Linux DVB API Version 4

http://www.linuxdvb.org

v 0.3

April 15, 2005

Written by
Michael Hunold <hunold@linuxtv.org>

Contents

1 Introduction

DVB is the abbreviation for "Digital Video Broadcasting" and is an industry project managed by the Digital Video Broadcasting Project.

It's an an industry-led consortium of broadcasters, manufacturers, network operators, software developers, regulatory bodies and others that are interesting in standards for the delivery of any digitized informations to the home.

LinuxTV is a vendor independent, non-profit Linux project that works on a standardized Linux DVB API since 2000. The Linux DVB API Version 3 is included in the 2.6 kernel series and is very popular on PC systems mostly in Europe and Australia.

It's used by lots of open-source projects and various commercial set-top-boxes (STB) on different hardware platforms.

Unfortunately, the Linux DVB API Version 3 has some design flaws that make it uncomfortable to use on embedded systems and set-top-boxes. Some of the hardware capabilities of modern chipsets cannot be used to the full extend and memory and processing power are wasted unnecessarily.

The Linux DVB API Version 4 honours the developments on the field of modern DVB chipsets and solves the existing problem by defining a complete new API. Porting old applications is fairly easy because the v3 API is a complete subset of the new v4 API.

It's inevitable to have some knowledge in the area of digital video broadcasting (DVB) and at least part I of the MPEG2 specification ISO/IEC 13818 (aka ITU-T H.222) to understand the Linux DVB API Version 4.

Most of the DVB standards documents are available for free from http://www.dvb.org or http://www.etsi.org.

DVB is based on MPEG2 transport streams, just like ATSC (USA) and ISDB (Japan). In theory, Linux DVB API Version 4 can easily be extended to cover these standards, too, but so far nobody has cared enough to provide any proposals.

1.1 Goals

The Linux DVB API Version 4 doesn't want to be a complete multimeda framework. Graphics output and sophisticated video scaler handling is handled best by DirectFB. There is no support for arbitrary multimedia data that the hardware cannot process directly. The

Linux DVB API Version 4 doesn't have support for auxillary hardware that is found in typical STBs or IDTVs, like smartcard interfaces.

The Linux DVB API Version 4 is a hardware independent, kernel level only driver framework to control digital TV hardware easily and efficiently

The idea is to make the life of both software and hardware developers easier and provide a consistent abstraction layer for different hardware.

Software developers can support different hardware platforms easier and make their applications truely hardware independent. The hardware vendors can provide support for their existing products easier and can provide a smooth transition from one chipset generation to the next.

1.2 Related technologies

"IP-over-DVB" uses techniques like Multi Protocol Encapsulation (MPE) or Ultra Light Encapsulation (ULE) to put IP packets into MPEG2 transport stream packets. The existing DVB infrastructure is used to provide a high bandwitdh network downstream.

"DVB-over-IP" puts MPEG2 transport stream packages into IP packages and uses existing IP infrastructure to transport DVB data. There is currently only an ETSI draft standard available, so currently RTP is used most of the time to ensure low-latency transmisssion.

Of course it's possible to put nearly everthing into MPEG2 transport stream packets. For example hardware vendors can provide a System Software Update (SSU) for their products.

Because of the fact that most hardware can playback MPEG2 program streams and MPEG1 data streams, at least the hardware can theoretically support DVD playback.

1.3 History

In 1998 the Technotrend GmbH develops the still very popular PC DVB card with a full-featured STB processor on it. In 1999 Siemens produces a card based on the Technotrend design and supports the development of the first Linux driver as a diploma thesis.

In 2000 Nokia develops a DVB API and approaches Convergence GmbH to implement this API for the Siemens card. At the same time, the community project LinuxTV is launched to promote the new API, which is sponsored by Convergence until mid 2004.

Nokia shortly after terminates it's efforts and the API is then heavily modified to become more Linux specific. During that time many new drivers are added to the repository by developers from all around the world to support a variety of DVB hardware.

In 2001 the ongoing developments finally result in the Linux DVB API Version 3 which is included into the Linux kernel 2.5.44 in 2002.

