ff0000
#define MEDIA_ENT_SUBTYPE_MASK                  0x0000ffff

#define MEDIA_ENT_T_DEVNODE_UNKNOWN             (MEDIA_ENT_F_OLD_
↪BASE | \
                                                 MEDIA_ENT_SUBTYPE_
↪MASK)

#define MEDIA_ENT_T_DEVNODE                     MEDIA_ENT_F_OLD_BASE
#define MEDIA_ENT_T_DEVNODE_V4L                 MEDIA_ENT_F_IO_V4L
#define MEDIA_ENT_T_DEVNODE_FB                  (MEDIA_ENT_F_OLD_
↪BASE + 2)
#define MEDIA_ENT_T_DEVNODE_ALSA                (MEDIA_ENT_F_OLD_
```

```
↪BASE + 3)
#define MEDIA_ENT_T_DEVNODE_DVB                 (MEDIA_ENT_F_OLD_
↪BASE + 4)

#define MEDIA_ENT_T_UNKNOWN                     MEDIA_ENT_F_UNKNOWN
#define MEDIA_ENT_T_V4L2_VIDEO                  MEDIA_ENT_F_IO_V4L
#define MEDIA_ENT_T_V4L2_SUBDEV                 MEDIA_ENT_F_V4L2_
↪SUBDEV_UNKNOWN
#define MEDIA_ENT_T_V4L2_SUBDEV_SENSOR          MEDIA_ENT_F_CAM_
↪SENSOR
#define MEDIA_ENT_T_V4L2_SUBDEV_FLASH           MEDIA_ENT_F_FLASH
#define MEDIA_ENT_T_V4L2_SUBDEV_LENS            MEDIA_ENT_F_LENS
#define MEDIA_ENT_T_V4L2_SUBDEV_DECODER         MEDIA_ENT_F_ATV_
↪DECODER
#define MEDIA_ENT_T_V4L2_SUBDEV_TUNER           MEDIA_ENT_F_TUNER

#define MEDIA_ENT_F_DTV_DECODER                 MEDIA_ENT_F_DV_
↪DECODER

/*
 * There is still no full ALSA support in the media controller.␣
↪These
 * defines should not have been added and we leave them here only
 * in case some application tries to use these defines.
 *
 * The ALSA defines that are in use have been moved into __KERNEL__
 * scope. As support gets added to these interface types, they␣
↪should
 * be moved into __KERNEL__ scope with the code that uses them.
 */
#define MEDIA_INTF_T_ALSA_COMPRESS              (MEDIA_INTF_T_ALSA_
↪BASE + 3)
#define MEDIA_INTF_T_ALSA_RAWMIDI               (MEDIA_INTF_T_ALSA_
↪BASE + 4)
#define MEDIA_INTF_T_ALSA_HWDEP                 (MEDIA_INTF_T_ALSA_
↪BASE + 5)
#define MEDIA_INTF_T_ALSA_SEQUENCER             (MEDIA_INTF_T_ALSA_
↪BASE + 6)
#define MEDIA_INTF_T_ALSA_TIMER                 (MEDIA_INTF_T_ALSA_
↪BASE + 7)

/* Obsolete symbol for media_version, no longer used in the kernel␣
↪*/
#define MEDIA_API_VERSION                       ((0 << 16) | (1 <<␣
↪8) | 0)

#endif

#endif /* __LINUX_MEDIA_H */
```

8.5.7 Revision and Copyright

Authors:

- Pinchart, Laurent <laurent.pinchart@ideasonboard.com>
- Initial version.
- Carvalho Chehab, Mauro <mchehab@kernel.org>
- MEDIA_IOC_G_TOPOLOGY documentation and documentation improvements.

8.5.8 Revision History

revision
1.1.0 / 2015-12-12 (*mcc*)

revision
1.0.0 / 2010-11-10 (*lp*)

Initial revision

8.6 Part V - Consumer Electronics Control API

This part describes the CEC: Consumer Electronics Control

8.6.1 Introduction

HDMI connectors provide a single pin for use by the Consumer Electronics Control protocol. This protocol allows different devices connected by an HDMI cable to communicate. The protocol for CEC version 1.4 is defined in supplements 1 (CEC) and 2 (HEAC or HDMI Ethernet and Audio Return Channel) of the HDMI 1.4a (*HDMI*) specification and the extensions added to CEC version 2.0 are defined in chapter 11 of the HDMI 2.0 (*HDMI2*) specification.

The bitrate is very slow (effectively no more than 36 bytes per second) and is based on the ancient AV.link protocol used in old SCART connectors. The protocol closely resembles a crazy Rube Goldberg contraption and is an unholy mix of low and high level messages. Some messages, especially those part of the HEAC protocol layered on top of CEC, need to be handled by the kernel, others can be handled either by the kernel or by userspace.

In addition, CEC can be implemented in HDMI receivers, transmitters and in USB devices that have an HDMI input and an HDMI output and that control just the CEC pin.

Drivers that support CEC will create a CEC device node (/dev/cecX) to give userspace access to the CEC adapter. The *ioctl CEC_ADAP_G_CAPS* ioctl will tell userspace what it is allowed to do.

In order to check the support and test it, it is suggested to download the v4l-utils package. It provides three tools to handle CEC:

- cec-ctl: the Swiss army knife of CEC. Allows you to configure, transmit and monitor CEC messages.
- cec-compliance: does a CEC compliance test of a remote CEC device to determine how compliant the CEC implementation is.
- cec-follower: emulates a CEC follower.

8.6.2 Function Reference

cec open()

Name

cec-open - Open a cec device

Synopsis

```
#include <fcntl.h>
```

int **open**(const char *device_name, int flags)

Arguments

device_name
: Device to be opened.

flags
: Open flags. Access mode must be `O_RDWR`.

  When the `O_NONBLOCK` flag is given, the *CEC_RECEIVE* and *CEC_DQEVENT* ioctls will return the `EAGAIN` error code when no message or event is available, and ioctls *CEC_TRANSMIT*, *CEC_ADAP_S_PHYS_ADDR* and *CEC_ADAP_S_LOG_ADDRS* all return 0.

  Other flags have no effect.

Description

To open a cec device applications call *open()* with the desired device name. The function has no side effects; the device configuration remain unchanged.

When the device is opened in read-only mode, attempts to modify its configuration will result in an error, and `errno` will be set to EBADF.

Return Value

open() returns the new file descriptor on success. On error, -1 is returned, and `errno` is set appropriately. Possible error codes include:

EACCES
The requested access to the file is not allowed.

EMFILE
The process already has the maximum number of files open.

ENFILE
The system limit on the total number of open files has been reached.

ENOMEM
Insufficient kernel memory was available.

ENXIO
No device corresponding to this device special file exists.

cec close()

Name

cec-close - Close a cec device

Synopsis

```
#include <unistd.h>
```

int **close**(int fd)

Arguments

fd
File descriptor returned by *open()*.

Description

Closes the cec device. Resources associated with the file descriptor are freed. The device configuration remain unchanged.

Return Value

close() returns 0 on success. On error, -1 is returned, and `errno` is set appropriately. Possible error codes are:

EBADF
: `fd` is not a valid open file descriptor.

cec ioctl()

Name

cec-ioctl - Control a cec device

Synopsis

```
#include <sys/ioctl.h>
```

`int ioctl(int fd, int request, void *argp)`

Arguments

fd
: File descriptor returned by *open()*.

request
: CEC ioctl request code as defined in the cec.h header file, for example *CEC_ADAP_G_CAPS*.

argp
: Pointer to a request-specific structure.

Description

The `ioctl()` function manipulates cec device parameters. The argument `fd` must be an open file descriptor.

The ioctl `request` code specifies the cec function to be called. It has encoded in it whether the argument is an input, output or read/write parameter, and the size of the argument `argp` in bytes.

Macros and structures definitions specifying cec ioctl requests and their parameters are located in the cec.h header file. All cec ioctl requests, their respective function and parameters are specified in *Function Reference*.

Return Value

On success 0 is returned, on error -1 and the `errno` variable is set appropriately. The generic error codes are described at the *Generic Error Codes* chapter.

Request-specific error codes are listed in the individual requests descriptions.

When an ioctl that takes an output or read/write parameter fails, the parameter remains unmodified.

cec poll()

Name

cec-poll - Wait for some event on a file descriptor

Synopsis