In 2003 Convergence and Toshiba Electronics Europe GmbH start the development of the Linux DVB API Version 4 with public discussion of the API features on the linux-dvb mailing list. The reason to create a complete new API was the fact that PCs and embedded platforms are diverging. On PCs, only budgets cards are currently produced, which only provide the full raw transport stream and leave all decoding and processing up to the main CPU. On embedded platforms, however, data is multiplexed by specialized hardware or firmware for direct application use which relieves the main CPU from these tasks. Because of the fact that there is no new "full-featured" PC DVB card in sight, the Linux DVB API Version 4 heads towards highly-integrated embedded STB and Integrated Digital TV (IDTV) systems.

In 2004 the Linux DVB API Version 4 is nearly fully specified and the generic DVB modules and a sample driver for the Siemens card is available.

Today, the LinuxTV project is a community project by DVB enthusiasts and developers interested in Digital TV. It's open, independent and non-profit and hosted on independent servers.

2 Design

The Linux DVB API Version 4 is a means to control digital tv hardware easily and efficiently. It's designed to support PCI/USB DVB extension cards, dedicated set-top-box (STB) chipsets and integrated digital TV (IDTV) solutions. It's a hardware independent driver framework that is available as a kernel level programming interface.

2.1 Present situation

Although the Linux DVB API Version 3 is widespread, in use by applications and well-known to the programmers, it's inevitable to establish a new API to circument some of the major problems the Linux DVB API Version 3 has.

First of all, PCs and embedded platforms are divering. For PCs, new cards are only available as "budget" cards, which means that they only provide the full, raw, unmodified TS to the system and put the burden of handling the data to the main CPU.

On embedded platforms, however, dedicated STB/IDTV chipsets demultiplex the data for direct application use and specialized hardware or firmware on DSPs relieves the main CPU greatly.

There is a new challenge with supporting embedded platforms running Linux and the Linux DVB API Version 4 heads towards highly-integrated embedded STB and IDTV systems.

2.2 Linux DVB API Version 3 problems

The Linux DVB API Version 3 was focussed on the popular Siemens PCI DVB card. Due to the pragmatic evolution of the API, there are namespace inconsitencies and inconsitent remains of things that really don't belong into the API, like ad-hoc DVD subtitle support or a very limited OSD API design.

There is a superfluous internal DVB kernel layer, because the initial idea was to have the possibility to provide a socket based interface to the DVB core in the future, which of course never happened.

The Linux DVB API Version 3 has very limited support for modern hardware. There is no explicit support for multiple frontends, video and audio decoders and no possibility to make explicit source-sink connections. Current implementations are highly hardware dependent.

There is no support for important features like special recording hardware and event logging facilities, that are provided by modern hardware.

The main drawnback of the Linux DVB API Version 3, however, is that all data transfers go through memory ringbuffers, which means that there is no support for zero-copy DMA. On PCs this does not matter much, but on embedded platforms, this is a major burden to the main CPU.

Because of the architectual problems of the core, the inconsitency of the API and the lack of zero-copy DMA it's not possible to simply extend the existing API. A complete new design is inevitable.

2.3 Linux DVB API Version 3 vs. Version 4

From the userspace perspective, there are not many differences between the Linux DVB API Version 4 and the Linux DVB API Version 3. Both are using a Linux/Posix character device interface under the /dev/dvb/adapter... tree. They use the standard Unix system calls like `open()`, `read()` and `ioctl()` to achieve cirtain action on the device. Most userspace programs can be easily ported to the Linux DVB API Version 4, because the core features that have been in the Linux DVB API Version 4 were only slightly changed. To use the new features like zero-copy DMA via the `mmap()` system call, however, bigger changes are necessary.

3 Concepts

Speaking of "concepts" is perhaps a little bit too much, because the design philosophy is basically the UNIX design philosophy.

Nevertheless it's necessary for application developers to know these concepts, because they apply to all xxxx

for everybody to get to know some of the concepts that are going to be used,

3.1 Control concept

foo

3.2 Capability concept

Because of the fact that different hardware has different capabilities, the Linux DVB API Version 4 provides a standardized way to query the capabilties of every device. Please note that sometimes even different devices of the same kind have different capabilities, because the hardware designers decided to make not every device equal. For every device the `DVB_xxx_GET_CAPS` ioctl (replace xxx with the appropriate device type like demux) can be used to query the specific device capabilties.