```
#include <sys/poll.h>
```

int **poll**(struct pollfd *ufds, unsigned int nfds, int timeout)

Arguments

ufds
List of FD events to be watched

nfds
Number of FD events at the *ufds array

timeout
Timeout to wait for events

Description

With the *poll()* function applications can wait for CEC events.

On success *poll()* returns the number of file descriptors that have been selected (that is, file descriptors for which the `revents` field of the respective struct `pollfd` is non-zero). CEC devices set the `POLLIN` and `POLLRDNORM` flags in the `revents` field if there are messages in the receive queue. If the transmit queue has room for new messages, the `POLLOUT` and `POLLWRNORM` flags are set. If there are events in the event queue, then the `POLLPRI` flag is set. When the function times out it returns a value of zero, on failure it returns -1 and the `errno` variable is set appropriately.

For more details see the *poll()* manual page.

Return Value

On success, *poll()* returns the number structures which have non-zero `revents` fields, or zero if the call timed out. On error -1 is returned, and the `errno` variable is set appropriately:

EBADF
: One or more of the `ufds` members specify an invalid file descriptor.

EFAULT
: `ufds` references an inaccessible memory area.

EINTR
: The call was interrupted by a signal.

EINVAL
: The `nfds` value exceeds the `RLIMIT_NOFILE` value. Use `getrlimit()` to obtain this value.

ioctl CEC_ADAP_G_CAPS

Name

CEC_ADAP_G_CAPS - Query device capabilities

Synopsis

CEC_ADAP_G_CAPS

`int ioctl(int fd, CEC_ADAP_G_CAPS, struct cec_caps *argp)`

Arguments

fd
: File descriptor returned by *open()*.

`argp`

Description

All cec devices must support *ioctl CEC_ADAP_G_CAPS*. To query device information, applications call the ioctl with a pointer to a struct *cec_caps*. The driver fills the structure and returns the information to the application. The ioctl never fails.

type **cec_caps**

Table 252: struct cec_caps

char	driver[32]	The name of the cec adapter driver.
char	name[32]	The name of this CEC adapter. The combination driver and name must be unique.
__u32	available_l	The maximum number of logical addresses that can be configured.
__u32	capabilitie	The capabilities of the CEC adapter, see *CEC Capabilities Flags*.
__u32	version	CEC Framework API version, formatted with the KERNEL_VERSION() macro.

Table 253: CEC Capabilities Flags

CEC_CAP_PHYS_ADDR	0x00000001	Userspace has to configure the physical address by calling *ioctl CEC_ADAP_S_PHYS_ADDR*. If this capability isn't set, then setting the physical address is handled by the kernel whenever the EDID is set (for an HDMI receiver) or read (for an HDMI transmitter).
CEC_CAP_LOG_ADDRS	0x00000002	Userspace has to configure the logical addresses by calling *ioctl CEC_ADAP_S_LOG_ADDRS*. If this capability isn't set, then the kernel will have configured this.
CEC_CAP_TRANSMIT	0x00000004	Userspace can transmit CEC messages by calling *ioctl CEC_TRANSMIT*. This implies that userspace can be a follower as well, since being able to transmit messages is a prerequisite of becoming a follower. If this capability isn't set, then the kernel will handle all CEC transmits and process all CEC messages it receives.
CEC_CAP_PASSTHROUGH	0x00000008	Userspace can use the passthrough mode by calling *ioctl CEC_S_MODE*.
CEC_CAP_RC	0x00000010	This adapter supports the remote control protocol.
CEC_CAP_MONITOR_ALL	0x00000020	The CEC hardware can monitor all messages, not just directed and broadcast messages.
CEC_CAP_NEEDS_HPD	0x00000040	The CEC hardware is only active if the HDMI Hotplug Detect pin is high. This makes it impossible to use CEC to wake up displays that set the HPD pin low when in standby mode, but keep the CEC bus alive.
CEC_CAP_MONITOR_PIN	0x00000080	The CEC hardware can monitor CEC pin changes from low to high voltage and vice versa. When in pin monitoring mode the application will receive CEC_EVENT_PIN_CEC_LOW and CEC_EVENT_PIN_CEC_HIGH events.
CEC_CAP_CONNECTOR_I	0x00000100	If this capability is set, then *ioctl CEC_ADAP_G_CONNECTOR_INFO* can be used.

Return Value

On success 0 is returned, on error -1 and the `errno` variable is set appropriately. The generic error codes are described at the *Generic Error Codes* chapter.

ioctls CEC_ADAP_G_LOG_ADDRS and CEC_ADAP_S_LOG_ADDRS

Name

CEC_ADAP_G_LOG_ADDRS, CEC_ADAP_S_LOG_ADDRS - Get or set the logical addresses

Synopsis

CEC_ADAP_G_LOG_ADDRS

`int ioctl(int fd, CEC_ADAP_G_LOG_ADDRS, struct cec_log_addrs *argp)`

CEC_ADAP_S_LOG_ADDRS

`int ioctl(int fd, CEC_ADAP_S_LOG_ADDRS, struct cec_log_addrs *argp)`

Arguments

fd
File descriptor returned by *open()*.

argp
Pointer to struct *cec_log_addrs*.

Description

To query the current CEC logical addresses, applications call *ioctl CEC_ADAP_G_LOG_ADDRS* with a pointer to a struct *cec_log_addrs* where the driver stores the logical addresses.

To set new logical addresses, applications fill in struct *cec_log_addrs* and call *ioctl CEC_ADAP_S_LOG_ADDRS* with a pointer to this struct. The *ioctl CEC_ADAP_S_LOG_ADDRS* is only available if `CEC_CAP_LOG_ADDRS` is set (the `ENOTTY` error code is returned otherwise). The *ioctl CEC_ADAP_S_LOG_ADDRS* can only be called by a file descriptor in initiator mode (see *ioctls CEC_G_MODE and CEC_S_MODE*), if not the `EBUSY` error code will be returned.

To clear existing logical addresses set `num_log_addrs` to 0. All other fields will be ignored in that case. The adapter will go to the unconfigured state and the `cec_version`, `vendor_id` and `osd_name` fields are all reset to their default values (CEC version 2.0, no vendor ID and an empty OSD name).

If the physical address is valid (see *ioctl CEC_ADAP_S_PHYS_ADDR*), then this ioctl will block until all requested logical addresses have been claimed. If the file descriptor is in non-blocking mode then it will not wait for the logical addresses to be claimed, instead it just returns 0.

A *CEC_EVENT_STATE_CHANGE* event is sent when the logical addresses are claimed or cleared.

Attempting to call *ioctl CEC_ADAP_S_LOG_ADDRS* when logical address types are already defined will return with error EBUSY.

type **cec_log_addrs**

Table 254: struct cec_log_addrs

__u8	log_addr[CEC_MAX_LOG_ADDRS]	The actual logical addresses that were claimed. This is set by the driver. If no logical address could be claimed, then it is set to CEC_LOG_ADDR_INVALID. If this adapter is Unregistered, then log_addr[0] is set to 0xf and all others to CEC_LOG_ADDR_INVALID.
__u16	log_addr_mask	The bitmask of all logical addresses this adapter has claimed. If this adapter is Unregistered then log_addr_mask sets bit 15 and clears all other bits. If this adapter is not configured at all, then log_addr_mask is set to 0. Set by the driver.
__u8	cec_version	The CEC version that this adapter shall use. See *CEC Versions*. Used to implement the CEC_MSG_CEC_VERSION and CEC_MSG_REPORT_FEATURES messages. Note that *CEC_OP_CEC_VERSION_1_3A* is not allowed by the CEC framework.
__u8	num_log_addrs	Number of logical addresses to set up. Must be ≤ available_log_addrs as returned by *ioctl CEC_ADAP_G_CAPS*. All arrays in this structure are only filled up to index available_log_addrs-1. The remaining array elements will be ignored. Note that the CEC 2.0 standard allows for a maximum of 2 logical addresses, although some hardware has support for more. CEC_MAX_LOG_ADDRS is 4. The driver will return the actual number of logical addresses it could claim, which may be less than what was requested. If this field is set to 0, then the CEC adapter shall clear all claimed logical addresses and all other fields will be ignored.

continues on next page

Table 254 – continued from previous page

__u32	vendor_id	The vendor ID is a 24-bit number that identifies the specific vendor or entity. Based on this ID vendor specific commands may be defined. If you do not want a vendor ID then set it to CEC_VENDOR_ID_NONE.
__u32	flags	Flags. See *Flags for struct cec_log_addrs* for a list of available flags.
char	osd_name[15]	The On-Screen Display name as is returned by the CEC_MSG_SET_OSD_NAME message.
__u8	primary_device_type[CEC_MAX_LOG_ADD	Primary device type for each logical address. See *CEC Primary Device Types* for possible types.
__u8	log_addr_type[CEC_MAX_LOG_ADDRS]	Logical address types. See *CEC Logical Address Types* for possible types. The driver will update this with the actual logical address type that it claimed (e.g. it may have to fallback to *CEC_LOG_ADDR_TYPE_UNREGISTERED*).
__u8	all_device_types[CEC_MAX_LOG_ADDRS]	CEC 2.0 specific: the bit mask of all device types. See *CEC All Device Types Flags*. It is used in the CEC 2.0 CEC_MSG_REPORT_FEATURES message. For CEC 1.4 you can either leave this field to 0, or fill it in according to the CEC 2.0 guidelines to give the CEC framework more information about the device type, even though the framework won't use it directly in the CEC message.
__u8	features[CEC_MAX_LOG_ADDRS][12]	Features for each logical address. It is used in the CEC 2.0 CEC_MSG_REPORT_FEATURES message. The 12 bytes include both the RC Profile and the Device Features. For CEC 1.4 you can either leave this field to all 0, or fill it in according to the CEC 2.0 guidelines to give the CEC framework more information about the device type, even though the framework won't use it directly in the CEC message.

Table 255: Flags for struct cec_log_addrs

CEC_LOG_ADDRS_FL_ALLOW_UNREG_FALLB	1	By default if no logical address of the requested type can be claimed, then it will go back to the unconfigured state. If this flag is set, then it will fallback to the Unregistered logical address. Note that if the Unregistered logical address was explicitly requested, then this flag has no effect.
CEC_LOG_ADDRS_FL_ALLOW_RC_PASSTHRU	2	By default the CEC_MSG_USER_CONTROL_PRESSED and CEC_MSG_USER_CONTROL_RELEASED messages are only passed on to the follower(s), if any. If this flag is set, then these messages are also passed on to the remote control input subsystem and will appear as keystrokes. This features needs to be enabled explicitly. If CEC is used to enter e.g. passwords, then you may not want to enable this to avoid trivial snooping of the keystrokes.
CEC_LOG_ADDRS_FL_CDC_ONLY	4	If this flag is set, then the device is CDC-Only. CDC-Only CEC devices are CEC devices that can only handle CDC messages. All other messages are ignored.

Table 256: CEC Versions

CEC_OP_CEC_VERSION_1_3A	4	CEC version according to the HDMI 1.3a standard.
CEC_OP_CEC_VERSION_1_4B	5	CEC version according to the HDMI 1.4b standard.
CEC_OP_CEC_VERSION_2_0	6	CEC version according to the HDMI 2.0 standard.

Table 257: CEC Primary Device Types

CEC_OP_PRIM_DEVTYPE_TV	0	Use for a TV.
CEC_OP_PRIM_DEVTYPE_RECORD	1	Use for a recording device.
CEC_OP_PRIM_DEVTYPE_TUNER	3	Use for a device with a tuner.
CEC_OP_PRIM_DEVTYPE_PLAYBACK	4	Use for a playback device.
CEC_OP_PRIM_DEVTYPE_AUDIOSYST	5	Use for an audio system (e.g. an audio/video receiver).
CEC_OP_PRIM_DEVTYPE_SWITCH	6	Use for a CEC switch.
CEC_OP_PRIM_DEVTYPE_VIDEOPROC	7	Use for a video processor device.

Table 258: CEC Logical Address Types

CEC_LOG_ADDR_TYPE_TV	0	Use for a TV.
CEC_LOG_ADDR_TYPE_RECORD	1	Use for a recording device.
CEC_LOG_ADDR_TYPE_TUNER	2	Use for a tuner device.
CEC_LOG_ADDR_TYPE_PLAYBACK	3	Use for a playback device.
CEC_LOG_ADDR_TYPE_AUDIOSYSTEM	4	Use for an audio system device.
CEC_LOG_ADDR_TYPE_SPECIFIC	5	Use for a second TV or for a video processor device.
CEC_LOG_ADDR_TYPE_UNREGISTERE	6	Use this if you just want to remain unregistered. Used for pure CEC switches or CDC-only devices (CDC: Capability Discovery and Control).

Table 259: CEC All Device Types Flags

CEC_OP_ALL_DEVTYPE_TV	0x80	This supports the TV type.
CEC_OP_ALL_DEVTYPE_RECORD	0x40	This supports the Recording type.
CEC_OP_ALL_DEVTYPE_TUNER	0x20	This supports the Tuner type.
CEC_OP_ALL_DEVTYPE_PLAYBACK	0x10	This supports the Playback type.
CEC_OP_ALL_DEVTYPE_AUDIOSYSTE	0x08	This supports the Audio System type.
CEC_OP_ALL_DEVTYPE_SWITCH	0x04	This supports the CEC Switch or Video Processing type.

Return Value

On success 0 is returned, on error -1 and the `errno` variable is set appropriately. The generic error codes are described at the *Generic Error Codes* chapter.

The *ioctl CEC_ADAP_S_LOG_ADDRS* can return the following error codes:

ENOTTY
The `CEC_CAP_LOG_ADDRS` capability wasn't set, so this ioctl is not supported.

EBUSY
The CEC adapter is currently configuring itself, or it is already configured and `num_log_addrs` is non-zero, or another filehandle is in exclusive follower or initiator mode, or the filehandle is in mode `CEC_MODE_NO_INITIATOR`.

EINVAL
The contents of struct *cec_log_addrs* is invalid.

ioctls CEC_ADAP_G_PHYS_ADDR and CEC_ADAP_S_PHYS_ADDR

Name

CEC_ADAP_G_PHYS_ADDR, CEC_ADAP_S_PHYS_ADDR - Get or set the physical address

Synopsis

CEC_ADAP_G_PHYS_ADDR

`int ioctl(int fd, CEC_ADAP_G_PHYS_ADDR, __u16 *argp)`

CEC_ADAP_S_PHYS_ADDR

`int ioctl(int fd, CEC_ADAP_S_PHYS_ADDR, __u16 *argp)`

Arguments

fd
File descriptor returned by *open()*.

argp
Pointer to the CEC address.

Description

To query the current physical address applications call *ioctl CEC_ADAP_G_PHYS_ADDR* with a pointer to a __u16 where the driver stores the physical address.

To set a new physical address applications store the physical address in a __u16 and call *ioctl CEC_ADAP_S_PHYS_ADDR* with a pointer to this integer. The *ioctl CEC_ADAP_S_PHYS_ADDR* is only available if `CEC_CAP_PHYS_ADDR` is set (the `ENOTTY` error code will be returned otherwise). The *ioctl CEC_ADAP_S_PHYS_ADDR* can only be called by a file descriptor in initiator mode (see *ioctls CEC_G_MODE and CEC_S_MODE*), if not the `EBUSY` error code will be returned.

To clear an existing physical address use `CEC_PHYS_ADDR_INVALID`. The adapter will go to the unconfigured state.

If logical address types have been defined (see *ioctl CEC_ADAP_S_LOG_ADDRS*), then this ioctl will block until all requested logical addresses have been claimed. If the file descriptor is in non-blocking mode then it will not wait for the logical addresses to be claimed, instead it just returns 0.

A *CEC_EVENT_STATE_CHANGE* event is sent when the physical address changes.

The physical address is a 16-bit number where each group of 4 bits represent a digit of the physical address a.b.c.d where the most significant 4 bits represent 'a' . The CEC root device (usually the TV) has address 0.0.0.0. Every device that is hooked up to an input of the TV has address a.0.0.0 (where 'a' is ≥ 1), devices hooked up to those in turn have addresses a.b.0.0, etc. So a topology of up to 5 devices deep is supported. The physical address a device shall use is stored in the EDID of the sink.

For example, the EDID for each HDMI input of the TV will have a different physical address of the form a.0.0.0 that the sources will read out and use as their physical address.

Return Value

On success 0 is returned, on error -1 and the `errno` variable is set appropriately. The generic error codes are described at the *Generic Error Codes* chapter.

The *ioctl CEC_ADAP_S_PHYS_ADDR* can return the following error codes:

ENOTTY
The `CEC_CAP_PHYS_ADDR` capability wasn't set, so this ioctl is not supported.

EBUSY
Another filehandle is in exclusive follower or initiator mode, or the filehandle is in mode `CEC_MODE_NO_INITIATOR`.

EINVAL
The physical address is malformed.

ioctl CEC_ADAP_G_CONNECTOR_INFO

Name

CEC_ADAP_G_CONNECTOR_INFO - Query HDMI connector information

Synopsis

CEC_ADAP_G_CONNECTOR_INFO

`int ioctl(int fd, CEC_ADAP_G_CONNECTOR_INFO, struct cec_connector_info *argp)`

Arguments

fd
File descriptor returned by *open()*.

`argp`

Description

Using this ioctl an application can learn which HDMI connector this CEC device corresponds to. While calling this ioctl the application should provide a pointer to a cec_connector_info struct which will be populated by the kernel with the info provided by the adapter's driver. This ioctl is only available if the `CEC_CAP_CONNECTOR_INFO` capability is set.

type **cec_connector_info**

Table 260: struct cec_connector_info

__u32	type	The type of connector this adapter is associated with.
union {	(anonymous)	
struc cec_c	drm	*struct cec_drm_connector_info*
}		

Table 261: Connector types

CEC_CONNECTOR_TYPE_I	0	No connector is associated with the adapter/the information is not provided by the driver.
CEC_CONNECTOR_TYPE_I	1	Indicates that a DRM connector is associated with this adapter. Information about the connector can be found in *struct cec_drm_connector_info*.

type **cec_drm_connector_info**

Table 262: struct cec_drm_connector_info

__u32	card_no	DRM card number: the number from a card's path, e.g. 0 in case of /dev/card0.
__u32	connector_i	DRM connector ID.

ioctl CEC_DQEVENT

Name

CEC_DQEVENT - Dequeue a CEC event

Synopsis

CEC_DQEVENT

`int ioctl(int fd, CEC_DQEVENT, struct cec_event *argp)`

Arguments

fd
File descriptor returned by *open()*.

`argp`

Description

CEC devices can send asynchronous events. These can be retrieved by calling *CEC_DQEVENT()*. If the file descriptor is in non-blocking mode and no event is pending, then it will return -1 and set errno to the `EAGAIN` error code.

The internal event queues are per-filehandle and per-event type. If there is no more room in a queue then the last event is overwritten with the new one. This means that intermediate results can be thrown away but that the latest event is always available. This also means that is it possible to read two successive events that have the same value (e.g. two *CEC_EVENT_STATE_CHANGE* events with the same state). In that case the intermediate state changes were lost but it is guaranteed that the state did change in between the two events.

type **cec_event_state_change**

Table 263: struct cec_event_state_change

__u16	phys_addr	The current physical address. This is CEC_PHYS_ADDR_INVALID if no valid physical address is set.
__u16	log_addr_mask	The current set of claimed logical addresses. This is 0 if no logical addresses are claimed or if phys_addr is CEC_PHYS_ADDR_INVALID. If bit 15 is set (1 << CEC_LOG_ADDR_UNREGISTERED) then this device has the unregistered logical address. In that case all other bits are 0.
__u16	have_conn_inf	If non-zero, then HDMI connector information is available. This field is only valid if CEC_CAP_CONNECTOR_INFO is set. If that capability is set and have_conn_info is zero, then that indicates that the HDMI connector device is not instantiated, either because the HDMI driver is still configuring the device or because the HDMI device was unbound.

type **cec_event_lost_msgs**

Table 264: struct cec_event_lost_msgs

__u32	lost_msgs	Set to the number of lost messages since the filehandle was opened or since the last time this event was dequeued for this filehandle. The messages lost are the oldest messages. So when a new message arrives and there is no more room, then the oldest message is discarded to make room for the new one. The internal size of the message queue guarantees that all messages received in the last two seconds will be stored. Since messages should be replied to within a second according to the CEC specification, this is more than enough.

type **cec_event**

Table 265: struct cec_event

__u64	ts	Timestamp of the event in ns. The timestamp has been taken from the CLOCK_MONOTONIC clock. To access the same clock from userspace use clock_gettime().
__u32	event	The CEC event type, see *CEC Events Types*.
__u32	flags	Event flags, see *CEC Event Flags*.
union {	(anonymous)	
struct cec_e	state_change	The new adapter state as sent by the *CEC_EVENT_STATE_CHANGE* event.
struct cec_e	lost_msgs	The number of lost messages as sent by the *CEC_EVENT_LOST_MSGS* event.
}		

Table 266: CEC Events Types

CEC_EVENT_STATE_CHANGE	1	Generated when the CEC Adapter's state changes. When open() is called an initial event will be generated for that filehandle with the CEC Adapter' s state at that time.
CEC_EVENT_LOST_MSGS	2	Generated if one or more CEC messages were lost because the application didn' t dequeue CEC messages fast enough.
CEC_EVENT_PIN_CEC_LOW	3	Generated if the CEC pin goes from a high voltage to a low voltage. Only applies to adapters that have the CEC_CAP_MONITOR_PIN capability set.
CEC_EVENT_PIN_CEC_HIGH	4	Generated if the CEC pin goes from a low voltage to a high voltage. Only applies to adapters that have the CEC_CAP_MONITOR_PIN capability set.
CEC_EVENT_PIN_HPD_LOW	5	Generated if the HPD pin goes from a high voltage to a low voltage. Only applies to adapters that have the CEC_CAP_MONITOR_PIN capability set. When open() is called, the HPD pin can be read and if the HPD is low, then an initial event will be generated for that filehandle.
CEC_EVENT_PIN_HPD_HIGH	6	Generated if the HPD pin goes from a low voltage to a high voltage. Only applies to adapters that have the CEC_CAP_MONITOR_PIN capability set. When open() is called, the HPD pin can be read and if the HPD is high, then an initial event will be generated for that filehandle.
CEC_EVENT_PIN_5V_LOW	6	Generated if the 5V pin goes from a high voltage to a low voltage. Only applies to adapters that have the CEC_CAP_MONITOR_PIN capability set. When open() is called, the 5V pin can be read and if the 5V is low, then an initial event will be generated for that filehandle.
CEC_EVENT_PIN_5V_HIGH	7	Generated if the 5V pin goes from a low voltage to a high voltage. Only applies to adapters that have the CEC_CAP_MONITOR_PIN capability set. When open() is called, the 5V pin can be read and if the 5V is high, then an initial event will be generated for that filehandle.

Table 267: CEC Event Flags

CEC_EVENT_FL_INITIAL_STATE	1	Set for the initial events that are generated when the device is opened. See the table above for which events do this. This allows applications to learn the initial state of the CEC adapter at open() time.
CEC_EVENT_FL_DROPPED_EVENTS	2	Set if one or more events of the given event type have been dropped. This is an indication that the application cannot keep up.

Return Value

On success 0 is returned, on error -1 and the `errno` variable is set appropriately. The generic error codes are described at the *Generic Error Codes* chapter.

The *ioctl CEC_DQEVENT* can return the following error codes:

EAGAIN
: This is returned when the filehandle is in non-blocking mode and there are no pending events.

ERESTARTSYS
: An interrupt (e.g. Ctrl-C) arrived while in blocking mode waiting for events to arrive.

ioctls CEC_G_MODE and CEC_S_MODE

CEC_G_MODE, CEC_S_MODE - Get or set exclusive use of the CEC adapter

Synopsis

CEC_G_MODE

`int ioctl(int fd, CEC_G_MODE, __u32 *argp)`

CEC_S_MODE

`int ioctl(int fd, CEC_S_MODE, __u32 *argp)`

Arguments

fd
: File descriptor returned by *open()*.

argp
: Pointer to CEC mode.

Description

By default any filehandle can use *ioctls CEC_RECEIVE and CEC_TRANSMIT*, but in order to prevent applications from stepping on each others toes it must be possible to obtain exclusive access to the CEC adapter. This ioctl sets the filehandle to initiator and/or follower mode which can be exclusive depending on the chosen mode. The initiator is the filehandle that is used to initiate messages, i.e. it commands other CEC devices. The follower is the filehandle that receives messages sent to the CEC adapter and processes them. The same filehandle can be both initiator and follower, or this role can be taken by two different filehandles.

When a CEC message is received, then the CEC framework will decide how it will be processed. If the message is a reply to an earlier transmitted message, then the reply is sent back to the filehandle that is waiting for it. In addition the CEC framework will process it.

If the message is not a reply, then the CEC framework will process it first. If there is no follower, then the message is just discarded and a feature abort is sent back to the initiator if the framework couldn't process it. If there is a follower, then the message is passed on to the follower who will use *ioctl CEC_RECEIVE* to dequeue the new message. The framework expects the follower to make the right decisions.

The CEC framework will process core messages unless requested otherwise by the follower. The follower can enable the passthrough mode. In that case, the CEC framework will pass on most core messages without processing them and the follower will have to implement those messages. There are some messages that the core will always process, regardless of the passthrough mode. See *Core Message Processing* for details.

If there is no initiator, then any CEC filehandle can use *ioctl CEC_TRANSMIT*. If there is an exclusive initiator then only that initiator can call *ioctls CEC_RECEIVE and CEC_TRANSMIT*. The follower can of course always call *ioctl CEC_TRANSMIT*.

Available initiator modes are:

Table 268: Initiator Modes

`CEC_MODE_NO_INITIATOR`	0x0	This is not an initiator, i.e. it cannot transmit CEC messages or make any other changes to the CEC adapter.
`CEC_MODE_INITIATOR`	0x1	This is an initiator (the default when the device is opened) and it can transmit CEC messages and make changes to the CEC adapter, unless there is an exclusive initiator.
`CEC_MODE_EXCL_INITIATOR`	0x2	This is an exclusive initiator and this file descriptor is the only one that can transmit CEC messages and make changes to the CEC adapter. If someone else is already the exclusive initiator then an attempt to become one will return the `EBUSY` error code error.

Available follower modes are:

Table 269: Follower Modes

CEC_MODE_NO_FOLLOWER	0x00	This is not a follower (the default when the device is opened).
CEC_MODE_FOLLOWER	0x10	This is a follower and it will receive CEC messages unless there is an exclusive follower. You cannot become a follower if *CEC_CAP_TRANSMIT* is not set or if *CEC_MODE_NO_INITIATOR* was specified, the EINVAL error code is returned in that case.
CEC_MODE_EXCL_FOLLOWER	0x20	This is an exclusive follower and only this file descriptor will receive CEC messages for processing. If someone else is already the exclusive follower then an attempt to become one will return the EBUSY error code. You cannot become a follower if *CEC_CAP_TRANSMIT* is not set or if *CEC_MODE_NO_INITIATOR* was specified, the EINVAL error code is returned in that case.
CEC_MODE_EXCL_FOLLOWER_PASSTH	0x30	This is an exclusive follower and only this file descriptor will receive CEC messages for processing. In addition it will put the CEC device into passthrough mode, allowing the exclusive follower to handle most core messages instead of relying on the CEC framework for that. If someone else is already the exclusive follower then an attempt to become one will return the EBUSY error code. You cannot become a follower if *CEC_CAP_TRANSMIT* is not set or if *CEC_MODE_NO_INITIATOR* was specified, the EINVAL error code is returned in that case.
CEC_MODE_MONITOR_PIN	0xd0	Put the file descriptor into pin monitoring mode. Can only be used in combination with *CEC_MODE_NO_INITIATOR*, otherwise the EINVAL error code will be returned. This mode requires that the *CEC_CAP_MONITOR_PIN* capability is set, otherwise the EINVAL error code is returned. While in pin monitoring mode this file descriptor can receive the CEC_EVENT_PIN_CEC_LOW and CEC_EVENT_PIN_CEC_HIGH events to see the low-level CEC pin transitions. This is very useful for debugging. This mode is only allowed if the process has the CAP_NET_ADMIN capability. If that is not set, then the EPERM error code is returned.

continues on next page

Table 269 – continued from previous page

CEC_MODE_MONITOR	0xe0	Put the file descriptor into monitor mode. Can only be used in combination with *CEC_MODE_NO_INITIATOR*, otherwise the EINVAL error code will be returned. In monitor mode all messages this CEC device transmits and all messages it receives (both broadcast messages and directed messages for one its logical addresses) will be reported. This is very useful for debugging. This is only allowed if the process has the CAP_NET_ADMIN capability. If that is not set, then the EPERM error code is returned.
CEC_MODE_MONITOR_ALL	0xf0	Put the file descriptor into 'monitor all' mode. Can only be used in combination with *CEC_MODE_NO_INITIATOR*, otherwise the EINVAL error code will be returned. In 'monitor all' mode all messages this CEC device transmits and all messages it receives, including directed messages for other CEC devices will be reported. This is very useful for debugging, but not all devices support this. This mode requires that the *CEC_CAP_MONITOR_ALL* capability is set, otherwise the EINVAL error code is returned. This is only allowed if the process has the CAP_NET_ADMIN capability. If that is not set, then the EPERM error code is returned.

Core message processing details:

Table 270: Core Message Processing

CEC_MSG_GET_CEC_VERSION	The core will return the CEC version that was set with *ioctl CEC_ADAP_S_LOG_ADDRS*, except when in passthrough mode. In passthrough mode the core does nothing and this message has to be handled by a follower instead.
CEC_MSG_GIVE_DEVICE_VENDOR_ID	The core will return the vendor ID that was set with *ioctl CEC_ADAP_S_LOG_ADDRS*, except when in passthrough mode. In passthrough mode the core does nothing and this message has to be handled by a follower instead.
CEC_MSG_ABORT	The core will return a Feature Abort message with reason 'Feature Refused' as per the specification, except when in passthrough mode. In passthrough mode the core does nothing and this message has to be handled by a follower instead.
CEC_MSG_GIVE_PHYSICAL_ADDR	The core will report the current physical address, except when in passthrough mode. In passthrough mode the core does nothing and this message has to be handled by a follower instead.
CEC_MSG_GIVE_OSD_NAME	The core will report the current OSD name that was set with *ioctl CEC_ADAP_S_LOG_ADDRS*, except when in passthrough mode. In passthrough mode the core does nothing and this message has to be handled by a follower instead.
CEC_MSG_GIVE_FEATURES	The core will do nothing if the CEC version is older than 2.0, otherwise it will report the current features that were set with *ioctl CEC_ADAP_S_LOG_ADDRS*, except when in passthrough mode. In passthrough mode the core does nothing (for any CEC version) and this message has to be handled by a follower instead.
CEC_MSG_USER_CONTROL_PRESSED	If *CEC_CAP_RC* is set and if *CEC_LOG_ADDRS_FL_ALLOW_RC_PASSTHRU* is set, then generate a remote control key press. This message is always passed on to the follower(s).
CEC_MSG_USER_CONTROL_RELEASED	If *CEC_CAP_RC* is set and if *CEC_LOG_ADDRS_FL_ALLOW_RC_PASSTHRU* is set, then generate a remote control key release. This message is always passed on to the follower(s).
CEC_MSG_REPORT_PHYSICAL_ADDR	The CEC framework will make note of the reported physical address and then just pass the message on to the follower(s).

Return Value

On success 0 is returned, on error -1 and the `errno` variable is set appropriately. The generic error codes are described at the *Generic Error Codes* chapter.

The *ioctl CEC_S_MODE* can return the following error codes:

EINVAL
: The requested mode is invalid.

EPERM
: Monitor mode is requested, but the process does have the `CAP_NET_ADMIN` capability.

EBUSY
: Someone else is already an exclusive follower or initiator.

ioctls CEC_RECEIVE and CEC_TRANSMIT

Name

CEC_RECEIVE, CEC_TRANSMIT - Receive or transmit a CEC message

Synopsis

`CEC_RECEIVE`

`int ioctl(int fd, CEC_RECEIVE, struct cec_msg *argp)`

`CEC_TRANSMIT`

`int ioctl(int fd, CEC_TRANSMIT, struct cec_msg *argp)`

Arguments

`fd`
: File descriptor returned by *`open()`*.

`argp`
: Pointer to struct cec_msg.

Description

To receive a CEC message the application has to fill in the `timeout` field of struct *`cec_msg`* and pass it to *ioctl CEC_RECEIVE*. If the file descriptor is in non-blocking mode and there are no received messages pending, then it will return -1 and set errno to the `EAGAIN` error code. If the file descriptor is in blocking mode and `timeout` is non-zero and no message arrived within `timeout` milliseconds, then it will return -1 and set errno to the `ETIMEDOUT` error code.

A received message can be:

1. a message received from another CEC device (the `sequence` field will be 0).
2. the result of an earlier non-blocking transmit (the `sequence` field will be non-zero).

To send a CEC message the application has to fill in the struct *cec_msg* and pass it to *ioctl CEC_TRANSMIT*. The *ioctl CEC_TRANSMIT* is only available if `CEC_CAP_TRANSMIT` is set. If there is no more room in the transmit queue, then it will return -1 and set errno to the `EBUSY` error code. The transmit queue has enough room for 18 messages (about 1 second worth of 2-byte messages). Note that the CEC kernel framework will also reply to core messages (see *Core Message Processing*), so it is not a good idea to fully fill up the transmit queue.

If the file descriptor is in non-blocking mode then the transmit will return 0 and the result of the transmit will be available via *ioctl CEC_RECEIVE* once the transmit has finished (including waiting for a reply, if requested).

The `sequence` field is filled in for every transmit and this can be checked against the received messages to find the corresponding transmit result.

Normally calling *ioctl CEC_TRANSMIT* when the physical address is invalid (due to e.g. a disconnect) will return `ENONET`.

However, the CEC specification allows sending messages from 'Unregistered' to 'TV' when the physical address is invalid since some TVs pull the hotplug detect pin of the HDMI connector low when they go into standby, or when switching to another input.

When the hotplug detect pin goes low the EDID disappears, and thus the physical address, but the cable is still connected and CEC still works. In order to detect/wake up the device it is allowed to send poll and 'Image/Text View On' messages from initiator 0xf ('Unregistered') to destination 0 ('TV').

type **cec_msg**

Table 271: struct cec_msg

__u64	tx_ts	Timestamp in ns of when the last byte of the message was transmitted. The timestamp has been taken from the `CLOCK_MONOTONIC` clock. To access the same clock from userspace use `clock_gettime()`.
__u64	rx_ts	Timestamp in ns of when the last byte of the message was received. The timestamp has been taken from the `CLOCK_MONOTONIC` clock. To access the same clock from userspace use `clock_gettime()`.
__u32	len	The length of the message. For *ioctl CEC_TRANSMIT* this is filled in by the application. The driver will fill this in for *ioctl CEC_RECEIVE*. For *ioctl CEC_TRANSMIT* it will be filled in by the driver with the length of the reply message if `reply` was set.
__u32	timeout	The timeout in milliseconds. This is the time the device will wait for a message to be received before timing out. If it is set to 0, then it will wait indefinitely when it is called by *ioctl CEC_RECEIVE*. If it is 0 and it is called by *ioctl CEC_TRANSMIT*, then it will be replaced by 1000 if the `reply` is non-zero or ignored if `reply` is 0.

continues on next page

Table 271 – continued from previous page

__u32	sequence	A non-zero sequence number is automatically assigned by the CEC framework for all transmitted messages. It is used by the CEC framework when it queues the transmit result (when transmit was called in non-blocking mode). This allows the application to associate the received message with the original transmit.
__u32	flags	Flags. See *Flags for struct cec_msg* for a list of available flags.
__u8	tx_status	The status bits of the transmitted message. See *CEC Transmit Status* for the possible status values. It is 0 if this message was received, not transmitted.
__u8	msg[16]	The message payload. For *ioctl CEC_TRANSMIT* this is filled in by the application. The driver will fill this in for *ioctl CEC_RECEIVE*. For *ioctl CEC_TRANSMIT* it will be filled in by the driver with the payload of the reply message if `timeout` was set.
__u8	reply	Wait until this message is replied. If `reply` is 0 and the `timeout` is 0, then don't wait for a reply but return after transmitting the message. Ignored by *ioctl CEC_RECEIVE*. The case where `reply` is 0 (this is the opcode for the Feature Abort message) and `timeout` is non-zero is specifically allowed to make it possible to send a message and wait up to `timeout` milliseconds for a Feature Abort reply. In this case `rx_status` will either be set to *CEC_RX_STATUS_TIMEOUT* or *CEC_RX_STATUS_FEATURE_ABORT*. If the transmitter message is `CEC_MSG_INITIATE_ARC` then the `reply` values `CEC_MSG_REPORT_ARC_INITIATED` and `CEC_MSG_REPORT_ARC_TERMINATED` are processed differently: either value will match both possible replies. The reason is that the `CEC_MSG_INITIATE_ARC` message is the only CEC message that has two possible replies other than Feature Abort. The `reply` field will be updated with the actual reply so that it is synchronized with the contents of the received message.
__u8	rx_status	The status bits of the received message. See *CEC Receive Status* for the possible status values. It is 0 if this message was transmitted, not received, unless this is the reply to a transmitted message. In that case both `rx_status` and `tx_status` are set.
__u8	tx_status	The status bits of the transmitted message. See *CEC Transmit Status* for the possible status values. It is 0 if this message was received, not transmitted.
__u8	tx_arb_lost_cnt	A counter of the number of transmit attempts that resulted in the Arbitration Lost error. This is only set if the hardware supports this, otherwise it is always 0. This counter is only valid if the *CEC_TX_STATUS_ARB_LOST* status bit is set.
__u8	tx_nack_cnt	A counter of the number of transmit attempts that resulted in the Not Acknowledged error. This is only set if the hardware supports this, otherwise it is always 0. This counter is only valid if the *CEC_TX_STATUS_NACK* status bit is set.
__u8	tx_low_drive_cnt	A counter of the number of transmit attempts that resulted in the Arbitration Lost error. This is only set if the hardware supports this, otherwise it is always 0. This counter is only valid if the *CEC_TX_STATUS_LOW_DRIVE* status bit is set.

continues on next page

Table 271 – continued from previous page

__u8	tx_error_cnt	A counter of the number of transmit errors other than Arbitration Lost or Not Acknowledged. This is only set if the hardware supports this, otherwise it is always 0. This counter is only valid if the *CEC_TX_STATUS_ERROR* status bit is set.

Table 272: Flags for struct cec_msg

CEC_MSG_FL_REPLY_TO_FOLLOWE	1	If a CEC transmit expects a reply, then by default that reply is only sent to the filehandle that called *ioctl CEC_TRANSMIT*. If this flag is set, then the reply is also sent to all followers, if any. If the filehandle that called *ioctl CEC_TRANSMIT* is also a follower, then that filehandle will receive the reply twice: once as the result of the *ioctl CEC_TRANSMIT*, and once via *ioctl CEC_RECEIVE*.
CEC_MSG_FL_RAW	2	Normally CEC messages are validated before transmitting them. If this flag is set when *ioctl CEC_TRANSMIT* is called, then no validation takes place and the message is transmitted as-is. This is useful when debugging CEC issues. This flag is only allowed if the process has the `CAP_SYS_RAWIO` capability. If that is not set, then the `EPERM` error code is returned.

Table 273: CEC Transmit Status

CEC_TX_STATUS_OK	0x01	The message was transmitted successfully. This is mutually exclusive with *CEC_TX_STATUS_MAX_RETRIES*. Other bits can still be set if earlier attempts met with failure before the transmit was eventually successful.
CEC_TX_STATUS_ARB_LOST	0x02	CEC line arbitration was lost, i.e. another transmit started at the same time with a higher priority. Optional status, not all hardware can detect this error condition.
CEC_TX_STATUS_NACK	0x04	Message was not acknowledged. Note that some hardware cannot tell apart a 'Not Acknowledged' status from other error conditions, i.e. the result of a transmit is just OK or FAIL. In that case this status will be returned when the transmit failed.
CEC_TX_STATUS_LOW_DRIVE	0x08	Low drive was detected on the CEC bus. This indicates that a follower detected an error on the bus and requests a retransmission. Optional status, not all hardware can detect this error condition.
CEC_TX_STATUS_ERROR	0x10	Some error occurred. This is used for any errors that do not fit CEC_TX_STATUS_ARB_LOST or CEC_TX_STATUS_LOW_DRIVE, either because the hardware could not tell which error occurred, or because the hardware tested for other conditions besides those two. Optional status.
CEC_TX_STATUS_MAX_RETRIES	0x20	The transmit failed after one or more retries. This status bit is mutually exclusive with *CEC_TX_STATUS_OK*. Other bits can still be set to explain which failures were seen.
CEC_TX_STATUS_ABORTED	0x40	The transmit was aborted due to an HDMI disconnect, or the adapter was unconfigured, or a transmit was interrupted, or the driver returned an error when attempting to start a transmit.
CEC_TX_STATUS_TIMEOUT	0x80	The transmit timed out. This should not normally happen and this indicates a driver problem.

Table 274: CEC Receive Status

CEC_RX_STATUS_OK	0x01	The message was received successfully.
CEC_RX_STATUS_TIMEOUT	0x02	The reply to an earlier transmitted message timed out.
CEC_RX_STATUS_FEATURE_AB(	0x04	The message was received successfully but the reply was CEC_MSG_FEATURE_ABORT. This status is only set if this message was the reply to an earlier transmitted message.
CEC_RX_STATUS_ABORTED	0x08	The wait for a reply to an earlier transmitted message was aborted because the HDMI cable was disconnected, the adapter was unconfigured or the *CEC_TRANSMIT* that waited for a reply was interrupted.

Return Value

On success 0 is returned, on error -1 and the `errno` variable is set appropriately. The generic error codes are described at the *Generic Error Codes* chapter.

The *ioctl CEC_RECEIVE* can return the following error codes:

EAGAIN
: No messages are in the receive queue, and the filehandle is in non-blocking mode.

ETIMEDOUT
: The `timeout` was reached while waiting for a message.

ERESTARTSYS
: The wait for a message was interrupted (e.g. by Ctrl-C).

The *ioctl CEC_TRANSMIT* can return the following error codes:

ENOTTY
: The `CEC_CAP_TRANSMIT` capability wasn't set, so this ioctl is not supported.

EPERM
: The CEC adapter is not configured, i.e. *ioctl CEC_ADAP_S_LOG_ADDRS* has never been called, or `CEC_MSG_FL_RAW` was used from a process that did not have the `CAP_SYS_RAWIO` capability.

ENONET
: The CEC adapter is not configured, i.e. *ioctl CEC_ADAP_S_LOG_ADDRS* was called, but the physical address is invalid so no logical address was claimed. An exception is made in this case for transmits from initiator 0xf ('Unregistered') to destination 0 ('TV'). In that case the transmit will proceed as usual.

EBUSY
: Another filehandle is in exclusive follower or initiator mode, or the filehandle is in mode `CEC_MODE_NO_INITIATOR`. This is also returned if the transmit queue is full.

EINVAL
The contents of struct *cec_msg* is invalid.

ERESTARTSYS
The wait for a successful transmit was interrupted (e.g. by Ctrl-C).

8.6.3 CEC Pin Framework Error Injection

The CEC Pin Framework is a core CEC framework for CEC hardware that only has low-level support for the CEC bus. Most hardware today will have high-level CEC support where the hardware deals with driving the CEC bus, but some older devices aren' t that fancy. However, this framework also allows you to connect the CEC pin to a GPIO on e.g. a Raspberry Pi and you have now made a CEC adapter.

What makes doing this so interesting is that since we have full control over the bus it is easy to support error injection. This is ideal to test how well CEC adapters can handle error conditions.

Currently only the cec-gpio driver (when the CEC line is directly connected to a pull-up GPIO line) and the AllWinner A10/A20 drm driver support this framework.

If `CONFIG_CEC_PIN_ERROR_INJ` is enabled, then error injection is available through debugfs. Specifically, in `/sys/kernel/debug/cec/cecX/` there is now an `error-inj` file.

Note: The error injection commands are not a stable ABI and may change in the future.

With `cat error-inj` you can see both the possible commands and the current error injection status:

```
$ cat /sys/kernel/debug/cec/cec0/error-inj
# Clear error injections:
#   clear          clear all rx and tx error injections
#   rx-clear       clear all rx error injections
#   tx-clear       clear all tx error injections
#   <op> clear     clear all rx and tx error injections for <op>
#   <op> rx-clear  clear all rx error injections for <op>
#   <op> tx-clear  clear all tx error injections for <op>
#
# RX error injection:
#   <op>[,<mode>] rx-nack              NACK the message instead of␣
↪sending an ACK
#   <op>[,<mode>] rx-low-drive <bit>   force a low-drive condition␣
↪at this bit position
#   <op>[,<mode>] rx-add-byte          add a spurious byte to the␣
↪received CEC message
#   <op>[,<mode>] rx-remove-byte       remove the last byte from␣
↪the received CEC message
#   <op>[,<mode>] rx-arb-lost <poll>   generate a POLL message to␣
↪trigger an arbitration lost
```

(continues on next page)

(continued from previous page)

```
#
# TX error injection settings:
#   tx-ignore-nack-until-eom            ignore early NACKs until EOM
#   tx-custom-low-usecs <usecs>         define the 'low' time for
↪the custom pulse
#   tx-custom-high-usecs <usecs>        define the 'high' time for
↪the custom pulse
#   tx-custom-pulse                     transmit the custom pulse
↪once the bus is idle
#
# TX error injection:
#   <op>[,<mode>] tx-no-eom             don't set the EOM bit
#   <op>[,<mode>] tx-early-eom          set the EOM bit one byte too
↪soon
#   <op>[,<mode>] tx-add-bytes <num>    append <num> (1-255)
↪spurious bytes to the message
#   <op>[,<mode>] tx-remove-byte        drop the last byte from the
↪message
#   <op>[,<mode>] tx-short-bit <bit>    make this bit shorter than
↪allowed
#   <op>[,<mode>] tx-long-bit <bit>     make this bit longer than
↪allowed
#   <op>[,<mode>] tx-custom-bit <bit>   send the custom pulse
↪instead of this bit
#   <op>[,<mode>] tx-short-start        send a start pulse that's
↪too short
#   <op>[,<mode>] tx-long-start         send a start pulse that's
↪too long
#   <op>[,<mode>] tx-custom-start       send the custom pulse
↪instead of the start pulse
#   <op>[,<mode>] tx-last-bit <bit>     stop sending after this bit
#   <op>[,<mode>] tx-low-drive <bit>    force a low-drive condition
↪at this bit position
#
# <op>       CEC message opcode (0-255) or 'any'
# <mode>     'once' (default), 'always', 'toggle' or 'off'
# <bit>      CEC message bit (0-159)
#            10 bits per 'byte': bits 0-7: data, bit 8: EOM, bit 9:
↪ACK
# <poll>     CEC poll message used to test arbitration lost (0x00-
↪0xff, default 0x0f)
# <usecs>    microseconds (0-10000000, default 1000)

clear
```

You can write error injection commands to `error-inj` using `echo 'cmd' >error-inj` or `cat cmd.txt >error-inj`. The `cat error-inj` output contains the current error commands. You can save the output to a file and use it as an input to `error-inj` later.

Basic Syntax

Leading spaces/tabs are ignored. If the next character is a # or the end of the line was reached, then the whole line is ignored. Otherwise a command is expected.

The error injection commands fall in two main groups: those relating to receiving CEC messages and those relating to transmitting CEC messages. In addition, there are commands to clear existing error injection commands and to create custom pulses on the CEC bus.

Most error injection commands can be executed for specific CEC opcodes or for all opcodes (`any`). Each command also has a 'mode' which can be `off` (can be used to turn off an existing error injection command), `once` (the default) which will trigger the error injection only once for the next received or transmitted message, `always` to always trigger the error injection and `toggle` to toggle the error injection on or off for every transmit or receive.

So '`any rx-nack`' will NACK the next received CEC message, '`any,always rx-nack`' will NACK all received CEC messages and '`0x82,toggle rx-nack`' will only NACK if an Active Source message was received and do that only for every other received message.

After an error was injected with mode `once` the error injection command is cleared automatically, so `once` is a one-time deal.

All combinations of `<op>` and error injection commands can co-exist. So this is fine:

```
0x9e tx-add-bytes 1
0x9e tx-early-eom
0x9f tx-add-bytes 2
any rx-nack
```

All four error injection commands will be active simultaneously.

However, if the same `<op>` and command combination is specified, but with different arguments:

```
0x9e tx-add-bytes 1
0x9e tx-add-bytes 2
```

Then the second will overwrite the first.

Clear Error Injections

`clear`
: Clear all error injections.

`rx-clear`
: Clear all receive error injections

`tx-clear`
: Clear all transmit error injections

`<op> clear`
: Clear all error injections for the given opcode.

<op> rx-clear
Clear all receive error injections for the given opcode.

<op> tx-clear
Clear all transmit error injections for the given opcode.

Receive Messages

<op>[,<mode>] rx-nack
NACK broadcast messages and messages directed to this CEC adapter. Every byte of the message will be NACKed in case the transmitter keeps transmitting after the first byte was NACKed.

<op>[,<mode>] rx-low-drive <bit>
Force a Low Drive condition at this bit position. If <op> specifies a specific CEC opcode then the bit position must be at least 18, otherwise the opcode hasn't been received yet. This tests if the transmitter can handle the Low Drive condition correctly and reports the error correctly. Note that a Low Drive in the first 4 bits can also be interpreted as an Arbitration Lost condition by the transmitter. This is implementation dependent.

<op>[,<mode>] rx-add-byte
Add a spurious 0x55 byte to the received CEC message, provided the message was 15 bytes long or less. This is useful to test the high-level protocol since spurious bytes should be ignored.

<op>[,<mode>] rx-remove-byte
Remove the last byte from the received CEC message, provided it was at least 2 bytes long. This is useful to test the high-level protocol since messages that are too short should be ignored.

<op>[,<mode>] rx-arb-lost <poll>
Generate a POLL message to trigger an Arbitration Lost condition. This command is only allowed for `<op>` values of `next` or `all`. As soon as a start bit has been received the CEC adapter will switch to transmit mode and it will transmit a POLL message. By default this is 0x0f, but it can also be specified explicitly via the `<poll>` argument.

This command can be used to test the Arbitration Lost condition in the remote CEC transmitter. Arbitration happens when two CEC adapters start sending a message at the same time. In that case the initiator with the most leading zeroes wins and the other transmitter has to stop transmitting ('Arbitration Lost'). This is very hard to test, except by using this error injection command.

This does not work if the remote CEC transmitter has logical address 0 ('TV') since that will always win.

Transmit Messages

tx-ignore-nack-until-eom
This setting changes the behavior of transmitting CEC messages. Normally as soon as the receiver NACKs a byte the transmit will stop, but the specification also allows that the full message is transmitted and only at the end will the transmitter look at the ACK bit. This is not recommended behavior since there is no point in keeping the CEC bus busy for longer than is strictly needed. Especially given how slow the bus is.

This setting can be used to test how well a receiver deals with transmitters that ignore NACKs until the very end of the message.

<op>[,<mode>] tx-no-eom
Don't set the EOM bit. Normally the last byte of the message has the EOM (End-Of-Message) bit set. With this command the transmit will just stop without ever sending an EOM. This can be used to test how a receiver handles this case. Normally receivers have a time-out after which they will go back to the Idle state.

<op>[,<mode>] tx-early-eom
Set the EOM bit one byte too soon. This obviously only works for messages of two bytes or more. The EOM bit will be set for the second-to-last byte and not for the final byte. The receiver should ignore the last byte in this case. Since the resulting message is likely to be too short for this same reason the whole message is typically ignored. The receiver should be in Idle state after the last byte was transmitted.

<op>[,<mode>] tx-add-bytes <num>
Append <num> (1-255) spurious bytes to the message. The extra bytes have the value of the byte position in the message. So if you transmit a two byte message (e.g. a Get CEC Version message) and add 2 bytes, then the full message received by the remote CEC adapter is `0x40 0x9f 0x02 0x03`.

This command can be used to test buffer overflows in the receiver. E.g. what does it do when it receives more than the maximum message size of 16 bytes.

<op>[,<mode>] tx-remove-byte
Drop the last byte from the message, provided the message is at least two bytes long. The receiver should ignore messages that are too short.

<op>[,<mode>] tx-short-bit <bit>
Make this bit period shorter than allowed. The bit position cannot be an Ack bit. If <op> specifies a specific CEC opcode then the bit position must be at least 18, otherwise the opcode hasn't been received yet. Normally the period of a data bit is between 2.05 and 2.75 milliseconds. With this command the period of this bit is 1.8 milliseconds, this is done by reducing the time the CEC bus is high. This bit period is less than is allowed and the receiver should respond with a Low Drive condition.

This command is ignored for 0 bits in bit positions 0 to 3. This is because the receiver also looks for an Arbitration Lost condition in those first four bits and it is undefined what will happen if it sees a too-short 0 bit.

<op>[,<mode>] tx-long-bit <bit>
Make this bit period longer than is valid. The bit position cannot be an Ack

bit. If <op> specifies a specific CEC opcode then the bit position must be at least 18, otherwise the opcode hasn't been received yet. Normally the period of a data bit is between 2.05 and 2.75 milliseconds. With this command the period of this bit is 2.9 milliseconds, this is done by increasing the time the CEC bus is high.

Even though this bit period is longer than is valid it is undefined what a receiver will do. It might just accept it, or it might time out and return to Idle state. Unfortunately the CEC specification is silent about this.

This command is ignored for 0 bits in bit positions 0 to 3. This is because the receiver also looks for an Arbitration Lost condition in those first four bits and it is undefined what will happen if it sees a too-long 0 bit.

`<op>[,<mode>] tx-short-start`
Make this start bit period shorter than allowed. Normally the period of a start bit is between 4.3 and 4.7 milliseconds. With this command the period of the start bit is 4.1 milliseconds, this is done by reducing the time the CEC bus is high. This start bit period is less than is allowed and the receiver should return to Idle state when this is detected.

`<op>[,<mode>] tx-long-start`
Make this start bit period longer than is valid. Normally the period of a start bit is between 4.3 and 4.7 milliseconds. With this command the period of the start bit is 5 milliseconds, this is done by increasing the time the CEC bus is high. This start bit period is more than is valid and the receiver should return to Idle state when this is detected.

Even though this start bit period is longer than is valid it is undefined what a receiver will do. It might just accept it, or it might time out and return to Idle state. Unfortunately the CEC specification is silent about this.

`<op>[,<mode>] tx-last-bit <bit>`
Just stop transmitting after this bit. If <op> specifies a specific CEC opcode then the bit position must be at least 18, otherwise the opcode hasn't been received yet. This command can be used to test how the receiver reacts when a message just suddenly stops. It should time out and go back to Idle state.

`<op>[,<mode>] tx-low-drive <bit>`
Force a Low Drive condition at this bit position. If <op> specifies a specific CEC opcode then the bit position must be at least 18, otherwise the opcode hasn't been received yet. This can be used to test how the receiver handles Low Drive conditions. Note that if this happens at bit positions 0-3 the receiver can interpret this as an Arbitration Lost condition. This is implementation dependent.

Custom Pulses

tx-custom-low-usecs <usecs>
This defines the duration in microseconds that the custom pulse pulls the CEC line low. The default is 1000 microseconds.

tx-custom-high-usecs <usecs>
This defines the duration in microseconds that the custom pulse keeps the CEC line high (unless another CEC adapter pulls it low in that time). The default is 1000 microseconds. The total period of the custom pulse is `tx-custom-low-usecs + tx-custom-high-usecs`.

<op>[,<mode>] tx-custom-bit <bit>
Send the custom bit instead of a regular data bit. The bit position cannot be an Ack bit. If <op> specifies a specific CEC opcode then the bit position must be at least 18, otherwise the opcode hasn't been received yet.

<op>[,<mode>] tx-custom-start
Send the custom bit instead of a regular start bit.

tx-custom-pulse
Transmit a single custom pulse as soon as the CEC bus is idle.

8.6.4 CEC Header File

cec.h