3.3 Connection concept

It's common that the hardware has multiple devices of the same kind, e.g. multiple frontends, demuxes and even multiple audio and video decoders. The Linux DVB API Version 4 provides a way to build a "filter chain" of devices using the `DVB_xxx_SET_SOURCE` ioctl. This is common for all processing, decoding and output devices. In general, you can open a device with write permissions only once. This device open is used to control the connection state of the device.

3.4 Status concept

The Linux DVB API Version 4 provides a common concept for getting to know the current status of a device and getting informed of changes in the device status. This concept is currently supported by the demux, audio and video devices on every read-only open.

If you have opened the device in blocking mode, then `DVB_xxx_GET_STATUS` will block until the device status changes. It's possible to mask out the interesting events with the `status` bitfield member of the parameter structure. For example, if you're only interested in the play state change of a audio device you simple set `status` to `DVB_AUDIO_PLAY_STATE`. Other status changes of the device will not wake up the sleep. Please note that the first call to the ioctl returns immediately to provide an initial device status.

If you have opened the device in non-blocking mode, then `DVB_xxx_GET_STATUS` will return immediately and provide the current device status to you. The `status` member of the return struct will indicate which members of the status struct have changed since the last call of the ioctl. If you set the `status` bitfield member of the ioctl parameter structure before calling the `ioctl()` system call, then the bitfield will only indicate the device changes you have specified. Please note that all other members of the structure will be updated regardlessly.

In any case you use the `poll()` system call to put the current process to sleep until there is a change in the device status.

4 Frontend API

A frontend is the combination of a tuner and an analog to digital demodulator that retrieves the digital data stream back from ananlog transmission via air or cable.

In case of a satellite frontend, there is support for legacy satellite equipment control (SEC) as well as Eutelsat's more sophisticated DiSEqC protocl for controlling peripheral satellite equipment.

The raw transport stream is internally forwarded to the demux for further stream processing.

4.1 Device informations

foo.

This enum describes the type of the frontend.

```
enum dvb_fe_type {
  DVB_FE_DVB_S = (1 << 0), /*!< DVB-S frontend with QPSK modulation */
  DVB_FE_DVB_C = (1 << 1), /*!< DVB-C frontend with QAM modulation */
  DVB_FE_DVB_T = (1 << 2), /*!< DVB-T frontend with OFDM modulation */
};
```

This enum describes the available forward error correction code rates.

```
enum dvb_fe_code_rate {
  DVB_FE_FEC_NONE = (1 << 0),
  DVB_FE_FEC_1_2  = (1 << 1),
  DVB_FE_FEC_2_3  = (1 << 2),
  DVB_FE_FEC_3_4  = (1 << 3),
  DVB_FE_FEC_4_5  = (1 << 4),
  DVB_FE_FEC_5_6  = (1 << 5),
  DVB_FE_FEC_6_7  = (1 << 6),
  DVB_FE_FEC_7_8  = (1 << 7),
  DVB_FE_FEC_8_9  = (1 << 8),
  DVB_FE_FEC_AUTO = (1 << 9),
};
```

This enum describes the available modulation types.

```
enum dvb_fe_modulation {
  DVB_FE_QPSK     = (1 << 0),
  DVB_FE_QAM_16   = (1 << 1),
  DVB_FE_QAM_32   = (1 << 2),
  DVB_FE_QAM_64   = (1 << 3),
  DVB_FE_QAM_128  = (1 << 4),
  DVB_FE_QAM_256  = (1 << 5),
  DVB_FE_QAM_AUTO = (1 << 6),
};
```

This enum describes common supported frontend capabilities.

```
enum dvb_fe_common_cap {
  DVB_FE_CAN_INVERSION_AUTO   = (1 << 0), /*!< fixme */
  DVB_FE_CAN_RECOVER          = (1 << 1), /*!< frontend can recover from a cable unplug automatically */
  DVB_FE_CAN_MUTE_TS          = (1 << 2), /*!< frontend can stop spurious TS data output */
  DVB_FE_SUP_HIGH_LNB_VOLTAGE = (1 << 3), /*!< fixme, frontend can deliver higher lnb voltages */
};
```

This enum describes other available, frontend type dependent capabilities.

```
enum dvb_fe_other_cap {
  DVB_FE_CAN_TRANSMISSION_MODE_AUTO = (1 << 0), /*!< fixme (DVB-T specific) */
  DVB_FE_CAN_BANDWIDTH_AUTO         = (1 << 1), /*!< fixme (DVB-T specific) */
  DVB_FE_CAN_GUARD_INTERVAL_AUTO    = (1 << 2), /*!< fixme (DVB-T specific) */
  DVB_FE_CAN_HIERARCHY_AUTO         = (1 << 3), /*!< fixme (DVB-T specific) */
};
```

This struct is used to query general informations about the frontend.

```
struct dvb_frontend_info {
  char             name[128];             /*!< brand name */
  enum dvb_fe_type type;                  /*!< frontend type */
  uint32_t         frequency_min;         /*!< minium tuning frequency, fixme unit? */
  uint32_t         frequency_max;         /*!< maximum tuning frequency, fixme unit?  */
  uint32_t         frequency_stepsize;    /*!< fixme */
  uint32_t         frequency_tolerance;   /*!< fixme */
  uint32_t         symbol_rate_min;       /*!< minium tuning frequency, fixme unit? */
  uint32_t         symbol_rate_max;       /*!< maximum tuning frequency, fixme unit? */
  uint32_t         symbol_rate_tolerance; /*!< in ppm, fixme */
  uint32_t         notifier_delay;        /*!< in ms, fixme */

  enum dvb_fe_code_rate  caps_fec;        /*!< supported fecs */
  enum dvb_fe_modulation caps_modulation; /*!< supported modulations */
  enum dvb_fe_common_cap caps_common;     /*!< supported common capabilities */
  enum dvb_fe_other_cap  caps_other;      /*!< supported other capabilities*/
};
```

This I/O-Control retrieves basic informations about a frontend device. It can be used on any device open and always returns 0.

```
#define DVB_FE_GET_INFO               _IOR(DVB_IOCTL_BASE,  0x00, struct dvb_frontend_info)
```

4.2 Satellite equipment control (SEC) commands

Before Eutelsat introduced its DiSEqC protocol to control peripheral phsatellite equipment, primitve means called satellite equipment control (SEC) commands were used to control the connected LNB.

The polarization of the LNB was selected by providing two different voltages to the LNB, the different bands were selected by a 22kHz tone, that was modulated onto the cable.

This enum describes the available voltage settings.

```
enum dvb_sec_voltage {
  DVB_SEC_VOLTAGE_13,
  DVB_SEC_VOLTAGE_18,
  DVB_SEC_VOLTAGE_OFF
};
```

This I/O-Control sets the desired voltage in order to switch between polarizations or turns off the voltage supply for the LNB.

```
#define DVB_FE_SEC_SET_VOLTAGE        _IOW(DVB_IOCTL_BASE, 0x06, enum dvb_sec_voltage)
```

Return codes:

- EOPNOTSUPP: frontend does not support setting the voltage
- ETIMEDOUT: communication with frontend failed

This enum describes the available tone settings.

```
enum dvb_sec_tone_mode {
  DVB_SEC_TONE_ON,
  DVB_SEC_TONE_OFF
};
```

This I/O-Control sets the desired tone setting in order to switch between the low and high band.

```
#define DVB_FE_SEC_SET_TONE           _IOW(DVB_IOCTL_BASE, 0x05, enum dvb_sec_tone_mode)
```

Return codes:

- EINVAL: the parameter is neither DVB_SEC_TONE_ON nor DVB_SEC_TONE_OFF
- EOPNOTSUPP: frontend does not support setting the voltage
- ETIMEDOUT: communication with frontend failed

This enum describes the available burst settings.

```
enum dvb_sec_tone_burst {
  DVB_SEC_BURST_A,
  DVB_SEC_BURST_B
};
```

This I/O-Control fixme.

```
#define DVB_FE_SEC_SEND_BURST         _IOW(DVB_IOCTL_BASE, 0x04, enum dvb_sec_tone_burst)
```

Return codes:

- EFIXME:

This I/O-Control enables or disables high lnb voltage.

If your LNB is connected through a quite long cable, the voltage might suffer due to the long distance. Most frontends support to output a higher lnb voltage, which is usually about 0.5 volts above the desired level.

```
#define DVB_FE_ENABLE_HIGH_LNB_VOLTAGE _IOW(DVB_IOCTL_BASE, 0x07, unsigned int)
```

Return codes:

- EOPNOTSUPP: frontend does not support setting the voltage

4.3 DiSEqC commands

In xxxx Eutelsat introduced the DiSEqC protocl for controlling peripheral satellite equipment. The first versions were designed as a replacement for the SEC commands, in later versions real bi-directional communication between a master and slaves is possible.