```
/* SPDX-License-Identifier: ((GPL-2.0 WITH Linux-syscall-note) OR␣
↪BSD-3-Clause) */
/*
 * cec - HDMI Consumer Electronics Control public header
 *
 * Copyright 2016 Cisco Systems, Inc. and/or its affiliates. All␣
↪rights reserved.
 */

#ifndef _CEC_UAPI_H
#define _CEC_UAPI_H

#include <linux/types.h>
#include <linux/string.h>

#define CEC_MAX_MSG_SIZE        16

/**
 * struct cec_msg - CEC message structure.
 * @tx_ts:      Timestamp in nanoseconds using CLOCK_MONOTONIC. Set␣
↪by the
 *              driver when the message transmission has finished.
 * @rx_ts:      Timestamp in nanoseconds using CLOCK_MONOTONIC. Set␣
↪by the
 *              driver when the message was received.
```

```
* @len:         Length in bytes of the message.
* @timeout:     The timeout (in ms) that is used to timeout CEC_
↪RECEIVE.
*               Set to 0 if you want to wait forever. This timeout␣
↪can also be
*               used with CEC_TRANSMIT as the timeout for waiting␣
↪for a reply.
*               If 0, then it will use a 1 second timeout instead␣
↪of waiting
*               forever as is done with CEC_RECEIVE.
* @sequence:    The framework assigns a sequence number to messages␣
↪that are
*               sent. This can be used to track replies to␣
↪previously sent
*               messages.
* @flags:       Set to 0.
* @msg:         The message payload.
* @reply:       This field is ignored with CEC_RECEIVE and is only␣
↪used by
*               CEC_TRANSMIT. If non-zero, then wait for a reply␣
↪with this
*               opcode. Set to CEC_MSG_FEATURE_ABORT if you want to␣
↪wait for
*               a possible ABORT reply. If there was an error when␣
↪sending the
*               msg or FeatureAbort was returned, then reply is set␣
↪to 0.
*               If reply is non-zero upon return, then len/msg are␣
↪set to
*               the received message.
*               If reply is zero upon return and status has the
*               CEC_TX_STATUS_FEATURE_ABORT bit set, then len/msg␣
↪are set to
*               the received feature abort message.
*               If reply is zero upon return and status has the
*               CEC_TX_STATUS_MAX_RETRIES bit set, then no reply␣
↪was seen at
*               all. If reply is non-zero for CEC_TRANSMIT and the␣
↪message is a
*               broadcast, then -EINVAL is returned.
*               if reply is non-zero, then timeout is set to 1000␣
↪(the required
*               maximum response time).
* @rx_status:   The message receive status bits. Set by the driver.
* @tx_status:   The message transmit status bits. Set by the driver.
* @tx_arb_lost_cnt: The number of 'Arbitration Lost' events. Set␣
↪by the driver.
* @tx_nack_cnt: The number of 'Not Acknowledged' events. Set by␣
↪the driver.
* @tx_low_drive_cnt: The number of 'Low Drive Detected' events.␣
↪Set by the the
```

```
 *              driver.
 * @tx_error_cnt: The number of 'Error' events. Set by the driver.
 */
struct cec_msg {
        __u64 tx_ts;
        __u64 rx_ts;
        __u32 len;
        __u32 timeout;
        __u32 sequence;
        __u32 flags;
        __u8 msg[CEC_MAX_MSG_SIZE];
        __u8 reply;
        __u8 rx_status;
        __u8 tx_status;
        __u8 tx_arb_lost_cnt;
        __u8 tx_nack_cnt;
        __u8 tx_low_drive_cnt;
        __u8 tx_error_cnt;
};

/**
 * cec_msg_initiator - return the initiator's logical address.
 * @msg:        the message structure
 */
static inline __u8 cec_msg_initiator(const struct cec_msg *msg)
{
        return msg->msg[0] >> 4;
}

/**
 * cec_msg_destination - return the destination's logical address.
 * @msg:        the message structure
 */
static inline __u8 cec_msg_destination(const struct cec_msg *msg)
{
        return msg->msg[0] & 0xf;
}

/**
 * cec_msg_opcode - return the opcode of the message, -1 for poll
 * @msg:        the message structure
 */
static inline int cec_msg_opcode(const struct cec_msg *msg)
{
        return msg->len > 1 ? msg->msg[1] : -1;
}

/**
 * cec_msg_is_broadcast - return true if this is a broadcast␣
↪message.
 * @msg:        the message structure
```

```
 */
static inline int cec_msg_is_broadcast(const struct cec_msg *msg)
{
        return (msg->msg[0] & 0xf) == 0xf;
}

/**
 * cec_msg_init - initialize the message structure.
 * @msg:        the message structure
 * @initiator:  the logical address of the initiator
 * @destination:the logical address of the destination (0xf for␣
↪broadcast)
 *
 * The whole structure is zeroed, the len field is set to 1 (i.e. a␣
↪poll
 * message) and the initiator and destination are filled in.
 */
static inline void cec_msg_init(struct cec_msg *msg,
                                __u8 initiator, __u8 destination)
{
        memset(msg, 0, sizeof(*msg));
        msg->msg[0] = (initiator << 4) | destination;
        msg->len = 1;
}

/**
 * cec_msg_set_reply_to - fill in destination/initiator in a reply␣
↪message.
 * @msg:        the message structure for the reply
 * @orig:       the original message structure
 *
 * Set the msg destination to the orig initiator and the msg␣
↪initiator to the
 * orig destination. Note that msg and orig may be the same pointer,
↪ in which
 * case the change is done in place.
 */
static inline void cec_msg_set_reply_to(struct cec_msg *msg,
                                        struct cec_msg *orig)
{
        /* The destination becomes the initiator and vice versa */
        msg->msg[0] = (cec_msg_destination(orig) << 4) |
                      cec_msg_initiator(orig);
        msg->reply = msg->timeout = 0;
}

/* cec_msg flags field */
#define CEC_MSG_FL_REPLY_TO_FOLLOWERS   (1 << 0)
#define CEC_MSG_FL_RAW                  (1 << 1)

/* cec_msg tx/rx_status field */
```

```
#define CEC_TX_STATUS_OK                (1 << 0)
#define CEC_TX_STATUS_ARB_LOST          (1 << 1)
#define CEC_TX_STATUS_NACK              (1 << 2)
#define CEC_TX_STATUS_LOW_DRIVE         (1 << 3)
#define CEC_TX_STATUS_ERROR             (1 << 4)
#define CEC_TX_STATUS_MAX_RETRIES       (1 << 5)
#define CEC_TX_STATUS_ABORTED           (1 << 6)
#define CEC_TX_STATUS_TIMEOUT           (1 << 7)

#define CEC_RX_STATUS_OK                (1 << 0)
#define CEC_RX_STATUS_TIMEOUT           (1 << 1)
#define CEC_RX_STATUS_FEATURE_ABORT     (1 << 2)
#define CEC_RX_STATUS_ABORTED           (1 << 3)

static inline int cec_msg_status_is_ok(const struct cec_msg *msg)
{
        if (msg->tx_status && !(msg->tx_status & CEC_TX_STATUS_OK))
                return 0;
        if (msg->rx_status && !(msg->rx_status & CEC_RX_STATUS_OK))
                return 0;
        if (!msg->tx_status && !msg->rx_status)
                return 0;
        return !(msg->rx_status & CEC_RX_STATUS_FEATURE_ABORT);
}

#define CEC_LOG_ADDR_INVALID            0xff
#define CEC_PHYS_ADDR_INVALID           0xffff

/*
 * The maximum number of logical addresses one device can be␣
↪assigned to.
 * The CEC 2.0 spec allows for only 2 logical addresses at the␣
↪moment. The
 * Analog Devices CEC hardware supports 3. So let's go wild and go␣
↪for 4.
 */
#define CEC_MAX_LOG_ADDRS 4

/* The logical addresses defined by CEC 2.0 */
#define CEC_LOG_ADDR_TV                 0
#define CEC_LOG_ADDR_RECORD_1           1
#define CEC_LOG_ADDR_RECORD_2           2
#define CEC_LOG_ADDR_TUNER_1            3
#define CEC_LOG_ADDR_PLAYBACK_1         4
#define CEC_LOG_ADDR_AUDIOSYSTEM        5
#define CEC_LOG_ADDR_TUNER_2            6
#define CEC_LOG_ADDR_TUNER_3            7
#define CEC_LOG_ADDR_PLAYBACK_2         8
#define CEC_LOG_ADDR_RECORD_3           9
#define CEC_LOG_ADDR_TUNER_4            10
#define CEC_LOG_ADDR_PLAYBACK_3         11
```

```
#define CEC_LOG_ADDR_BACKUP_1            12
#define CEC_LOG_ADDR_BACKUP_2            13
#define CEC_LOG_ADDR_SPECIFIC            14
#define CEC_LOG_ADDR_UNREGISTERED        15 /* as initiator address␣
 ↪*/
#define CEC_LOG_ADDR_BROADCAST           15 /* as destination␣
 ↪address */

/* The logical address types that the CEC device wants to claim */
#define CEC_LOG_ADDR_TYPE_TV             0
#define CEC_LOG_ADDR_TYPE_RECORD         1
#define CEC_LOG_ADDR_TYPE_TUNER          2
#define CEC_LOG_ADDR_TYPE_PLAYBACK       3
#define CEC_LOG_ADDR_TYPE_AUDIOSYSTEM    4
#define CEC_LOG_ADDR_TYPE_SPECIFIC       5
#define CEC_LOG_ADDR_TYPE_UNREGISTERED   6
/*
 * Switches should use UNREGISTERED.
 * Processors should use SPECIFIC.
 */

#define CEC_LOG_ADDR_MASK_TV             (1 << CEC_LOG_ADDR_TV)
#define CEC_LOG_ADDR_MASK_RECORD         ((1 << CEC_LOG_ADDR_RECORD_
 ↪1) | \
                                          (1 << CEC_LOG_ADDR_RECORD_
 ↪2) | \
                                          (1 << CEC_LOG_ADDR_RECORD_
 ↪3))
#define CEC_LOG_ADDR_MASK_TUNER          ((1 << CEC_LOG_ADDR_TUNER_
 ↪1) | \
                                          (1 << CEC_LOG_ADDR_TUNER_
 ↪2) | \
                                          (1 << CEC_LOG_ADDR_TUNER_
 ↪3) | \
                                          (1 << CEC_LOG_ADDR_TUNER_
 ↪4))
#define CEC_LOG_ADDR_MASK_PLAYBACK       ((1 << CEC_LOG_ADDR_
 ↪PLAYBACK_1) | \
                                          (1 << CEC_LOG_ADDR_
 ↪PLAYBACK_2) | \
                                          (1 << CEC_LOG_ADDR_
 ↪PLAYBACK_3))
#define CEC_LOG_ADDR_MASK_AUDIOSYSTEM    (1 << CEC_LOG_ADDR_
 ↪AUDIOSYSTEM)
#define CEC_LOG_ADDR_MASK_BACKUP         ((1 << CEC_LOG_ADDR_BACKUP_
 ↪1) | \
                                          (1 << CEC_LOG_ADDR_BACKUP_
 ↪2))
#define CEC_LOG_ADDR_MASK_SPECIFIC       (1 << CEC_LOG_ADDR_SPECIFIC)
#define CEC_LOG_ADDR_MASK_UNREGISTERED   (1 << CEC_LOG_ADDR_
 ↪UNREGISTERED)
```

```
static inline int cec_has_tv(__u16 log_addr_mask)
{
        return log_addr_mask & CEC_LOG_ADDR_MASK_TV;
}

static inline int cec_has_record(__u16 log_addr_mask)
{
        return log_addr_mask & CEC_LOG_ADDR_MASK_RECORD;
}

static inline int cec_has_tuner(__u16 log_addr_mask)
{
        return log_addr_mask & CEC_LOG_ADDR_MASK_TUNER;
}

static inline int cec_has_playback(__u16 log_addr_mask)
{
        return log_addr_mask & CEC_LOG_ADDR_MASK_PLAYBACK;
}

static inline int cec_has_audiosystem(__u16 log_addr_mask)
{
        return log_addr_mask & CEC_LOG_ADDR_MASK_AUDIOSYSTEM;
}

static inline int cec_has_backup(__u16 log_addr_mask)
{
        return log_addr_mask & CEC_LOG_ADDR_MASK_BACKUP;
}

static inline int cec_has_specific(__u16 log_addr_mask)
{
        return log_addr_mask & CEC_LOG_ADDR_MASK_SPECIFIC;
}

static inline int cec_is_unregistered(__u16 log_addr_mask)
{
        return log_addr_mask & CEC_LOG_ADDR_MASK_UNREGISTERED;
}

static inline int cec_is_unconfigured(__u16 log_addr_mask)
{
        return log_addr_mask == 0;
}

/*
 * Use this if there is no vendor ID (CEC_G_VENDOR_ID) or if the␣
↪vendor ID
 * should be disabled (CEC_S_VENDOR_ID)
 */
```

```
#define CEC_VENDOR_ID_NONE              0xffffffff

/* The message handling modes */
/* Modes for initiator */
#define CEC_MODE_NO_INITIATOR           (0x0 << 0)
#define CEC_MODE_INITIATOR              (0x1 << 0)
#define CEC_MODE_EXCL_INITIATOR         (0x2 << 0)
#define CEC_MODE_INITIATOR_MSK          0x0f

/* Modes for follower */
#define CEC_MODE_NO_FOLLOWER            (0x0 << 4)
#define CEC_MODE_FOLLOWER               (0x1 << 4)
#define CEC_MODE_EXCL_FOLLOWER          (0x2 << 4)
#define CEC_MODE_EXCL_FOLLOWER_PASSTHRU (0x3 << 4)
#define CEC_MODE_MONITOR_PIN            (0xd << 4)
#define CEC_MODE_MONITOR                (0xe << 4)
#define CEC_MODE_MONITOR_ALL            (0xf << 4)
#define CEC_MODE_FOLLOWER_MSK           0xf0

/* Userspace has to configure the physical address */
#define CEC_CAP_PHYS_ADDR       (1 << 0)
/* Userspace has to configure the logical addresses */
#define CEC_CAP_LOG_ADDRS       (1 << 1)
/* Userspace can transmit messages (and thus become follower as␣
↪well) */
#define CEC_CAP_TRANSMIT        (1 << 2)
/*
 * Passthrough all messages instead of processing them.
 */
#define CEC_CAP_PASSTHROUGH     (1 << 3)
/* Supports remote control */
#define CEC_CAP_RC              (1 << 4)
/* Hardware can monitor all messages, not just directed and␣
↪broadcast. */
#define CEC_CAP_MONITOR_ALL     (1 << 5)
/* Hardware can use CEC only if the HDMI HPD pin is high. */
#define CEC_CAP_NEEDS_HPD       (1 << 6)
/* Hardware can monitor CEC pin transitions */
#define CEC_CAP_MONITOR_PIN     (1 << 7)
/* CEC_ADAP_G_CONNECTOR_INFO is available */
#define CEC_CAP_CONNECTOR_INFO  (1 << 8)

/**
 * struct cec_caps - CEC capabilities structure.
 * @driver: name of the CEC device driver.
 * @name: name of the CEC device. @driver + @name must be unique.
 * @available_log_addrs: number of available logical addresses.
 * @capabilities: capabilities of the CEC adapter.
 * @version: version of the CEC adapter framework.
 */
struct cec_caps {
```

```
        char driver[32];
        char name[32];
        __u32 available_log_addrs;
        __u32 capabilities;
        __u32 version;
};

/**
 * struct cec_log_addrs - CEC logical addresses structure.
 * @log_addr: the claimed logical addresses. Set by the driver.
 * @log_addr_mask: current logical address mask. Set by the driver.
 * @cec_version: the CEC version that the adapter should implement.␣
↪Set by the
 *      caller.
 * @num_log_addrs: how many logical addresses should be claimed.␣
↪Set by the
 *      caller.
 * @vendor_id: the vendor ID of the device. Set by the caller.
 * @flags: flags.
 * @osd_name: the OSD name of the device. Set by the caller.
 * @primary_device_type: the primary device type for each logical␣
↪address.
 *      Set by the caller.
 * @log_addr_type: the logical address types. Set by the caller.
 * @all_device_types: CEC 2.0: all device types represented by the␣
↪logical
 *      address. Set by the caller.
 * @features:   CEC 2.0: The logical address features. Set by the␣
↪caller.
 */
struct cec_log_addrs {
        __u8 log_addr[CEC_MAX_LOG_ADDRS];
        __u16 log_addr_mask;
        __u8 cec_version;
        __u8 num_log_addrs;
        __u32 vendor_id;
        __u32 flags;
        char osd_name[15];
        __u8 primary_device_type[CEC_MAX_LOG_ADDRS];
        __u8 log_addr_type[CEC_MAX_LOG_ADDRS];

        /* CEC 2.0 */
        __u8 all_device_types[CEC_MAX_LOG_ADDRS];
        __u8 features[CEC_MAX_LOG_ADDRS][12];
};

/* Allow a fallback to unregistered */
#define CEC_LOG_ADDRS_FL_ALLOW_UNREG_FALLBACK   (1 << 0)
/* Passthrough RC messages to the input subsystem */
#define CEC_LOG_ADDRS_FL_ALLOW_RC_PASSTHRU      (1 << 1)
/* CDC-Only device: supports only CDC messages */
```

```
#define CEC_LOG_ADDRS_FL_CDC_ONLY               (1 << 2)

/**
 * struct cec_drm_connector_info - tells which drm connector is
 * associated with the CEC adapter.
 * @card_no: drm card number
 * @connector_id: drm connector ID
 */
struct cec_drm_connector_info {
        __u32 card_no;
        __u32 connector_id;
};

#define CEC_CONNECTOR_TYPE_NO_CONNECTOR 0
#define CEC_CONNECTOR_TYPE_DRM          1

/**
 * struct cec_connector_info - tells if and which connector is
 * associated with the CEC adapter.
 * @type: connector type (if any)
 * @drm: drm connector info
 */
struct cec_connector_info {
        __u32 type;
        union {
                struct cec_drm_connector_info drm;
                __u32 raw[16];
        };
};

/* Events */

/* Event that occurs when the adapter state changes */
#define CEC_EVENT_STATE_CHANGE          1
/*
 * This event is sent when messages are lost because the application
 * didn't empty the message queue in time
 */
#define CEC_EVENT_LOST_MSGS             2
#define CEC_EVENT_PIN_CEC_LOW           3
#define CEC_EVENT_PIN_CEC_HIGH          4
#define CEC_EVENT_PIN_HPD_LOW           5
#define CEC_EVENT_PIN_HPD_HIGH          6
#define CEC_EVENT_PIN_5V_LOW            7
#define CEC_EVENT_PIN_5V_HIGH           8

#define CEC_EVENT_FL_INITIAL_STATE      (1 << 0)
#define CEC_EVENT_FL_DROPPED_EVENTS     (1 << 1)

/**
 * struct cec_event_state_change - used when the CEC adapter␣
```

```
↪changes state.
 * @phys_addr: the current physical address
 * @log_addr_mask: the current logical address mask
 * @have_conn_info: if non-zero, then HDMI connector information is␣
↪available.
 *      This field is only valid if CEC_CAP_CONNECTOR_INFO is set.␣
↪If that
 *      capability is set and @have_conn_info is zero, then that␣
↪indicates
 *      that the HDMI connector device is not instantiated, either␣
↪because
 *      the HDMI driver is still configuring the device or because␣
↪the HDMI
 *      device was unbound.
 */
struct cec_event_state_change {
        __u16 phys_addr;
        __u16 log_addr_mask;
        __u16 have_conn_info;
};

/**
 * struct cec_event_lost_msgs - tells you how many messages were␣
↪lost.
 * @lost_msgs: how many messages were lost.
 */
struct cec_event_lost_msgs {
        __u32 lost_msgs;
};

/**
 * struct cec_event - CEC event structure
 * @ts: the timestamp of when the event was sent.
 * @event: the event.
 * array.
 * @state_change: the event payload for CEC_EVENT_STATE_CHANGE.
 * @lost_msgs: the event payload for CEC_EVENT_LOST_MSGS.
 * @raw: array to pad the union.
 */
struct cec_event {
        __u64 ts;
        __u32 event;
        __u32 flags;
        union {
                struct cec_event_state_change state_change;
                struct cec_event_lost_msgs lost_msgs;
                __u32 raw[16];
        };
};

/* ioctls */
```

```
/* Adapter capabilities */
#define CEC_ADAP_G_CAPS         _IOWR('a',  0, struct cec_caps)

/*
 * phys_addr is either 0 (if this is the CEC root device)
 * or a valid physical address obtained from the sink's EDID
 * as read by this CEC device (if this is a source device)
 * or a physical address obtained and modified from a sink
 * EDID and used for a sink CEC device.
 * If nothing is connected, then phys_addr is 0xffff.
 * See HDMI 1.4b, section 8.7 (Physical Address).
 *
 * The CEC_ADAP_S_PHYS_ADDR ioctl may not be available if that is␣
↪handled
 * internally.
 */
#define CEC_ADAP_G_PHYS_ADDR    _IOR('a',  1, __u16)
#define CEC_ADAP_S_PHYS_ADDR    _IOW('a',  2, __u16)

/*
 * Configure the CEC adapter. It sets the device type and which
 * logical types it will try to claim. It will return which
 * logical addresses it could actually claim.
 * An error is returned if the adapter is disabled or if there
 * is no physical address assigned.
 */

#define CEC_ADAP_G_LOG_ADDRS    _IOR('a',  3, struct cec_log_addrs)
#define CEC_ADAP_S_LOG_ADDRS    _IOWR('a',  4, struct cec_log_addrs)

/* Transmit/receive a CEC command */
#define CEC_TRANSMIT            _IOWR('a',  5, struct cec_msg)
#define CEC_RECEIVE             _IOWR('a',  6, struct cec_msg)

/* Dequeue CEC events */
#define CEC_DQEVENT             _IOWR('a',  7, struct cec_event)

/*
 * Get and set the message handling mode for this filehandle.
 */
#define CEC_G_MODE              _IOR('a',  8, __u32)
#define CEC_S_MODE              _IOW('a',  9, __u32)

/* Get the connector info */
#define CEC_ADAP_G_CONNECTOR_INFO _IOR('a',  10, struct cec_
↪connector_info)

/*
 * The remainder of this header defines all CEC messages and␣
↪operands.
```

```
 * The format matters since it the cec-ctl utility parses it to␣
↪generate
 * code for implementing all these messages.
 *
 * Comments ending with 'Feature' group messages for each feature.
 * If messages are part of multiple features, then the "Has also"
 * comment is used to list the previously defined messages that are
 * supported by the feature.
 *
 * Before operands are defined a comment is added that gives the
 * name of the operand and in brackets the variable name of the
 * corresponding argument in the cec-funcs.h function.
 */

/* Messages */

/* One Touch Play Feature */
#define CEC_MSG_ACTIVE_SOURCE                           0x82
#define CEC_MSG_IMAGE_VIEW_ON                           0x04
#define CEC_MSG_TEXT_VIEW_ON                            0x0d

/* Routing Control Feature */

/*
 * Has also:
 *      CEC_MSG_ACTIVE_SOURCE
 */

#define CEC_MSG_INACTIVE_SOURCE                         0x9d
#define CEC_MSG_REQUEST_ACTIVE_SOURCE                   0x85
#define CEC_MSG_ROUTING_CHANGE                          0x80
#define CEC_MSG_ROUTING_INFORMATION                     0x81
#define CEC_MSG_SET_STREAM_PATH                         0x86

/* Standby Feature */
#define CEC_MSG_STANDBY                                 0x36

/* One Touch Record Feature */
#define CEC_MSG_RECORD_OFF                              0x0b
#define CEC_MSG_RECORD_ON                               0x09
/* Record Source Type Operand (rec_src_type) */
#define CEC_OP_RECORD_SRC_OWN                           1
#define CEC_OP_RECORD_SRC_DIGITAL                       2
#define CEC_OP_RECORD_SRC_ANALOG                        3
#define CEC_OP_RECORD_SRC_EXT_PLUG                      4
#define CEC_OP_RECORD_SRC_EXT_PHYS_ADDR                 5
/* Service Identification Method Operand (service_id_method) */
#define CEC_OP_SERVICE_ID_METHOD_BY_DIG_ID              0
#define CEC_OP_SERVICE_ID_METHOD_BY_CHANNEL             1
/* Digital Service Broadcast System Operand (dig_bcast_system) */
#define CEC_OP_DIG_SERVICE_BCAST_SYSTEM_ARIB_GEN        0x00
```

```
#define CEC_OP_DIG_SERVICE_BCAST_SYSTEM_ATSC_GEN        0x01
#define CEC_OP_DIG_SERVICE_BCAST_SYSTEM_DVB_GEN         0x02
#define CEC_OP_DIG_SERVICE_BCAST_SYSTEM_ARIB_BS         0x08
#define CEC_OP_DIG_SERVICE_BCAST_SYSTEM_ARIB_CS         0x09
#define CEC_OP_DIG_SERVICE_BCAST_SYSTEM_ARIB_T          0x0a
#define CEC_OP_DIG_SERVICE_BCAST_SYSTEM_ATSC_CABLE      0x10
#define CEC_OP_DIG_SERVICE_BCAST_SYSTEM_ATSC_SAT        0x11
#define CEC_OP_DIG_SERVICE_BCAST_SYSTEM_ATSC_T          0x12
#define CEC_OP_DIG_SERVICE_BCAST_SYSTEM_DVB_C           0x18
#define CEC_OP_DIG_SERVICE_BCAST_SYSTEM_DVB_S           0x19
#define CEC_OP_DIG_SERVICE_BCAST_SYSTEM_DVB_S2          0x1a
#define CEC_OP_DIG_SERVICE_BCAST_SYSTEM_DVB_T           0x1b
/* Analogue Broadcast Type Operand (ana_bcast_type) */
#define CEC_OP_ANA_BCAST_TYPE_CABLE                     0
#define CEC_OP_ANA_BCAST_TYPE_SATELLITE                 1
#define CEC_OP_ANA_BCAST_TYPE_TERRESTRIAL               2
/* Broadcast System Operand (bcast_system) */
#define CEC_OP_BCAST_SYSTEM_PAL_BG                      0x00
#define CEC_OP_BCAST_SYSTEM_SECAM_LQ                    0x01 /*␣
↪SECAM L' */
#define CEC_OP_BCAST_SYSTEM_PAL_M                       0x02
#define CEC_OP_BCAST_SYSTEM_NTSC_M                      0x03
#define CEC_OP_BCAST_SYSTEM_PAL_I                       0x04
#define CEC_OP_BCAST_SYSTEM_SECAM_DK                    0x05
#define CEC_OP_BCAST_SYSTEM_SECAM_BG                    0x06
#define CEC_OP_BCAST_SYSTEM_SECAM_L                     0x07
#define CEC_OP_BCAST_SYSTEM_PAL_DK                      0x08
#define CEC_OP_BCAST_SYSTEM_OTHER                       0x1f
/* Channel Number Format Operand (channel_number_fmt) */
#define CEC_OP_CHANNEL_NUMBER_FMT_1_PART                0x01
#define CEC_OP_CHANNEL_NUMBER_FMT_2_PART                0x02

#define CEC_MSG_RECORD_STATUS                           0x0a
/* Record Status Operand (rec_status) */
#define CEC_OP_RECORD_STATUS_CUR_SRC                    0x01
#define CEC_OP_RECORD_STATUS_DIG_SERVICE                0x02
#define CEC_OP_RECORD_STATUS_ANA_SERVICE                0x03
#define CEC_OP_RECORD_STATUS_EXT_INPUT                  0x04
#define CEC_OP_RECORD_STATUS_NO_DIG_SERVICE             0x05
#define CEC_OP_RECORD_STATUS_NO_ANA_SERVICE             0x06
#define CEC_OP_RECORD_STATUS_NO_SERVICE                 0x07
#define CEC_OP_RECORD_STATUS_INVALID_EXT_PLUG           0x09
#define CEC_OP_RECORD_STATUS_INVALID_EXT_PHYS_ADDR      0x0a
#define CEC_OP_RECORD_STATUS_UNSUP_CA                   0x0b
#define CEC_OP_RECORD_STATUS_NO_CA_ENTITLEMENTS         0x0c
#define CEC_OP_RECORD_STATUS_CANT_COPY_SRC              0x0d
#define CEC_OP_RECORD_STATUS_NO_MORE_COPIES             0x0e
#define CEC_OP_RECORD_STATUS_NO_MEDIA                   0x10
#define CEC_OP_RECORD_STATUS_PLAYING                    0x11
#define CEC_OP_RECORD_STATUS_ALREADY_RECORDING          0x12
#define CEC_OP_RECORD_STATUS_MEDIA_PROT                 0x13
```

```
#define CEC_OP_RECORD_STATUS_NO_SIGNAL                  0x14
#define CEC_OP_RECORD_STATUS_MEDIA_PROBLEM              0x15
#define CEC_OP_RECORD_STATUS_NO_SPACE                   0x16
#define CEC_OP_RECORD_STATUS_PARENTAL_LOCK              0x17
#define CEC_OP_RECORD_STATUS_TERMINATED_OK              0x1a
#define CEC_OP_RECORD_STATUS_ALREADY_TERM               0x1b
#define CEC_OP_RECORD_STATUS_OTHER                      0x1f

#define CEC_MSG_RECORD_TV_SCREEN                        0x0f

/* Timer Programming Feature */
#define CEC_MSG_CLEAR_ANALOGUE_TIMER                    0x33
/* Recording Sequence Operand (recording_seq) */
#define CEC_OP_REC_SEQ_SUNDAY                           0x01
#define CEC_OP_REC_SEQ_MONDAY                           0x02
#define CEC_OP_REC_SEQ_TUESDAY                          0x04
#define CEC_OP_REC_SEQ_WEDNESDAY                        0x08
#define CEC_OP_REC_SEQ_THURSDAY                         0x10
#define CEC_OP_REC_SEQ_FRIDAY                           0x20
#define CEC_OP_REC_SEQ_SATERDAY                         0x40
#define CEC_OP_REC_SEQ_ONCE_ONLY                        0x00

#define CEC_MSG_CLEAR_DIGITAL_TIMER                     0x99

#define CEC_MSG_CLEAR_EXT_TIMER                         0xa1
/* External Source Specifier Operand (ext_src_spec) */
#define CEC_OP_EXT_SRC_PLUG                             0x04
#define CEC_OP_EXT_SRC_PHYS_ADDR                        0x05

#define CEC_MSG_SET_ANALOGUE_TIMER                      0x34
#define CEC_MSG_SET_DIGITAL_TIMER                       0x97
#define CEC_MSG_SET_EXT_TIMER                           0xa2

#define CEC_MSG_SET_TIMER_PROGRAM_TITLE                 0x67
#define CEC_MSG_TIMER_CLEARED_STATUS                    0x43
/* Timer Cleared Status Data Operand (timer_cleared_status) */
#define CEC_OP_TIMER_CLR_STAT_RECORDING                 0x00
#define CEC_OP_TIMER_CLR_STAT_NO_MATCHING               0x01
#define CEC_OP_TIMER_CLR_STAT_NO_INFO                   0x02
#define CEC_OP_TIMER_CLR_STAT_CLEARED                   0x80

#define CEC_MSG_TIMER_STATUS                            0x35
/* Timer Overlap Warning Operand (timer_overlap_warning) */
#define CEC_OP_TIMER_OVERLAP_WARNING_NO_OVERLAP         0
#define CEC_OP_TIMER_OVERLAP_WARNING_OVERLAP            1
/* Media Info Operand (media_info) */
#define CEC_OP_MEDIA_INFO_UNPROT_MEDIA                  0
#define CEC_OP_MEDIA_INFO_PROT_MEDIA                    1
#define CEC_OP_MEDIA_INFO_NO_MEDIA                      2
/* Programmed Indicator Operand (prog_indicator) */
#define CEC_OP_PROG_IND_NOT_PROGRAMMED                  0
```

```
#define CEC_OP_PROG_IND_PROGRAMMED                      1
/* Programmed Info Operand (prog_info) */
#define CEC_OP_PROG_INFO_ENOUGH_SPACE                   0x08
#define CEC_OP_PROG_INFO_NOT_ENOUGH_SPACE               0x09
#define CEC_OP_PROG_INFO_MIGHT_NOT_BE_ENOUGH_SPACE      0x0b
#define CEC_OP_PROG_INFO_NONE_AVAILABLE                 0x0a
/* Not Programmed Error Info Operand (prog_error) */
#define CEC_OP_PROG_ERROR_NO_FREE_TIMER                 0x01
#define CEC_OP_PROG_ERROR_DATE_OUT_OF_RANGE             0x02
#define CEC_OP_PROG_ERROR_REC_SEQ_ERROR                 0x03
#define CEC_OP_PROG_ERROR_INV_EXT_PLUG                  0x04
#define CEC_OP_PROG_ERROR_INV_EXT_PHYS_ADDR             0x05
#define CEC_OP_PROG_ERROR_CA_UNSUPP                     0x06
#define CEC_OP_PROG_ERROR_INSUF_CA_ENTITLEMENTS         0x07
#define CEC_OP_PROG_ERROR_RESOLUTION_UNSUPP             0x08
#define CEC_OP_PROG_ERROR_PARENTAL_LOCK                 0x09
#define CEC_OP_PROG_ERROR_CLOCK_FAILURE                 0x0a
#define CEC_OP_PROG_ERROR_DUPLICATE                     0x0e

/* System Information Feature */
#define CEC_MSG_CEC_VERSION                             0x9e
/* CEC Version Operand (cec_version) */
#define CEC_OP_CEC_VERSION_1_3A                         4
#define CEC_OP_CEC_VERSION_1_4                          5
#define CEC_OP_CEC_VERSION_2_0                          6

#define CEC_MSG_GET_CEC_VERSION                         0x9f
#define CEC_MSG_GIVE_PHYSICAL_ADDR                      0x83
#define CEC_MSG_GET_MENU_LANGUAGE                       0x91
#define CEC_MSG_REPORT_PHYSICAL_ADDR                    0x84
/* Primary Device Type Operand (prim_devtype) */
#define CEC_OP_PRIM_DEVTYPE_TV                          0
#define CEC_OP_PRIM_DEVTYPE_RECORD                      1
#define CEC_OP_PRIM_DEVTYPE_TUNER                       3
#define CEC_OP_PRIM_DEVTYPE_PLAYBACK                    4
#define CEC_OP_PRIM_DEVTYPE_AUDIOSYSTEM                 5
#define CEC_OP_PRIM_DEVTYPE_SWITCH                      6
#define CEC_OP_PRIM_DEVTYPE_PROCESSOR                   7

#define CEC_MSG_SET_MENU_LANGUAGE                       0x32
#define CEC_MSG_REPORT_FEATURES                         0xa6    /*
↪HDMI 2.0 */
/* All Device Types Operand (all_device_types) */
#define CEC_OP_ALL_DEVTYPE_TV                           0x80
#define CEC_OP_ALL_DEVTYPE_RECORD                       0x40
#define CEC_OP_ALL_DEVTYPE_TUNER                        0x20
#define CEC_OP_ALL_DEVTYPE_PLAYBACK                     0x10
#define CEC_OP_ALL_DEVTYPE_AUDIOSYSTEM                  0x08
#define CEC_OP_ALL_DEVTYPE_SWITCH                       0x04
/*
 * And if you wondering what happened to PROCESSOR devices: those
```

```
↪should
 * be mapped to a SWITCH.
 */

/* Valid for RC Profile and Device Feature operands */
#define CEC_OP_FEAT_EXT                                 0x80    /*␣
↪Extension bit */
/* RC Profile Operand (rc_profile) */
#define CEC_OP_FEAT_RC_TV_PROFILE_NONE                  0x00
#define CEC_OP_FEAT_RC_TV_PROFILE_1                     0x02
#define CEC_OP_FEAT_RC_TV_PROFILE_2                     0x06
#define CEC_OP_FEAT_RC_TV_PROFILE_3                     0x0a
#define CEC_OP_FEAT_RC_TV_PROFILE_4                     0x0e
#define CEC_OP_FEAT_RC_SRC_HAS_DEV_ROOT_MENU            0x50
#define CEC_OP_FEAT_RC_SRC_HAS_DEV_SETUP_MENU           0x48
#define CEC_OP_FEAT_RC_SRC_HAS_CONTENTS_MENU            0x44
#define CEC_OP_FEAT_RC_SRC_HAS_MEDIA_TOP_MENU           0x42
#define CEC_OP_FEAT_RC_SRC_HAS_MEDIA_CONTEXT_MENU       0x41
/* Device Feature Operand (dev_features) */
#define CEC_OP_FEAT_DEV_HAS_RECORD_TV_SCREEN            0x40
#define CEC_OP_FEAT_DEV_HAS_SET_OSD_STRING              0x20
#define CEC_OP_FEAT_DEV_HAS_DECK_CONTROL                0x10
#define CEC_OP_FEAT_DEV_HAS_SET_AUDIO_RATE              0x08
#define CEC_OP_FEAT_DEV_SINK_HAS_ARC_TX                 0x04
#define CEC_OP_FEAT_DEV_SOURCE_HAS_ARC_RX               0x02

#define CEC_MSG_GIVE_FEATURES                           0xa5    /*␣
↪HDMI 2.0 */

/* Deck Control Feature */
#define CEC_MSG_DECK_CONTROL                            0x42
/* Deck Control Mode Operand (deck_control_mode) */
#define CEC_OP_DECK_CTL_MODE_SKIP_FWD                   1
#define CEC_OP_DECK_CTL_MODE_SKIP_REV                   2
#define CEC_OP_DECK_CTL_MODE_STOP                       3
#define CEC_OP_DECK_CTL_MODE_EJECT                      4

#define CEC_MSG_DECK_STATUS                             0x1b
/* Deck Info Operand (deck_info) */
#define CEC_OP_DECK_INFO_PLAY                           0x11
#define CEC_OP_DECK_INFO_RECORD                         0x12
#define CEC_OP_DECK_INFO_PLAY_REV                       0x13
#define CEC_OP_DECK_INFO_STILL                          0x14
#define CEC_OP_DECK_INFO_SLOW                           0x15
#define CEC_OP_DECK_INFO_SLOW_REV                       0x16
#define CEC_OP_DECK_INFO_FAST_FWD                       0x17
#define CEC_OP_DECK_INFO_FAST_REV                       0x18
#define CEC_OP_DECK_INFO_NO_MEDIA                       0x19
#define CEC_OP_DECK_INFO_STOP                           0x1a
#define CEC_OP_DECK_INFO_SKIP_FWD                       0x1b
#define CEC_OP_DECK_INFO_SKIP_REV                       0x1c
```

```
#define CEC_OP_DECK_INFO_INDEX_SEARCH_FWD               0x1d
#define CEC_OP_DECK_INFO_INDEX_SEARCH_REV               0x1e
#define CEC_OP_DECK_INFO_OTHER                          0x1f

#define CEC_MSG_GIVE_DECK_STATUS                        0x1a
/* Status Request Operand (status_req) */
#define CEC_OP_STATUS_REQ_ON                            1
#define CEC_OP_STATUS_REQ_OFF                           2
#define CEC_OP_STATUS_REQ_ONCE                          3

#define CEC_MSG_PLAY                                    0x41
/* Play Mode Operand (play_mode) */
#define CEC_OP_PLAY_MODE_PLAY_FWD                       0x24
#define CEC_OP_PLAY_MODE_PLAY_REV                       0x20
#define CEC_OP_PLAY_MODE_PLAY_STILL                     0x25
#define CEC_OP_PLAY_MODE_PLAY_FAST_FWD_MIN              0x05
#define CEC_OP_PLAY_MODE_PLAY_FAST_FWD_MED              0x06
#define CEC_OP_PLAY_MODE_PLAY_FAST_FWD_MAX              0x07
#define CEC_OP_PLAY_MODE_PLAY_FAST_REV_MIN              0x09
#define CEC_OP_PLAY_MODE_PLAY_FAST_REV_MED              0x0a
#define CEC_OP_PLAY_MODE_PLAY_FAST_REV_MAX              0x0b
#define CEC_OP_PLAY_MODE_PLAY_SLOW_FWD_MIN              0x15
#define CEC_OP_PLAY_MODE_PLAY_SLOW_FWD_MED              0x16
#define CEC_OP_PLAY_MODE_PLAY_SLOW_FWD_MAX              0x17
#define CEC_OP_PLAY_MODE_PLAY_SLOW_REV_MIN              0x19
#define CEC_OP_PLAY_MODE_PLAY_SLOW_REV_MED              0x1a
#define CEC_OP_PLAY_MODE_PLAY_SLOW_REV_MAX              0x1b

/* Tuner Control Feature */
#define CEC_MSG_GIVE_TUNER_DEVICE_STATUS                0x08
#define CEC_MSG_SELECT_ANALOGUE_SERVICE                 0x92
#define CEC_MSG_SELECT_DIGITAL_SERVICE                  0x93
#define CEC_MSG_TUNER_DEVICE_STATUS                     0x07
/* Recording Flag Operand (rec_flag) */
#define CEC_OP_REC_FLAG_NOT_USED                        0
#define CEC_OP_REC_FLAG_USED                            1
/* Tuner Display Info Operand (tuner_display_info) */
#define CEC_OP_TUNER_DISPLAY_INFO_DIGITAL               0
#define CEC_OP_TUNER_DISPLAY_INFO_NONE                  1
#define CEC_OP_TUNER_DISPLAY_INFO_ANALOGUE              2

#define CEC_MSG_TUNER_STEP_DECREMENT                    0x06
#define CEC_MSG_TUNER_STEP_INCREMENT                    0x05

/* Vendor Specific Commands Feature */

/*
 * Has also:
 *      CEC_MSG_CEC_VERSION
 *      CEC_MSG_GET_CEC_VERSION
 */
```

```
#define CEC_MSG_DEVICE_VENDOR_ID                        0x87
#define CEC_MSG_GIVE_DEVICE_VENDOR_ID                   0x8c
#define CEC_MSG_VENDOR_COMMAND                          0x89
#define CEC_MSG_VENDOR_COMMAND_WITH_ID                  0xa0
#define CEC_MSG_VENDOR_REMOTE_BUTTON_DOWN               0x8a
#define CEC_MSG_VENDOR_REMOTE_BUTTON_UP                 0x8b