Please check out the DiSEqC bus spec available on http://www.eutelsat.org/ for further informations.

This I/O-Control fixme.

```
#define DVB_FE_DISEQC_RESET_OVERLOAD   _IOW(DVB_IOCTL_BASE, 0x01, unsigned int)
```

Return codes:

- EFIXME:

This struct is used for sending a DiSEqC message. Please refer to the DiSEqC bus spec available on http://www.eutelsat.org/ for further informations.

```
struct dvb_diseqc_master_cmd {
  uint8_t msg [6];  /*!< { framing, address, command, data [3] } */
  uint8_t msg_len;  /*!< valid values are 3...6  */
};
```

This I/O-Control sends a DiSEqC message.

```
#define DVB_FE_DISEQC_SEND_MASTER_CMD  _IOW(DVB_IOCTL_BASE,  0x02, struct dvb_diseqc_master_cmd)
```

This struct is used for for receiving a DiSEqC reply. Please refer to the DiSEqC bus spec available on http://www.eutelsat.org/ for further informations.

```
struct dvb_diseqc_slave_reply {
  uint8_t msg [4];  /*!< { framing, data [3] } */
  uint8_t msg_len;  /*!< valid values are 0...4, 0 means no msg  */
  int     timeout;  /*!< return from ioctl after timeout ms with  errorcode when no message was received */
};
```

This I/O-Control receives a DiSEqC message from a slave.

```
#define DVB_FE_DISEQC_RECV_SLAVE_REPLY _IOR(DVB_IOCTL_BASE,  0x03, struct dvb_diseqc_slave_reply)
```

4.4 Frontend status

This enum describes the current frontend status.

```
enum dvb_fe_status {
  DVB_FE_HAS_SIGNAL     = (1 << 0),   /*!< found something above the noise level */
  DVB_FE_HAS_CARRIER    = (1 << 1),   /*!< found a DVB signal  */
  DVB_FE_HAS_VITERBI    = (1 << 2),   /*!< FEC is stable  */
  DVB_FE_HAS_SYNC       = (1 << 3),   /*!< found sync bytes  */
  DVB_FE_HAS_LOCK       = (1 << 4),   /*!< everything's working */
  DVB_FE_TIMEDOUT       = (1 << 5),   /*!< no lock within the last ~2 seconds */
  DVB_FE_REINIT         = (1 << 6),   /*!< frontend was reinitialized, application is recommended to reset DiSEqC, tone
};
```

This I/O-Control retrieves the current frontend status.

```
#define DVB_FE_READ_STATUS             _IOR(DVB_IOCTL_BASE,  0x08, enum dvb_fe_status)
```

This enum describes some frontend statistical informations, if available by the frontend.

```
enum dvb_fe_statistics_avail {
  DVB_FE_BER                = (1 << 0), /*!< bit error rate */
  DVB_FE_SIGNAL_STRENGTH    = (1 << 1), /*!< signal strength */
  DVB_FE_SNR                = (1 << 2), /*!< signal to noise ratio */
  DVB_FE_UNCORRECTED_BLOCKS = (1 << 3), /*!< number of uncorrected blocks */
};
```

This struct is used to retrive statistical informations from the frontend.

```
struct dvb_fe_statistics {
  enum dvb_fe_statistics_avail avail; /*!< describes the available informations */
  uint32_t ber;                       /*!< bit error rate, if available */
  uint16_t signal_strength;           /*!< signal strength, if available */
  uint16_t snr;                       /*!< signal to noise ratio, if available */
  uint32_t uncorrected_blocks;        /*!< number of uncorrected blocks, if available */
};
```

This I/O-Control retrives statistical informations from the frontend. the informations are reset in the frontend after the data has been retrieved.

```
#define DVB_FE_READ_STATIISTICS        _IOR(DVB_IOCTL_BASE,  0x09, struct dvb_fe_statistics)
```

4.5 Configuration and tuning

This enum describes the spectral inversion setting of the frontend.

```
enum dvb_fe_spectral_inversion {
  DVB_FE_INVERSION_OFF,
  DVB_FE_INVERSION_ON,
  DVB_FE_INVERSION_AUTO
};
```