/* OSD Display Feature */
#define CEC_MSG_SET_OSD_STRING                          0x64
/* Display Control Operand (disp_ctl) */
#define CEC_OP_DISP_CTL_DEFAULT                         0x00
#define CEC_OP_DISP_CTL_UNTIL_CLEARED                   0x40
#define CEC_OP_DISP_CTL_CLEAR                           0x80

/* Device OSD Transfer Feature */
#define CEC_MSG_GIVE_OSD_NAME                           0x46
#define CEC_MSG_SET_OSD_NAME                            0x47

/* Device Menu Control Feature */
#define CEC_MSG_MENU_REQUEST                            0x8d
/* Menu Request Type Operand (menu_req) */
#define CEC_OP_MENU_REQUEST_ACTIVATE                    0x00
#define CEC_OP_MENU_REQUEST_DEACTIVATE                  0x01
#define CEC_OP_MENU_REQUEST_QUERY                       0x02

#define CEC_MSG_MENU_STATUS                             0x8e
/* Menu State Operand (menu_state) */
#define CEC_OP_MENU_STATE_ACTIVATED                     0x00
#define CEC_OP_MENU_STATE_DEACTIVATED                   0x01

#define CEC_MSG_USER_CONTROL_PRESSED                    0x44
/* UI Command Operand (ui_cmd) */
#define CEC_OP_UI_CMD_SELECT                            0x00
#define CEC_OP_UI_CMD_UP                                0x01
#define CEC_OP_UI_CMD_DOWN                              0x02
#define CEC_OP_UI_CMD_LEFT                              0x03
#define CEC_OP_UI_CMD_RIGHT                             0x04
#define CEC_OP_UI_CMD_RIGHT_UP                          0x05
#define CEC_OP_UI_CMD_RIGHT_DOWN                        0x06
#define CEC_OP_UI_CMD_LEFT_UP                           0x07
#define CEC_OP_UI_CMD_LEFT_DOWN                         0x08
#define CEC_OP_UI_CMD_DEVICE_ROOT_MENU                  0x09
#define CEC_OP_UI_CMD_DEVICE_SETUP_MENU                 0x0a
#define CEC_OP_UI_CMD_CONTENTS_MENU                     0x0b
#define CEC_OP_UI_CMD_FAVORITE_MENU                     0x0c
#define CEC_OP_UI_CMD_BACK                              0x0d
#define CEC_OP_UI_CMD_MEDIA_TOP_MENU                    0x10
#define CEC_OP_UI_CMD_MEDIA_CONTEXT_SENSITIVE_MENU      0x11
#define CEC_OP_UI_CMD_NUMBER_ENTRY_MODE                 0x1d
#define CEC_OP_UI_CMD_NUMBER_11                         0x1e
#define CEC_OP_UI_CMD_NUMBER_12                         0x1f
```

```
#define CEC_OP_UI_CMD_NUMBER_0_OR_NUMBER_10          0x20
#define CEC_OP_UI_CMD_NUMBER_1                       0x21
#define CEC_OP_UI_CMD_NUMBER_2                       0x22
#define CEC_OP_UI_CMD_NUMBER_3                       0x23
#define CEC_OP_UI_CMD_NUMBER_4                       0x24
#define CEC_OP_UI_CMD_NUMBER_5                       0x25
#define CEC_OP_UI_CMD_NUMBER_6                       0x26
#define CEC_OP_UI_CMD_NUMBER_7                       0x27
#define CEC_OP_UI_CMD_NUMBER_8                       0x28
#define CEC_OP_UI_CMD_NUMBER_9                       0x29
#define CEC_OP_UI_CMD_DOT                            0x2a
#define CEC_OP_UI_CMD_ENTER                          0x2b
#define CEC_OP_UI_CMD_CLEAR                          0x2c
#define CEC_OP_UI_CMD_NEXT_FAVORITE                  0x2f
#define CEC_OP_UI_CMD_CHANNEL_UP                     0x30
#define CEC_OP_UI_CMD_CHANNEL_DOWN                   0x31
#define CEC_OP_UI_CMD_PREVIOUS_CHANNEL               0x32
#define CEC_OP_UI_CMD_SOUND_SELECT                   0x33
#define CEC_OP_UI_CMD_INPUT_SELECT                   0x34
#define CEC_OP_UI_CMD_DISPLAY_INFORMATION            0x35
#define CEC_OP_UI_CMD_HELP                           0x36
#define CEC_OP_UI_CMD_PAGE_UP                        0x37
#define CEC_OP_UI_CMD_PAGE_DOWN                      0x38
#define CEC_OP_UI_CMD_POWER                          0x40
#define CEC_OP_UI_CMD_VOLUME_UP                      0x41
#define CEC_OP_UI_CMD_VOLUME_DOWN                    0x42
#define CEC_OP_UI_CMD_MUTE                           0x43
#define CEC_OP_UI_CMD_PLAY                           0x44
#define CEC_OP_UI_CMD_STOP                           0x45
#define CEC_OP_UI_CMD_PAUSE                          0x46
#define CEC_OP_UI_CMD_RECORD                         0x47
#define CEC_OP_UI_CMD_REWIND                         0x48
#define CEC_OP_UI_CMD_FAST_FORWARD                   0x49
#define CEC_OP_UI_CMD_EJECT                          0x4a
#define CEC_OP_UI_CMD_SKIP_FORWARD                   0x4b
#define CEC_OP_UI_CMD_SKIP_BACKWARD                  0x4c
#define CEC_OP_UI_CMD_STOP_RECORD                    0x4d
#define CEC_OP_UI_CMD_PAUSE_RECORD                   0x4e
#define CEC_OP_UI_CMD_ANGLE                          0x50
#define CEC_OP_UI_CMD_SUB_PICTURE                    0x51
#define CEC_OP_UI_CMD_VIDEO_ON_DEMAND                0x52
#define CEC_OP_UI_CMD_ELECTRONIC_PROGRAM_GUIDE       0x53
#define CEC_OP_UI_CMD_TIMER_PROGRAMMING              0x54
#define CEC_OP_UI_CMD_INITIAL_CONFIGURATION          0x55
#define CEC_OP_UI_CMD_SELECT_BROADCAST_TYPE          0x56
#define CEC_OP_UI_CMD_SELECT_SOUND_PRESENTATION      0x57
#define CEC_OP_UI_CMD_AUDIO_DESCRIPTION              0x58
#define CEC_OP_UI_CMD_INTERNET                       0x59
#define CEC_OP_UI_CMD_3D_MODE                        0x5a
#define CEC_OP_UI_CMD_PLAY_FUNCTION                  0x60
#define CEC_OP_UI_CMD_PAUSE_PLAY_FUNCTION            0x61
```

```
#define CEC_OP_UI_CMD_RECORD_FUNCTION                   0x62
#define CEC_OP_UI_CMD_PAUSE_RECORD_FUNCTION             0x63
#define CEC_OP_UI_CMD_STOP_FUNCTION                     0x64
#define CEC_OP_UI_CMD_MUTE_FUNCTION                     0x65
#define CEC_OP_UI_CMD_RESTORE_VOLUME_FUNCTION           0x66
#define CEC_OP_UI_CMD_TUNE_FUNCTION                     0x67
#define CEC_OP_UI_CMD_SELECT_MEDIA_FUNCTION             0x68
#define CEC_OP_UI_CMD_SELECT_AV_INPUT_FUNCTION          0x69
#define CEC_OP_UI_CMD_SELECT_AUDIO_INPUT_FUNCTION       0x6a
#define CEC_OP_UI_CMD_POWER_TOGGLE_FUNCTION             0x6b
#define CEC_OP_UI_CMD_POWER_OFF_FUNCTION                0x6c
#define CEC_OP_UI_CMD_POWER_ON_FUNCTION                 0x6d
#define CEC_OP_UI_CMD_F1_BLUE                           0x71
#define CEC_OP_UI_CMD_F2_RED                            0x72
#define CEC_OP_UI_CMD_F3_GREEN                          0x73
#define CEC_OP_UI_CMD_F4_YELLOW                         0x74
#define CEC_OP_UI_CMD_F5                                0x75
#define CEC_OP_UI_CMD_DATA                              0x76
/* UI Broadcast Type Operand (ui_bcast_type) */
#define CEC_OP_UI_BCAST_TYPE_TOGGLE_ALL                 0x00
#define CEC_OP_UI_BCAST_TYPE_TOGGLE_DIG_ANA             0x01
#define CEC_OP_UI_BCAST_TYPE_ANALOGUE                   0x10
#define CEC_OP_UI_BCAST_TYPE_ANALOGUE_T                 0x20
#define CEC_OP_UI_BCAST_TYPE_ANALOGUE_CABLE             0x30
#define CEC_OP_UI_BCAST_TYPE_ANALOGUE_SAT               0x40
#define CEC_OP_UI_BCAST_TYPE_DIGITAL                    0x50
#define CEC_OP_UI_BCAST_TYPE_DIGITAL_T                  0x60
#define CEC_OP_UI_BCAST_TYPE_DIGITAL_CABLE              0x70
#define CEC_OP_UI_BCAST_TYPE_DIGITAL_SAT                0x80
#define CEC_OP_UI_BCAST_TYPE_DIGITAL_COM_SAT            0x90
#define CEC_OP_UI_BCAST_TYPE_DIGITAL_COM_SAT2           0x91
#define CEC_OP_UI_BCAST_TYPE_IP                         0xa0
/* UI Sound Presentation Control Operand (ui_snd_pres_ctl) */
#define CEC_OP_UI_SND_PRES_CTL_DUAL_MONO                0x10
#define CEC_OP_UI_SND_PRES_CTL_KARAOKE                  0x20
#define CEC_OP_UI_SND_PRES_CTL_DOWNMIX                  0x80
#define CEC_OP_UI_SND_PRES_CTL_REVERB                   0x90
#define CEC_OP_UI_SND_PRES_CTL_EQUALIZER                0xa0
#define CEC_OP_UI_SND_PRES_CTL_BASS_UP                  0xb1
#define CEC_OP_UI_SND_PRES_CTL_BASS_NEUTRAL             0xb2
#define CEC_OP_UI_SND_PRES_CTL_BASS_DOWN                0xb3
#define CEC_OP_UI_SND_PRES_CTL_TREBLE_UP                0xc1
#define CEC_OP_UI_SND_PRES_CTL_TREBLE_NEUTRAL           0xc2
#define CEC_OP_UI_SND_PRES_CTL_TREBLE_DOWN              0xc3

#define CEC_MSG_USER_CONTROL_RELEASED                   0x45

/* Remote Control Passthrough Feature */

/*
 * Has also:
```

```
 *      CEC_MSG_USER_CONTROL_PRESSED
 *      CEC_MSG_USER_CONTROL_RELEASED
 */

/* Power Status Feature */
#define CEC_MSG_GIVE_DEVICE_POWER_STATUS               0x8f
#define CEC_MSG_REPORT_POWER_STATUS                    0x90
/* Power Status Operand (pwr_state) */
#define CEC_OP_POWER_STATUS_ON                         0
#define CEC_OP_POWER_STATUS_STANDBY                    1
#define CEC_OP_POWER_STATUS_TO_ON                      2
#define CEC_OP_POWER_STATUS_TO_STANDBY                 3

/* General Protocol Messages */
#define CEC_MSG_FEATURE_ABORT                          0x00
/* Abort Reason Operand (reason) */
#define CEC_OP_ABORT_UNRECOGNIZED_OP                   0
#define CEC_OP_ABORT_INCORRECT_MODE                    1
#define CEC_OP_ABORT_NO_SOURCE                         2
#define CEC_OP_ABORT_INVALID_OP                        3
#define CEC_OP_ABORT_REFUSED                           4
#define CEC_OP_ABORT_UNDETERMINED                      5

#define CEC_MSG_ABORT                                  0xff

/* System Audio Control Feature */

/*
 * Has also:
 *      CEC_MSG_USER_CONTROL_PRESSED
 *      CEC_MSG_USER_CONTROL_RELEASED
 */
#define CEC_MSG_GIVE_AUDIO_STATUS                      0x71
#define CEC_MSG_GIVE_SYSTEM_AUDIO_MODE_STATUS          0x7d
#define CEC_MSG_REPORT_AUDIO_STATUS                    0x7a
/* Audio Mute Status Operand (aud_mute_status) */
#define CEC_OP_AUD_MUTE_STATUS_OFF                     0
#define CEC_OP_AUD_MUTE_STATUS_ON                      1

#define CEC_MSG_REPORT_SHORT_AUDIO_DESCRIPTOR          0xa3
#define CEC_MSG_REQUEST_SHORT_AUDIO_DESCRIPTOR         0xa4
#define CEC_MSG_SET_SYSTEM_AUDIO_MODE                  0x72
/* System Audio Status Operand (sys_aud_status) */
#define CEC_OP_SYS_AUD_STATUS_OFF                      0
#define CEC_OP_SYS_AUD_STATUS_ON                       1

#define CEC_MSG_SYSTEM_AUDIO_MODE_REQUEST              0x70
#define CEC_MSG_SYSTEM_AUDIO_MODE_STATUS               0x7e
/* Audio Format ID Operand (audio_format_id) */
#define CEC_OP_AUD_FMT_ID_CEA861                       0
#define CEC_OP_AUD_FMT_ID_CEA861_CXT                   1
```

```
/* Audio Rate Control Feature */
#define CEC_MSG_SET_AUDIO_RATE                          0x9a
/* Audio Rate Operand (audio_rate) */
#define CEC_OP_AUD_RATE_OFF                             0
#define CEC_OP_AUD_RATE_WIDE_STD                        1
#define CEC_OP_AUD_RATE_WIDE_FAST                       2
#define CEC_OP_AUD_RATE_WIDE_SLOW                       3
#define CEC_OP_AUD_RATE_NARROW_STD                      4
#define CEC_OP_AUD_RATE_NARROW_FAST                     5
#define CEC_OP_AUD_RATE_NARROW_SLOW                     6

/* Audio Return Channel Control Feature */
#define CEC_MSG_INITIATE_ARC                            0xc0
#define CEC_MSG_REPORT_ARC_INITIATED                    0xc1
#define CEC_MSG_REPORT_ARC_TERMINATED                   0xc2
#define CEC_MSG_REQUEST_ARC_INITIATION                  0xc3
#define CEC_MSG_REQUEST_ARC_TERMINATION                 0xc4
#define CEC_MSG_TERMINATE_ARC                           0xc5

/* Dynamic Audio Lipsync Feature */
/* Only for CEC 2.0 and up */
#define CEC_MSG_REQUEST_CURRENT_LATENCY                 0xa7
#define CEC_MSG_REPORT_CURRENT_LATENCY                  0xa8
/* Low Latency Mode Operand (low_latency_mode) */
#define CEC_OP_LOW_LATENCY_MODE_OFF                     0
#define CEC_OP_LOW_LATENCY_MODE_ON                      1
/* Audio Output Compensated Operand (audio_out_compensated) */
#define CEC_OP_AUD_OUT_COMPENSATED_NA                   0
#define CEC_OP_AUD_OUT_COMPENSATED_DELAY                1
#define CEC_OP_AUD_OUT_COMPENSATED_NO_DELAY             2
#define CEC_OP_AUD_OUT_COMPENSATED_PARTIAL_DELAY        3

/* Capability Discovery and Control Feature */
#define CEC_MSG_CDC_MESSAGE                             0xf8
/* Ethernet-over-HDMI: nobody ever does this... */
#define CEC_MSG_CDC_HEC_INQUIRE_STATE                   0x00
#define CEC_MSG_CDC_HEC_REPORT_STATE                    0x01
/* HEC Functionality State Operand (hec_func_state) */
#define CEC_OP_HEC_FUNC_STATE_NOT_SUPPORTED             0
#define CEC_OP_HEC_FUNC_STATE_INACTIVE                  1
#define CEC_OP_HEC_FUNC_STATE_ACTIVE                    2
#define CEC_OP_HEC_FUNC_STATE_ACTIVATION_FIELD          3
/* Host Functionality State Operand (host_func_state) */
#define CEC_OP_HOST_FUNC_STATE_NOT_SUPPORTED            0
#define CEC_OP_HOST_FUNC_STATE_INACTIVE                 1
#define CEC_OP_HOST_FUNC_STATE_ACTIVE                   2
/* ENC Functionality State Operand (enc_func_state) */
#define CEC_OP_ENC_FUNC_STATE_EXT_CON_NOT_SUPPORTED     0
#define CEC_OP_ENC_FUNC_STATE_EXT_CON_INACTIVE          1
#define CEC_OP_ENC_FUNC_STATE_EXT_CON_ACTIVE            2
```

```
/* CDC Error Code Operand (cdc_errcode) */
#define CEC_OP_CDC_ERROR_CODE_NONE                      0
#define CEC_OP_CDC_ERROR_CODE_CAP_UNSUPPORTED           1
#define CEC_OP_CDC_ERROR_CODE_WRONG_STATE               2
#define CEC_OP_CDC_ERROR_CODE_OTHER                     3
/* HEC Support Operand (hec_support) */
#define CEC_OP_HEC_SUPPORT_NO                           0
#define CEC_OP_HEC_SUPPORT_YES                          1
/* HEC Activation Operand (hec_activation) */
#define CEC_OP_HEC_ACTIVATION_ON                        0
#define CEC_OP_HEC_ACTIVATION_OFF                       1

#define CEC_MSG_CDC_HEC_SET_STATE_ADJACENT              0x02
#define CEC_MSG_CDC_HEC_SET_STATE                       0x03
/* HEC Set State Operand (hec_set_state) */
#define CEC_OP_HEC_SET_STATE_DEACTIVATE                 0
#define CEC_OP_HEC_SET_STATE_ACTIVATE                   1

#define CEC_MSG_CDC_HEC_REQUEST_DEACTIVATION            0x04
#define CEC_MSG_CDC_HEC_NOTIFY_ALIVE                    0x05
#define CEC_MSG_CDC_HEC_DISCOVER                        0x06
/* Hotplug Detect messages */
#define CEC_MSG_CDC_HPD_SET_STATE                       0x10
/* HPD State Operand (hpd_state) */
#define CEC_OP_HPD_STATE_CP_EDID_DISABLE                0
#define CEC_OP_HPD_STATE_CP_EDID_ENABLE                 1
#define CEC_OP_HPD_STATE_CP_EDID_DISABLE_ENABLE         2
#define CEC_OP_HPD_STATE_EDID_DISABLE                   3
#define CEC_OP_HPD_STATE_EDID_ENABLE                    4
#define CEC_OP_HPD_STATE_EDID_DISABLE_ENABLE            5
#define CEC_MSG_CDC_HPD_REPORT_STATE                    0x11
/* HPD Error Code Operand (hpd_error) */
#define CEC_OP_HPD_ERROR_NONE                           0
#define CEC_OP_HPD_ERROR_INITIATOR_NOT_CAPABLE          1
#define CEC_OP_HPD_ERROR_INITIATOR_WRONG_STATE          2
#define CEC_OP_HPD_ERROR_OTHER                          3
#define CEC_OP_HPD_ERROR_NONE_NO_VIDEO                  4

/* End of Messages */

/* Helper functions to identify the 'special' CEC devices */

static inline int cec_is_2nd_tv(const struct cec_log_addrs *las)
{
        /*
         * It is a second TV if the logical address is 14 or 15 and␣
↪the
         * primary device type is a TV.
         */
        return las->num_log_addrs &&
               las->log_addr[0] >= CEC_LOG_ADDR_SPECIFIC &&
```

```
                las->primary_device_type[0] == CEC_OP_PRIM_DEVTYPE_
↪TV;
}

static inline int cec_is_processor(const struct cec_log_addrs *las)
{
        /*
         * It is a processor if the logical address is 12-15 and the
         * primary device type is a Processor.
         */
        return las->num_log_addrs &&
                las->log_addr[0] >= CEC_LOG_ADDR_BACKUP_1 &&
                las->primary_device_type[0] == CEC_OP_PRIM_DEVTYPE_
↪PROCESSOR;
}

static inline int cec_is_switch(const struct cec_log_addrs *las)
{
        /*
         * It is a switch if the logical address is 15 and the
         * primary device type is a Switch and the CDC-Only flag is␣
↪not set.
         */
        return las->num_log_addrs == 1 &&
                las->log_addr[0] == CEC_LOG_ADDR_UNREGISTERED &&
                las->primary_device_type[0] == CEC_OP_PRIM_DEVTYPE_
↪SWITCH &&
                !(las->flags & CEC_LOG_ADDRS_FL_CDC_ONLY);
}

static inline int cec_is_cdc_only(const struct cec_log_addrs *las)
{
        /*
         * It is a CDC-only device if the logical address is 15 and␣
↪the
         * primary device type is a Switch and the CDC-Only flag is␣
↪set.
         */
        return las->num_log_addrs == 1 &&
                las->log_addr[0] == CEC_LOG_ADDR_UNREGISTERED &&
                las->primary_device_type[0] == CEC_OP_PRIM_DEVTYPE_
↪SWITCH &&
                (las->flags & CEC_LOG_ADDRS_FL_CDC_ONLY);
}