This struct tuning parameters for DVB-S frontends.

```
struct dvb_dvb_s_parameters {
  uint32_t               symbol_rate;  /*!< symbol rate in Symbols per second */
  enum dvb_fe_code_rate  fec_inner;    /*!< forward error correction (see above) */
};
```

This struct tuning parameters for DVB-C frontends.

```
struct dvb_dvb_c_parameters {
  uint32_t                symbol_rate; /*!< symbol rate in Symbols per second */
  enum dvb_fe_code_rate   fec_inner;   /*!< forward error correction (see above) */
  enum dvb_fe_modulation  modulation;  /*!< modulation type (see above) */
};
```

This enum fixme.

```
enum dvb_fe_bandwidth {
  DVB_BANDWIDTH_8_MHZ,
  DVB_BANDWIDTH_7_MHZ,
  DVB_BANDWIDTH_6_MHZ,
  DVB_BANDWIDTH_AUTO
};
```

This enum fixme.

```
enum dvb_fe_transmit_mode {
  DVB_TRANSMISSION_MODE_2K,
  DVB_TRANSMISSION_MODE_8K,
  DVB_TRANSMISSION_MODE_AUTO
};
```

This enum fixme.

```
enum dvb_fe_guard_interval {
  DVB_GUARD_INTERVAL_1_32,
  DVB_GUARD_INTERVAL_1_16,
  DVB_GUARD_INTERVAL_1_8,
  DVB_GUARD_INTERVAL_1_4,
  DVB_GUARD_INTERVAL_AUTO
};
```

This enum fixme.

```
enum dvb_fe_hierarchy {
  DVB_HIERARCHY_NONE,
  DVB_HIERARCHY_1,
  DVB_HIERARCHY_2,
  DVB_HIERARCHY_4,
  DVB_HIERARCHY_AUTO
};
```

This struct tuning parameters for DVB-T frontends.

```
struct dvb_dvb_t_parameters {
  enum dvb_fe_bandwidth      bandwidth;              /*!< fixme */
  enum dvb_fe_code_rate      code_rate_HP;           /*!< high priority stream code rate */
  enum dvb_fe_code_rate      code_rate_LP;           /*!< low priority stream code rate */
  enum dvb_fe_modulation     constellation;          /*!< modulation type (see above) */
  enum dvb_fe_transmit_mode  transmission_mode;      /*!< fixme */
  enum dvb_fe_guard_interval guard_interval;         /*!< fixme */
  enum dvb_fe_hierarchy      hierarchy_information;  /*!< fixme */
};
```

This struct is used for tuning a frontend.

```
struct dvb_frontend_parameters {
  uint32_t frequency;                       /*!< frequency in 100 Hz (absolute for DVB-C/DVB-T, intermediate DVB-S) */
  enum dvb_fe_spectral_inversion inversion;  /*!< spectral inversion setting */
  union {
    struct dvb_dvb_s_parameters dvb_s; /*!< if frontend is DVB-S */
    struct dvb_dvb_c_parameters dvb_c; /*!< if frontend is DVB-C */
    struct dvb_dvb_t_parameters dvb_t; /*!< if frontend is DVB-T */
  } u; /*!< tuning parameters */
};
```

This I/O-Control tunes a frontend using the specified tuning parameters.

```
#define DVB_FE_SET_FRONTEND           _IOW(DVB_IOCTL_BASE,  0x0d, struct dvb_frontend_parameters)
```

This I/O-Control retrieves the current tuning parameters fromt the frontend.

```
#define DVB_FE_GET_FRONTEND           _IOR(DVB_IOCTL_BASE,  0x0e, struct dvb_frontend_parameters)
```

4.6 Event handling

This struct describes a frontend event.

```
struct dvb_frontend_event {
  enum dvb_fe_status status;                /*!< bitfield */
  struct dvb_frontend_parameters parameters; /*!< tuning parameters at the time the event happened */
};
```

This I/O-Control retrieves the latest tuning event from the frontend, blocks if device wasn't opened with O_NONBLOCK.

```
#define DVB_FE_GET_EVENT              _IOR(DVB_IOCTL_BASE,  0x0f, struct dvb_frontend_event)
```

5 Memory input API

A memory input can be used to provide raw data stream (TS, PS or PES) from userspace using memory mapping thus allowing zero-copy DMA if the hardware supports it. The data is usually internally directly routed to a demux device.

5.1 Device informations

This struct describes general informations about a memory input.

```
struct dvb_memory_info {
  char name[128];                /*!< descriptive device name */
  enum dvb_source_format formats; /*!< supported formats */
};
```

This I/O-Control retrieves basic informations about a memory input. It never fails and always returns 0.

```
#define DVB_MEMORY_GET_INFO _IOR(DVB_IOCTL_BASE, 0x80, struct dvb_memory_info)
```

5.2 Configuration

Before a memory input is able to deliver data to the demux, it's necessary to configure the memory input appropriately.

This struct describes a configuration of a memory input.

```
struct dvb_memory_configuration {
  /* in */
  enum dvb_source_format format; /*!< chosen data format */
  size_t  size;                  /*!< desired size of buffer */
  size_t  threshold;             /*!< notification threshold */

  /* out */
  size_t  mmap_size;   /*!< size of memory area to mmap() */
  size_t  mmap_offset; /*!< offset into memory for data */
};
```

This I/O-Control configures a memory input.

```
#define DVB_MEMORY_SET_CONFIGURATION _IOWR(DVB_IOCTL_BASE, 0x81, struct dvb_memory_configuration)
```

Return codes:

- `EINVAL`: size too small or threshold is bigger than size

- E2BIG: size exceeds DVB_MAX_BUFFER_SIZE
- EBUSY: memory input has been configured before
- ENOMEM: out of memory while allocating buffer

5.3 Data input

After a configuration has been set for the memory input, the userapplication needs to map the input buffer to the process space using the mmap() system call.

Idealy, it then has direct access to some hardware buffer and can provide data efficently. Syncronisation with the hardware is achieved with the following two ioctls.

This struct describes a free section of the input buffer.

```
struct dvb_memory_data {
  size_t offset; /*!< offset of confirmed data into buffer */
  size_t len;    /*!< length of confirmed data */
};
```

This I/O-Control retrieves informations about a data area, where new data can be put.

```
#define DVB_MEMORY_RETRIEVE_DATA_AREA _IOR (DVB_IOCTL_BASE, 0x83, struct dvb_memory_data)
```

Return codes:

- EFAULT: memory configuration not set before
- EBUSY: request pending
- EIO: memory has not been mmap()ed

This I/O-Control confirms the specified amount of bytes to the memory input for further processing.

```
#define DVB_MEMORY_CONFIRM_DATA_AREA  _IOWR(DVB_IOCTL_BASE, 0x84, size_t)
```

Return codes:

- EBUSY: no request pending
- EINVAL: size argument is 0 or invalid. maximum possible size is returned.

6 Demux API

In short, a demux provides switching facilites between a frontend or a memory input and existing hardware MPEG video and audio decoders and can usually extract specific portions from the stream and provides data capture to system memory.

Within the scope of the LinuxDVB API a demux is a logical device with exactly one input stream and a number of output streams.

If the hardware can process multiple input streams in parallel, each one will be represented by one logical demux device. Often the hardware has an input router (e.g. can simultaneously demux two streams out of a selection of five), so the input for each demux device must be set via DVB_DEMUX_SET_SOURCE.

Data from each input stream can be selected by a number of filters, which either route their output to connected hardware decoders or output to memory for processing by the application. For stream recording different filter types are provided to make use of special recording hardware.

Stream recording hardware may support searching for start codes in MPEG video streams, and can generate events when they are found. This is used for bulding index files for TS recording.

Most hardware also supports a STC notification facility for synchronization purposes.

Before any filtering can be done, the device input has to be configured on with a separate device open using the `DVB_DMX_SET_SOURCE` ioctl to accept data from a frontend or a memory input.

It depends on the capabilities of the input device and of the demux, which filters are available and how much can be set afterwards.

The different possible filters are:

1. single MPEG-2 TS PID filter
2. MPEG-2 PSI / DVB SI section filters
3. MPEG-2 PS / MPEG-1 system stream / multiplexed PES filters
4. recording filters based on MPEG-2 TS PID filter
5. direct feeding to a hardware decoder device (decoder feed)

Data from the first four filter types is usally written to userspace and processed there. For the fifth filter type, routing of data to hardware decoders is done by passing the file descriptor of the filter to the decoder device's `SET_SOURCE` ioctl.