#endif
```

8.6.5 Revision and Copyright

Authors:

- Verkuil, Hans <hverkuil-cisco@xs4all.nl>
- Initial version.

8.6.6 Revision History

revision
1.0.0 / 2016-03-17 (*hv*)

Initial revision

8.7 Generic Error Codes

Table 275: Generic error codes

EAGAIN (aka EWOULDBLOCK	The ioctl can't be handled because the device is in state where it can't perform it. This could happen for example in case where device is sleeping and ioctl is performed to query statistics. It is also returned when the ioctl would need to wait for an event, but the device was opened in non-blocking mode.
EBADF	The file descriptor is not a valid.
EBUSY	The ioctl can' t be handled because the device is busy. This is typically return while device is streaming, and an ioctl tried to change something that would affect the stream, or would require the usage of a hardware resource that was already allocated. The ioctl must not be retried without performing another action to fix the problem first (typically: stop the stream before retrying).
EFAULT	There was a failure while copying data from/to userspace, probably caused by an invalid pointer reference.
EINVAL	One or more of the ioctl parameters are invalid or out of the allowed range. This is a widely used error code. See the individual ioctl requests for specific causes.
ENODEV	Device not found or was removed.
ENOMEM	There' s not enough memory to handle the desired operation.
ENOTTY	The ioctl is not supported by the driver, actually meaning that the required functionality is not available, or the file descriptor is not for a media device.
ENOSPC	On USB devices, the stream ioctl' s can return this error, meaning that this request would overcommit the usb bandwidth reserved for periodic transfers (up to 80% of the USB bandwidth).
EPERM	Permission denied. Can be returned if the device needs write permission, or some special capabilities is needed (e. g. root)
EIO	I/O error. Typically used when there are problems communicating with a hardware device. This could indicate broken or flaky hardware. It' s a 'Something is wrong, I give up!' type of error.
ENXIO	No device corresponding to this device special file exists.

Note:

1. This list is not exhaustive; ioctls may return other error codes. Since errors may have side effects such as a driver reset, applications should abort on unexpected errors, or otherwise assume that the device is in a bad state.
2. Request-specific error codes are listed in the individual requests descriptions.

8.8 Glossary

Note: The goal of this section is to standardize the terms used within the media userspace API documentation. This is Work In Progress.

Bridge Driver
: A *Device Driver* that implements the main logic to talk with media hardware.

CEC API
: **Consumer Electronics Control API**

  An API designed to receive and transmit data via an HDMI CEC interface.

  See *Part V - Consumer Electronics Control API*.

Device Driver
: Part of the Linux Kernel that implements support for a hardware component.

Device Node
: A character device node in the file system used to control and transfer data in and out of a Kernel driver.

Digital TV API
: **Previously known as DVB API**

  An API designed to control a subset of the *Media Hardware* that implements digital TV (e. g. DVB, ATSC, ISDB, etc).

  See *Part II - Digital TV API*.

DSP
: **Digital Signal Processor**

  A specialized *Microprocessor*, with its architecture optimized for the operational needs of digital signal processing.

FPGA
: **Field-programmable Gate Array**

  An *IC* circuit designed to be configured by a customer or a designer after manufacturing.

  See https://en.wikipedia.org/wiki/Field-programmable_gate_array.

Hardware Component

A subset of the *Media Hardware*. For example an *I^2C* or *SPI* device, or an *IP Block* inside an *SoC* or *FPGA*.

Hardware Peripheral

A group of *hardware components* that together make a larger user-facing functional peripheral. For instance, the *SoC ISP IP Block* and the external camera sensors together make a camera hardware peripheral.

Also known as *Peripheral*.

I^2C

Inter-Integrated Circuit

A multi-master, multi-slave, packet switched, single-ended, serial computer bus used to control some hardware components like sub-device hardware components.

See http://www.nxp.com/docs/en/user-guide/UM10204.pdf.

IC

Integrated circuit

A set of electronic circuits on one small flat piece of semiconductor material, normally silicon.

Also known as chip.

IP Block

Intellectual property core

In electronic design a semiconductor intellectual property core, is a reusable unit of logic, cell, or integrated circuit layout design that is the intellectual property of one party. IP Blocks may be licensed to another party or can be owned and used by a single party alone.

See https://en.wikipedia.org/wiki/Semiconductor_intellectual_property_core).

ISP

Image Signal Processor

A specialized processor that implements a set of algorithms for processing image data. ISPs may implement algorithms for lens shading correction, demosaicing, scaling and pixel format conversion as well as produce statistics for the use of the control algorithms (e.g. automatic exposure, white balance and focus).

Media API

A set of userspace APIs used to control the media hardware. It is composed by:

- *CEC API*;
- *Digital TV API*;
- *MC API*;
- *RC API*; and

- *V4L2 API*.

See *Linux Media Infrastructure userspace API*.

MC API
Media Controller API

An API designed to expose and control the relationships between multimedia devices and sub-devices.

See *Part IV - Media Controller API*.

MC-centric
V4L2 Hardware device driver that requires *MC API*.

Such drivers have `V4L2_CAP_IO_MC` device_caps field set (see *ioctl VIDIOC_QUERYCAP*).

See *Controlling a hardware peripheral via V4L2* for more details.

Media Hardware
Subset of the hardware that is supported by the Linux Media API.

This includes audio and video capture and playback hardware, digital and analog TV, camera sensors, ISPs, remote controllers, codecs, HDMI Consumer Electronics Control, HDMI capture, etc.

Microprocessor
Electronic circuitry that carries out the instructions of a computer program by performing the basic arithmetic, logical, control and input/output (I/O) operations specified by the instructions on a single integrated circuit.

Peripheral
The same as *Hardware Peripheral*.

RC API
Remote Controller API

An API designed to receive and transmit data from remote controllers.

See *Part III - Remote Controller API*.

SMBus
A subset of I^2C, which defines a stricter usage of the bus.

SPI
Serial Peripheral Interface Bus

Synchronous serial communication interface specification used for short distance communication, primarily in embedded systems.

SoC
System on a Chip

An integrated circuit that integrates all components of a computer or other electronic systems.

V4L2 API
V4L2 userspace API

The userspace API defined in *Part I - Video for Linux API*, which is used to control a V4L2 hardware.

V4L2 Device Node
A *Device Node* that is associated to a V4L driver.

The V4L2 device node naming is specified at *V4L2 Device Node Naming*.

V4L2 Hardware
Part of the media hardware which is supported by the *V4L2 API*.

V4L2 Sub-device
V4L2 hardware components that aren't controlled by a *Bridge Driver*. See *Sub-device Interface*.

Video-node-centric
V4L2 device driver that doesn't require a media controller to be used.

Such drivers have the `V4L2_CAP_IO_MC` device_caps field unset (see *ioctl VIDIOC_QUERYCAP*).

V4L2 Sub-device API
Part of the *V4L2 API* which control *V4L2 sub-devices*, like sensors, HDMI receivers, scalers, deinterlacers.

See *Controlling a hardware peripheral via V4L2* for more details.

8.9 GNU Free Documentation License

8.9.1 0. PREAMBLE

The purpose of this License is to make a manual, textbook, or other written document "free" in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or noncommercially. Secondarily, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

This License is a kind of "copyleft", which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

8.9.2 1. APPLICABILITY AND DEFINITIONS

This License applies to any manual or other work that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. The "Document", below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as "you".

A "Modified Version" of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

A "Secondary Section" is a named appendix or a front-matter section of the *Document* that deals exclusively with the relationship of the publishers or authors of the Document to the Document's overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (For example, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

The "Invariant Sections" are certain *Secondary Sections* whose titles are designated, as being those of Invariant Sections, in the notice that says that the *Document* is released under this License.

The "Cover Texts" are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the *Document* is released under this License.

A "Transparent" copy of the *Document* means a machine-readable copy, represented in a format whose specification is available to the general public, whose contents can be viewed and edited directly and straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup has been designed to thwart or discourage subsequent modification by readers is not Transparent. A copy that is not "Transparent" is called "Opaque".

Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTeX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML designed for human modification. Opaque formats include PostScript, PDF, proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML produced by some word processors for output purposes only.

The "Title Page" means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, "Title Page" means the text near the most prominent appearance of the work's title, preceding the beginning of the body of the text.

8.9.3 2. VERBATIM COPYING

You may copy and distribute the *Document* in any medium, either commercially or noncommercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in *section 3*.

You may also lend copies, under the same conditions stated above, and you may publicly display copies.

8.9.4 3. COPYING IN QUANTITY

If you publish printed copies of the *Document* numbering more than 100, and the Document's license notice requires *Cover Texts*, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the *Document* and satisfy these conditions, can be treated as verbatim copying in other respects.

If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

If you publish or distribute *Opaque* copies of the *Document* numbering more than 100, you must either include a machine-readable *Transparent* copy along with each Opaque copy, or state in or with each Opaque copy a publicly-accessible computer-network location containing a complete Transparent copy of the Document, free of added material, which the general network-using public has access to download anonymously at no charge using public-standard network protocols. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

It is requested, but not required, that you contact the authors of the *Document* well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

8.9.5 4. MODIFICATIONS

You may copy and distribute a *Modified Version* of the *Document* under the conditions of sections *2* and *3* above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

- **A.** Use in the *Title Page* (and on the covers, if any) a title distinct from that of the *Document*, and from those of previous versions (which should, if there were any, be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission.
- **B.** List on the *Title Page*, as authors, one or more persons or entities responsible for authorship of the modifications in the *Modified Version*, together with at least five of the principal authors of the *Document* (all of its principal authors, if it has less than five).
- **C.** State on the *Title Page* the name of the publisher of the *Modified Version*, as the publisher.
- **D.** Preserve all the copyright notices of the *Document*.
- **E.** Add an appropriate copyright notice for your modifications adjacent to the other copyright notices.
- **F.** Include, immediately after the copyright notices, a license notice giving the public permission to use the *Modified Version* under the terms of this License, in the form shown in the Addendum below.
- **G.** Preserve in that license notice the full lists of *Invariant Sections* and required *Cover Texts* given in the *Document's* license notice.
- **H.** Include an unaltered copy of this License.
- **I.** Preserve the section entitled "History", and its title, and add to it an item stating at least the title, year, new authors, and publisher of the *Modified Version* as given on the *Title Page*. If there is no section entitled "History" in the *Document*, create one stating the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence.
- **J.** Preserve the network location, if any, given in the *Document* for public access to a *Transparent* copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the "History" section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission.
- **K.** In any section entitled "Acknowledgements" or "Dedications", preserve the section's title, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein.
- **L.** Preserve all the *Invariant Sections* of the *Document*, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part

of the section titles.

- **M.** Delete any section entitled "Endorsements". Such a section may not be included in the *Modified Version*.
- **N.** Do not retitle any existing section as "Endorsements" or to conflict in title with any *Invariant Section*.

If the *Modified Version* includes new front-matter sections or appendices that qualify as *Secondary Sections* and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their titles to the list of *Invariant Sections* in the Modified Version's license notice. These titles must be distinct from any other section titles.

You may add a section entitled "Endorsements", provided it contains nothing but endorsements of your *Modified Version* by various parties–for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a *Front-Cover Text*, and a passage of up to 25 words as a *Back-Cover Text*, to the end of the list of *Cover Texts* in the *Modified Version*. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the *Document* already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the *Document* do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any *Modified Version*.

8.9.6 5. COMBINING DOCUMENTS

You may combine the *Document* with other documents released under this License, under the terms defined in *section 4* above for modified versions, provided that you include in the combination all of the *Invariant Sections* of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice.

The combined work need only contain one copy of this License, and multiple identical *Invariant Sections* may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

In the combination, you must combine any sections entitled "History" in the various original documents, forming one section entitled "History"; likewise combine any sections entitled "Acknowledgements", and any sections entitled "Dedications". You must delete all sections entitled "Endorsements."

8.9.7 6. COLLECTIONS OF DOCUMENTS

You may make a collection consisting of the *Document* and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.

You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

8.9.8 7. AGGREGATION WITH INDEPENDENT WORKS

A compilation of the *Document* or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, does not as a whole count as a *Modified Version* of the Document, provided no compilation copyright is claimed for the compilation. Such a compilation is called an "aggregate", and this License does not apply to the other self-contained works thus compiled with the Document , on account of their being thus compiled, if they are not themselves derivative works of the Document. If the *Cover Text* requirement of *section 3* is applicable to these copies of the Document, then if the Document is less than one quarter of the entire aggregate, the Document' s Cover Texts may be placed on covers that surround only the Document within the aggregate. Otherwise they must appear on covers around the whole aggregate.

8.9.9 8. TRANSLATION

Translation is considered a kind of modification, so you may distribute translations of the *Document* under the terms of *section 4*. Replacing *Invariant Sections* with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License provided that you also include the original English version of this License. In case of a disagreement between the translation and the original English version of this License, the original English version will prevail.

8.9.10 9. TERMINATION

You may not copy, modify, sublicense, or distribute the *Document* except as expressly provided for under this License. Any other attempt to copy, modify, sublicense or distribute the Document is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

8.9.11 10. FUTURE REVISIONS OF THIS LICENSE

The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. See http://www.gnu.org/copyleft/.

Each version of the License is given a distinguishing version number. If the *Document* specifies that a particular numbered version of this License "or any later version" applies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation.

8.9.12 Addendum

To use this License in a document you have written, include a copy of the License in the document and put the following copyright and license notices just after the title page:

> Copyright © YEAR YOUR NAME.
>
> Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.1 or any later version published by the Free Software Foundation; with the *Invariant Sections* being LIST THEIR TITLES, with the *Front-Cover Texts* being LIST, and with the *Back-Cover Texts* being LIST. A copy of the license is included in the section entitled "GNU Free Documentation License".

If you have no *Invariant Sections*, write "with no Invariant Sections" instead of saying which ones are invariant. If you have no *Front-Cover Texts*, write "no Front-Cover Texts" instead of "Front-Cover Texts being LIST"; likewise for *Back-Cover Texts*.

If your document contains nontrivial examples of program code, we recommend releasing these examples in parallel under your choice of free software license, such as the GNU General Public License, to permit their use in free software.

8.10 Video4Linux (V4L) driver-specific documentation

For more details see the file COPYING in the source distribution of Linux.

8.10.1 The cx2341x driver

Non-compressed file format

The cx23416 can produce (and the cx23415 can also read) raw YUV output. The format of a YUV frame is specific to this chip and is called HM12. 'HM' stands for 'Hauppauge Macroblock', which is a misnomer as 'Conexant Macroblock' would be more accurate.

The format is YUV 4:2:0 which uses 1 Y byte per pixel and 1 U and V byte per four pixels.

The data is encoded as two macroblock planes, the first containing the Y values, the second containing UV macroblocks.

The Y plane is divided into blocks of 16x16 pixels from left to right and from top to bottom. Each block is transmitted in turn, line-by-line.

So the first 16 bytes are the first line of the top-left block, the second 16 bytes are the second line of the top-left block, etc. After transmitting this block the first line of the block on the right to the first block is transmitted, etc.

The UV plane is divided into blocks of 16x8 UV values going from left to right, top to bottom. Each block is transmitted in turn, line-by-line.

So the first 16 bytes are the first line of the top-left block and contain 8 UV value pairs (16 bytes in total). The second 16 bytes are the second line of 8 UV pairs of the top-left block, etc. After transmitting this block the first line of the block on the right to the first block is transmitted, etc.

The code below is given as an example on how to convert HM12 to separate Y, U and V planes. This code assumes frames of 720x576 (PAL) pixels.

The width of a frame is always 720 pixels, regardless of the actual specified width.

If the height is not a multiple of 32 lines, then the captured video is missing macroblocks at the end and is unusable. So the height must be a multiple of 32.

Raw format c example

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

static unsigned char frame[576*720*3/2];
static unsigned char framey[576*720];
static unsigned char frameu[576*720 / 4];
static unsigned char framev[576*720 / 4];

static void de_macro_y(unsigned char* dst, unsigned char *src, int␣
↪dstride, int w, int h)
```

(continues on next page)

(continued from previous page)

```
{
unsigned int y, x, i;

// descramble Y plane
// dstride = 720 = w
// The Y plane is divided into blocks of 16x16 pixels
// Each block in transmitted in turn, line-by-line.
for (y = 0; y < h; y += 16) {
        for (x = 0; x < w; x += 16) {
        for (i = 0; i < 16; i++) {
                memcpy(dst + x + (y + i) * dstride, src, 16);
                src += 16;
        }
        }
}
}

static void de_macro_uv(unsigned char *dstu, unsigned char *dstv,␣
↪unsigned char *src, int dstride, int w, int h)
{
unsigned int y, x, i;

// descramble U/V plane
// dstride = 720 / 2 = w
// The U/V values are interlaced (UVUV...).
// Again, the UV plane is divided into blocks of 16x16 UV values.
// Each block in transmitted in turn, line-by-line.
for (y = 0; y < h; y += 16) {
        for (x = 0; x < w; x += 8) {
        for (i = 0; i < 16; i++) {
                int idx = x + (y + i) * dstride;

                dstu[idx+0] = src[0];  dstv[idx+0] = src[1];
                dstu[idx+1] = src[2];  dstv[idx+1] = src[3];
                dstu[idx+2] = src[4];  dstv[idx+2] = src[5];
                dstu[idx+3] = src[6];  dstv[idx+3] = src[7];
                dstu[idx+4] = src[8];  dstv[idx+4] = src[9];
                dstu[idx+5] = src[10]; dstv[idx+5] = src[11];
                dstu[idx+6] = src[12]; dstv[idx+6] = src[13];
                dstu[idx+7] = src[14]; dstv[idx+7] = src[15];
                src += 16;
        }
        }
}
}

/
↪**********************************************************************/
↪
```

(continues on next page)

(continued from previous page)

```
int main(int argc, char **argv)
{
FILE *fin;
int i;

if (argc == 1) fin = stdin;
else fin = fopen(argv[1], "r");

if (fin == NULL) {
        fprintf(stderr, "cannot open input\n");
        exit(-1);
}
while (fread(frame, sizeof(frame), 1, fin) == 1) {
        de_macro_y(framey, frame, 720, 720, 576);
        de_macro_uv(frameu, framev, frame + 720 * 576, 720 / 2, 720␣
↪/ 2, 576 / 2);
        fwrite(framey, sizeof(framey), 1, stdout);
        fwrite(framev, sizeof(framev), 1, stdout);
        fwrite(frameu, sizeof(frameu), 1, stdout);
}
fclose(fin);
return 0;
}
```

Format of embedded V4L2_MPEG_STREAM_VBI_FMT_IVTV VBI data

Author: Hans Verkuil <hverkuil@xs4all.nl>

This section describes the V4L2_MPEG_STREAM_VBI_FMT_IVTV format of the VBI data embedded in an MPEG-2 program stream. This format is in part dictated by some hardware limitations of the ivtv driver (the driver for the Conexant cx23415/6 chips), in particular a maximum size for the VBI data. Anything longer is cut off when the MPEG stream is played back through the cx23415.

The advantage of this format is it is very compact and that all VBI data for all lines can be stored while still fitting within the maximum allowed size.

The stream ID of the VBI data is 0xBD. The maximum size of the embedded data is 4 + 43 * 36, which is 4 bytes for a header and 2 * 18 VBI lines with a 1 byte header and a 42 bytes payload each. Anything beyond this limit is cut off by the cx23415/6 firmware. Besides the data for the VBI lines we also need 36 bits for a bitmask determining which lines are captured and 4 bytes for a magic cookie, signifying that this data package contains V4L2_MPEG_STREAM_VBI_FMT_IVTV VBI data. If all lines are used, then there is no longer room for the bitmask. To solve this two different magic numbers were introduced:

'itv0' : After this magic number two unsigned longs follow. Bits 0-17 of the first unsigned long denote which lines of the first field are captured. Bits 18-31 of the first unsigned long and bits 0-3 of the second unsigned long are used for the second field.

'ITV0' : This magic number assumes all VBI lines are captured, i.e. it implicitly implies that the bitmasks are 0xffffffff and 0xf.

After these magic cookies (and the 8 byte bitmask in case of cookie 'itv0') the captured VBI lines start:

For each line the least significant 4 bits of the first byte contain the data type. Possible values are shown in the table below. The payload is in the following 42 bytes.

Here is the list of possible data types:

```
#define IVTV_SLICED_TYPE_TELETEXT       0x1     // Teletext (uses␣
↪lines 6-22 for PAL)
#define IVTV_SLICED_TYPE_CC             0x4     // Closed Captions␣
↪(line 21 NTSC)
#define IVTV_SLICED_TYPE_WSS            0x5     // Wide Screen␣
↪Signal (line 23 PAL)
#define IVTV_SLICED_TYPE_VPS            0x7     // Video␣
↪Programming System (PAL) (line 16)
```

8.10.2 i.MX Video Capture Driver

Events

ipuX_csiY

This subdev can generate the following event when enabling the second IDMAC source pad:

- V4L2_EVENT_IMX_FRAME_INTERVAL_ERROR

The user application can subscribe to this event from the ipuX_csiY subdev node. This event is generated by the Frame Interval Monitor (see below for more on the FIM).

Controls

Frame Interval Monitor in ipuX_csiY

The adv718x decoders can occasionally send corrupt fields during NTSC/PAL signal re-sync (too little or too many video lines). When this happens, the IPU triggers a mechanism to re-establish vertical sync by adding 1 dummy line every frame, which causes a rolling effect from image to image, and can last a long time before a stable image is recovered. Or sometimes the mechanism doesn' t work at all, causing a permanent split image (one frame contains lines from two consecutive captured images).

From experiment it was found that during image rolling, the frame intervals (elapsed time between two EOF' s) drop below the nominal value for the current standard, by about one frame time (60 usec), and remain at that value until rolling stops.

While the reason for this observation isn't known (the IPU dummy line mechanism should show an increase in the intervals by 1 line time every frame, not a fixed value), we can use it to detect the corrupt fields using a frame interval monitor. If the FIM detects a bad frame interval, the ipuX_csiY subdev will send the event V4L2_EVENT_IMX_FRAME_INTERVAL_ERROR. Userland can register with the FIM event notification on the ipuX_csiY subdev device node. Userland can issue a streaming restart when this event is received to correct the rolling/split image.

The ipuX_csiY subdev includes custom controls to tweak some dials for FIM. If one of these controls is changed during streaming, the FIM will be reset and will continue at the new settings.

- V4L2_CID_IMX_FIM_ENABLE

Enable/disable the FIM.

- V4L2_CID_IMX_FIM_NUM

How many frame interval measurements to average before comparing against the nominal frame interval reported by the sensor. This can reduce noise caused by interrupt latency.

- V4L2_CID_IMX_FIM_TOLERANCE_MIN

If the averaged intervals fall outside nominal by this amount, in microseconds, the V4L2_EVENT_IMX_FRAME_INTERVAL_ERROR event is sent.

- V4L2_CID_IMX_FIM_TOLERANCE_MAX

If any intervals are higher than this value, those samples are discarded and do not enter into the average. This can be used to discard really high interval errors that might be due to interrupt latency from high system load.

- V4L2_CID_IMX_FIM_NUM_SKIP

How many frames to skip after a FIM reset or stream restart before FIM begins to average intervals.

- V4L2_CID_IMX_FIM_ICAP_CHANNEL / V4L2_CID_IMX_FIM_ICAP_EDGE

These controls will configure an input capture channel as the method for measuring frame intervals. This is superior to the default method of measuring frame intervals via EOF interrupt, since it is not subject to uncertainty errors introduced by interrupt latency.

Input capture requires hardware support. A VSYNC signal must be routed to one of the i.MX6 input capture channel pads.

V4L2_CID_IMX_FIM_ICAP_CHANNEL configures which i.MX6 input capture channel to use. This must be 0 or 1.

V4L2_CID_IMX_FIM_ICAP_EDGE configures which signal edge will trigger input capture events. By default the input capture method is disabled with a value of IRQ_TYPE_NONE. Set this control to IRQ_TYPE_EDGE_RISING, IRQ_TYPE_EDGE_FALLING, or IRQ_TYPE_EDGE_BOTH to enable input capture, triggered on the given signal edge(s).

When input capture is disabled, frame intervals will be measured via EOF interrupt.

File list

drivers/staging/media/imx/ include/media/imx.h include/linux/imx-media.h

Authors

- Steve Longerbeam <steve_longerbeam@mentor.com>
- Philipp Zabel <kernel@pengutronix.de>
- Russell King <linux@armlinux.org.uk>

8.10.3 Maxim Integrated MAX2175 RF to bits tuner driver

The MAX2175 driver implements the following driver-specific controls:

`V4L2_CID_MAX2175_I2S_ENABLE`

Enable/Disable I2S output of the tuner. This is a private control that can be accessed only using the subdev interface. Refer to Documentation/driver-api/media/v4l2-controls.rst for more details.

`(0)`	I2S output is disabled.
`(1)`	I2S output is enabled.

`V4L2_CID_MAX2175_HSLS`

The high-side/low-side (HSLS) control of the tuner for a given band.

`(0)`	The LO frequency position is below the desired frequency.
`(1)`	The LO frequency position is above the desired frequency.

`V4L2_CID_MAX2175_RX_MODE (menu)`

The Rx mode controls a number of preset parameters of the tuner like sample clock (sck), sampling rate etc. These multiple settings are provided under one single label called Rx mode in the datasheet. The list below shows the supported modes with a brief description.

"Europe modes"	
"FM 1.2" (0)	This configures FM band with a sample rate of 0.512 million samples/sec with a 10.24 MHz sck.
"DAB 1.2" (1)	This configures VHF band with a sample rate of 2.048 million samples/sec with a 32.768 MHz sck.
"North America modes"	
"FM 1.0" (0)	This configures FM band with a sample rate of 0.7441875 million samples/sec with a 14.88375 MHz sck.
"DAB 1.2" (1)	This configures FM band with a sample rate of 0.372 million samples/sec with a 7.441875 MHz sck.

8.10.4 Vaio Picturebook Motion Eye Camera Driver

Private API

The driver supports frame grabbing with the video4linux API, so all video4linux tools (like xawtv) should work with this driver.

Besides the video4linux interface, the driver has a private interface for accessing the Motion Eye extended parameters (camera sharpness, agc, video framerate), the snapshot and the MJPEG capture facilities.

This interface consists of several ioctls (prototypes and structures can be found in include/linux/meye.h):

MEYEIOC_G_PARAMS and MEYEIOC_S_PARAMS
: Get and set the extended parameters of the motion eye camera. The user should always query the current parameters with MEYEIOC_G_PARAMS, change what he likes and then issue the MEYEIOC_S_PARAMS call (checking for -EINVAL). The extended parameters are described by the meye_params structure.

MEYEIOC_QBUF_CAPT
: Queue a buffer for capture (the buffers must have been obtained with a VIDIOCGMBUF call and mmap'ed by the application). The argument to MEYEIOC_QBUF_CAPT is the buffer number to queue (or -1 to end capture). The first call to MEYEIOC_QBUF_CAPT starts the streaming capture.

MEYEIOC_SYNC
: Takes as an argument the buffer number you want to sync. This ioctl blocks until the buffer is filled and ready for the application to use. It returns the buffer size.

MEYEIOC_STILLCAPT and MEYEIOC_STILLJCAPT
: Takes a snapshot in an uncompressed or compressed jpeg format. This ioctl blocks until the snapshot is done and returns (for jpeg snapshot) the size of the image. The image data is available from the first mmap'ed buffer.

Look at the 'motioneye' application code for an actual example.

8.10.5 OMAP 3 Image Signal Processor (ISP) driver

Contacts: Laurent Pinchart <laurent.pinchart@ideasonboard.com>, Sakari Ailus <sakari.ailus@iki.fi>, David Cohen <dacohen@gmail.com>

Events

The OMAP 3 ISP driver does support the V4L2 event interface on CCDC and statistics (AEWB, AF and histogram) subdevs.

The CCDC subdev produces V4L2_EVENT_FRAME_SYNC type event on HS_VS interrupt which is used to signal frame start. Earlier version of this driver used V4L2_EVENT_OMAP3ISP_HS_VS for this purpose. The event is triggered exactly when the reception of the first line of the frame starts in the CCDC module. The event can be subscribed on the CCDC subdev.

(When using parallel interface one must pay account to correct configuration of the VS signal polarity. This is automatically correct when using the serial receivers.)

Each of the statistics subdevs is able to produce events. An event is generated whenever a statistics buffer can be dequeued by a user space application using the VIDIOC_OMAP3ISP_STAT_REQ IOCTL. The events available are:

- V4L2_EVENT_OMAP3ISP_AEWB
- V4L2_EVENT_OMAP3ISP_AF
- V4L2_EVENT_OMAP3ISP_HIST

The type of the event data is struct omap3isp_stat_event_status for these ioctls. If there is an error calculating the statistics, there will be an event as usual, but no related statistics buffer. In this case omap3isp_stat_event_status.buf_err is set to non-zero.

Private IOCTLs

The OMAP 3 ISP driver supports standard V4L2 IOCTLs and controls where possible and practical. Much of the functions provided by the ISP, however, does not fall under the standard IOCTLs —gamma tables and configuration of statistics collection are examples of such.

In general, there is a private ioctl for configuring each of the blocks containing hardware-dependent functions.

The following private IOCTLs are supported:

- VIDIOC_OMAP3ISP_CCDC_CFG
- VIDIOC_OMAP3ISP_PRV_CFG

- VIDIOC_OMAP3ISP_AEWB_CFG
- VIDIOC_OMAP3ISP_HIST_CFG
- VIDIOC_OMAP3ISP_AF_CFG
- VIDIOC_OMAP3ISP_STAT_REQ
- VIDIOC_OMAP3ISP_STAT_EN

The parameter structures used by these ioctls are described in include/linux/omap3isp.h. The detailed functions of the ISP itself related to a given ISP block is described in the Technical Reference Manuals (TRMs) —see the end of the document for those.

While it is possible to use the ISP driver without any use of these private IOCTLs it is not possible to obtain optimal image quality this way. The AEWB, AF and histogram modules cannot be used without configuring them using the appropriate private IOCTLs.

CCDC and preview block IOCTLs

The VIDIOC_OMAP3ISP_CCDC_CFG and VIDIOC_OMAP3ISP_PRV_CFG IOCTLs are used to configure, enable and disable functions in the CCDC and preview blocks, respectively. Both IOCTLs control several functions in the blocks they control. VIDIOC_OMAP3ISP_CCDC_CFG IOCTL accepts a pointer to struct omap3isp_ccdc_update_config as its argument. Similarly VIDIOC_OMAP3ISP_PRV_CFG accepts a pointer to struct omap3isp_prev_update_config. The definition of both structures is available in[1].

The update field in the structures tells whether to update the configuration for the specific function and the flag tells whether to enable or disable the function.

The update and flag bit masks accept the following values. Each separate functions in the CCDC and preview blocks is associated with a flag (either disable or enable; part of the flag field in the structure) and a pointer to configuration data for the function.

Valid values for the update and flag fields are listed here for VIDIOC_OMAP3ISP_CCDC_CFG. Values may be or’ ed to configure more than one function in the same IOCTL call.

- OMAP3ISP_CCDC_ALAW
- OMAP3ISP_CCDC_LPF
- OMAP3ISP_CCDC_BLCLAMP
- OMAP3ISP_CCDC_BCOMP
- OMAP3ISP_CCDC_FPC
- OMAP3ISP_CCDC_CULL
- OMAP3ISP_CCDC_CONFIG_LSC
- OMAP3ISP_CCDC_TBL_LSC

[1] include/linux/omap3isp.h

The corresponding values for the VIDIOC_OMAP3ISP_PRV_CFG are here:

- OMAP3ISP_PREV_LUMAENH
- OMAP3ISP_PREV_INVALAW
- OMAP3ISP_PREV_HRZ_MED
- OMAP3ISP_PREV_CFA
- OMAP3ISP_PREV_CHROMA_SUPP
- OMAP3ISP_PREV_WB
- OMAP3ISP_PREV_BLKADJ
- OMAP3ISP_PREV_RGB2RGB
- OMAP3ISP_PREV_COLOR_CONV
- OMAP3ISP_PREV_YC_LIMIT
- OMAP3ISP_PREV_DEFECT_COR
- OMAP3ISP_PREV_GAMMABYPASS
- OMAP3ISP_PREV_DRK_FRM_CAPTURE
- OMAP3ISP_PREV_DRK_FRM_SUBTRACT
- OMAP3ISP_PREV_LENS_SHADING
- OMAP3ISP_PREV_NF
- OMAP3ISP_PREV_GAMMA

The associated configuration pointer for the function may not be NULL when enabling the function. When disabling a function the configuration pointer is ignored.

Statistic blocks IOCTLs

The statistics subdevs do offer more dynamic configuration options than the other subdevs. They can be enabled, disable and reconfigured when the pipeline is in streaming state.

The statistics blocks always get the input image data from the CCDC (as the histogram memory read isn't implemented). The statistics are dequeueable by the user from the statistics subdev nodes using private IOCTLs.

The private IOCTLs offered by the AEWB, AF and histogram subdevs are heavily reflected by the register level interface offered by the ISP hardware. There are aspects that are purely related to the driver implementation and these are discussed next.

VIDIOC_OMAP3ISP_STAT_EN

This private IOCTL enables/disables a statistic module. If this request is done before streaming, it will take effect as soon as the pipeline starts to stream. If the pipeline is already streaming, it will take effect as soon as the CCDC becomes idle.

VIDIOC_OMAP3ISP_AEWB_CFG, VIDIOC_OMAP3ISP_HIST_CFG and VIDIOC_OMAP3ISP_AF_CFG

Those IOCTLs are used to configure the modules. They require user applications to have an in-depth knowledge of the hardware. Most of the fields explanation can be found on OMAP's TRMs. The two following fields common to all the above configure private IOCTLs require explanation for better understanding as they are not part of the TRM.

omap3isp_[h3a_af/h3a_aewb/hist]_config.buf_size:

The modules handle their buffers internally. The necessary buffer size for the module's data output depends on the requested configuration. Although the driver supports reconfiguration while streaming, it does not support a reconfiguration which requires bigger buffer size than what is already internally allocated if the module is enabled. It will return -EBUSY on this case. In order to avoid such condition, either disable/reconfigure/enable the module or request the necessary buffer size during the first configuration while the module is disabled.

The internal buffer size allocation considers the requested configuration's minimum buffer size and the value set on buf_size field. If buf_size field is out of [minimum, maximum] buffer size range, it's clamped to fit in there. The driver then selects the biggest value. The corrected buf_size value is written back to user application.

omap3isp_[h3a_af/h3a_aewb/hist]_config.config_counter:

As the configuration doesn't take effect synchronously to the request, the driver must provide a way to track this information to provide more accurate data. After a configuration is requested, the config_counter returned to user space application will be an unique value associated to that request. When user application receives an event for buffer availability or when a new buffer is requested, this config_counter is used to match a buffer data and a configuration.

VIDIOC_OMAP3ISP_STAT_REQ

Send to user space the oldest data available in the internal buffer queue and discards such buffer afterwards. The field omap3isp_stat_data.frame_number matches with the video buffer's field_count.

References

8.10.6 The Linux USB Video Class (UVC) driver

This file documents some driver-specific aspects of the UVC driver, such as driver-specific ioctls and implementation notes.

Questions and remarks can be sent to the Linux UVC development mailing list at linux-uvc-devel@lists.berlios.de.

Extension Unit (XU) support

Introduction

The UVC specification allows for vendor-specific extensions through extension units (XUs). The Linux UVC driver supports extension unit controls (XU controls) through two separate mechanisms:

- through mappings of XU controls to V4L2 controls
- through a driver-specific ioctl interface

The first one allows generic V4L2 applications to use XU controls by mapping certain XU controls onto V4L2 controls, which then show up during ordinary control enumeration.

The second mechanism requires uvcvideo-specific knowledge for the application to access XU controls but exposes the entire UVC XU concept to user space for maximum flexibility.

Both mechanisms complement each other and are described in more detail below.

Control mappings

The UVC driver provides an API for user space applications to define so-called control mappings at runtime. These allow for individual XU controls or byte ranges thereof to be mapped to new V4L2 controls. Such controls appear and function exactly like normal V4L2 controls (i.e. the stock controls, such as brightness, contrast, etc.). However, reading or writing of such a V4L2 controls triggers a read or write of the associated XU control.

The ioctl used to create these control mappings is called UVCIOC_CTRL_MAP. Previous driver versions (before 0.2.0) required another ioctl to be used beforehand (UVCIOC_CTRL_ADD) to pass XU control information to the UVC driver. This is no longer necessary as newer uvcvideo versions query the information directly from the device.

For details on the UVCIOC_CTRL_MAP ioctl please refer to the section titled "IOCTL reference" below.

3. Driver specific XU control interface

For applications that need to access XU controls directly, e.g. for testing purposes, firmware upload, or accessing binary controls, a second mechanism to access XU controls is provided in the form of a driver-specific ioctl, namely UVCIOC_CTRL_QUERY.

A call to this ioctl allows applications to send queries to the UVC driver that directly map to the low-level UVC control requests.

In order to make such a request the UVC unit ID of the control's extension unit and the control selector need to be known. This information either needs to be hardcoded in the application or queried using other ways such as by parsing the UVC descriptor or, if available, using the media controller API to enumerate a device's entities.

Unless the control size is already known it is necessary to first make a UVC_GET_LEN requests in order to be able to allocate a sufficiently large buffer and set the buffer size to the correct value. Similarly, to find out whether UVC_GET_CUR or UVC_SET_CUR are valid requests for a given control, a UVC_GET_INFO request should be made. The bits 0 (GET supported) and 1 (SET supported) of the resulting byte indicate which requests are valid.

With the addition of the UVCIOC_CTRL_QUERY ioctl the UVCIOC_CTRL_GET and UVCIOC_CTRL_SET ioctls have become obsolete since their functionality is a subset of the former ioctl. For the time being they are still supported but application developers are encouraged to use UVCIOC_CTRL_QUERY instead.

For details on the UVCIOC_CTRL_QUERY ioctl please refer to the section titled "IOCTL reference" below.

Security

The API doesn't currently provide a fine-grained access control facility. The UVCIOC_CTRL_ADD and UVCIOC_CTRL_MAP ioctls require super user permissions.

Suggestions on how to improve this are welcome.

Debugging

In order to debug problems related to XU controls or controls in general it is recommended to enable the UVC_TRACE_CONTROL bit in the module parameter 'trace' . This causes extra output to be written into the system log.

IOCTL reference

UVCIOC_CTRL_MAP - Map a UVC control to a V4L2 control

Argument: struct uvc_xu_control_mapping

Description:

> This ioctl creates a mapping between a UVC control or part of a UVC control and a V4L2 control. Once mappings are defined, userspace applica-

tions can access vendor-defined UVC control through the V4L2 control API.

To create a mapping, applications fill the uvc_xu_control_mapping structure with information about an existing UVC control defined with UVCIOC_CTRL_ADD and a new V4L2 control.

A UVC control can be mapped to several V4L2 controls. For instance, a UVC pan/tilt control could be mapped to separate pan and tilt V4L2 controls. The UVC control is divided into non overlapping fields using the 'size' and 'offset' fields and are then independently mapped to V4L2 control.

For signed integer V4L2 controls the data_type field should be set to UVC_CTRL_DATA_TYPE_SIGNED. Other values are currently ignored.

Return value:

On success 0 is returned. On error -1 is returned and errno is set appropriately.

ENOMEM
Not enough memory to perform the operation.

EPERM
Insufficient privileges (super user privileges are required).

EINVAL
No such UVC control.

EOVERFLOW
The requested offset and size would overflow the UVC control.

EEXIST
Mapping already exists.

Data types:

```
* struct uvc_xu_control_mapping

__u32   id              V4L2 control identifier
__u8    name[32]        V4L2 control name
__u8    entity[16]      UVC extension unit GUID
__u8    selector        UVC control selector
__u8    size            V4L2 control size (in bits)
__u8    offset          V4L2 control offset (in bits)
enum v4l2_ctrl_type
        v4l2_type       V4L2 control type
enum uvc_control_data_type
        data_type       UVC control data type
struct uvc_menu_info
        *menu_info      Array of menu entries (for menu controls␣
↪only)
__u32   menu_count      Number of menu entries (for menu controls␣
↪only)
```

(continues on next page)

(continued from previous page)

```
* struct uvc_menu_info

__u32   value           Menu entry value used by the device
__u8    name[32]        Menu entry name


* enum uvc_control_data_type

UVC_CTRL_DATA_TYPE_RAW          Raw control (byte array)
UVC_CTRL_DATA_TYPE_SIGNED       Signed integer
UVC_CTRL_DATA_TYPE_UNSIGNED     Unsigned integer
UVC_CTRL_DATA_TYPE_BOOLEAN      Boolean
UVC_CTRL_DATA_TYPE_ENUM         Enumeration
UVC_CTRL_DATA_TYPE_BITMASK      Bitmask
```

UVCIOC_CTRL_QUERY - Query a UVC XU control

Argument: struct uvc_xu_control_query

Description:

This ioctl queries a UVC XU control identified by its extension unit ID and control selector.

There are a number of different queries available that closely correspond to the low-level control requests described in the UVC specification. These requests are:

UVC_GET_CUR
: Obtain the current value of the control.

UVC_GET_MIN
: Obtain the minimum value of the control.

UVC_GET_MAX
: Obtain the maximum value of the control.

UVC_GET_DEF
: Obtain the default value of the control.

UVC_GET_RES
: Query the resolution of the control, i.e. the step size of the allowed control values.

UVC_GET_LEN
: Query the size of the control in bytes.

UVC_GET_INFO
: Query the control information bitmap, which indicates whether get/set requests are supported.

UVC_SET_CUR
: Update the value of the control.

Applications must set the 'size' field to the correct length for the control. Exceptions are the UVC_GET_LEN and UVC_GET_INFO queries, for which the size must be set to 2 and 1, respectively. The 'data' field must point to a valid writable buffer big enough to hold the indicated number of data bytes.

Data is copied directly from the device without any driver-side processing. Applications are responsible for data buffer formatting, including little-endian/big-endian conversion. This is particularly important for the result of the UVC_GET_LEN requests, which is always returned as a little-endian 16-bit integer by the device.

Return value:

On success 0 is returned. On error -1 is returned and errno is set appropriately.

ENOENT
The device does not support the given control or the specified extension unit could not be found.

ENOBUFS
The specified buffer size is incorrect (too big or too small).

EINVAL
An invalid request code was passed.

EBADRQC
The given request is not supported by the given control.

EFAULT
The data pointer references an inaccessible memory area.

Data types:

```
* struct uvc_xu_control_query

__u8    unit            Extension unit ID
__u8    selector        Control selector
__u8    query           Request code to send to the device
__u16   size            Control data size (in bytes)
__u8    *data           Control value
```