Any demux device can only opened once for writing (ie. O_WRONLY); use that open to control the input routing via DVB_DMX_SET_SOURCE. All filtering opens must be O_RDONLY. Opens using the O_RDWR permission are not allowed.

In short, a demux provides switching facilites between a frontend or a memory input and existing hardware MPEG video and audio decoders and can usually extract specific portions from the stream and provides data capture to system memory.

Within the scope of the LinuxDVB API a demux is a logical device with exactly one input stream and a number of output streams.

If the hardware can process multiple input streams in parallel, each one will be represented by one logical demux device. Often the hardware has an input router (e.g. can simultaneously demux two streams out of a selection of five), so the input for each demux device must be set via DVB_DEMUX_SET_SOURCE.

Data from each input stream can be selected by a number of filters, which either route their output to connected hardware decoders or output to memory for processing by the application. For stream recording different filter types are provided to make use of special recording hardware.

Stream recording hardware may support searching for start codes in MPEG video streams, and can generate events when they are found. This is used for bulding index files for TS recording.

Most hardware also supports a STC notification facility for synchronization purposes.

6.1 Capabilities

It's not unusual that different demuxes on the same hardware have different filtering capabilities on the input data and (e.g. only demux2 can accept MPEG-2 PS). User application should carefully query all existing demuxes for their capabilities.

Also, the filter numbers are maximum values since some hardware shares filters between demux channels, and decoder/payload filters may use a PID filter internally.

This enum describes the different capabilities that may be supported by the demux device.

```
enum dvb_demux_capability {
  DVB_DEMUX_CAP_SOURCE_FORMATS,         /*!< bitfield, source formats the demux can handle (\ref dvb_source_format) */
  DVB_DEMUX_CAP_NUM_PES_FILTERS,        /*!< integer, number of available PES filters (\ref DVB_DEMUX_SET_PES_FILTER) *
  DVB_DEMUX_CAP_NUM_AUDIO_FILTERS,      /*!< integer, number of available audio filters (\ref DVB_DEMUX_SET_TS_DECODER_
  DVB_DEMUX_CAP_NUM_VIDEO_FILTERS,      /*!< integer, number of available video filters (\ref DVB_DEMUX_SET_TS_DECODER_
  DVB_DEMUX_CAP_NUM_PCR_FILTERS,        /*!< integer, number of available pcr filters (\ref DVB_DEMUX_SET_TS_DECODER_FI
  DVB_DEMUX_CAP_NUM_SECTION_FILTERS,    /*!< integer, number of available section filters (\ref DVB_DEMUX_SET_SECTION_I
  DVB_DEMUX_CAP_NUM_PID_FILTERS,        /*!< integer, number of available pid filters (\ref DVB_DEMUX_SET_PID_FILTER)*/
  DVB_DEMUX_CAP_PID_FILTER_FLAGS,       /*!< bitfield, supported flags for pid filters (\ref dvb_demux_pid_filter_flags
  DVB_DEMUX_CAP_NUM_RECORDING_FILTERS,  /*!< integer, number of available recording filters (\ref DVB_DEMUX_SET_RECORDI
  DVB_DEMUX_CAP_RECORDING_EVENTS,       /*!< bitfield, supported recording events by recording pid filters (\ref dvb_de
  DVB_DEMUX_CAP_RECORDING_TYPES,        /*!< bitfield, available recording types (\ref dvb_demux_recording_type) */
  DVB_DEMUX_CAP_NUM_DESCR_KEY_PAIRS,    /*!< integer, number of available descrambling key pairs (fixme, add ref)*/
```

```
};
```

This struct is used to query the capabilities of a demux device.

```
struct dvb_demux_caps {
  enum dvb_demux_capability cap; /*!< capability to query  */
  unsigned int            val;   /*!< output value */
};
```

This I/O-Control queries one specific demux capability. A demux device is expected to support quering *all* capabilities mentioned above (ie. return 0 if a capability is not supported by the hardware.

```
#define DVB_DEMUX_GET_CAPS _IOWR(DVB_IOCTL_BASE, 0x20, struct dvb_demux_caps)
```

Return codes:

- EINVAL: the capability is unknown

6.2 Device input setup

Configuring the input of a demux is only allowed if the device was opened with write permissions (ie. using O_WRONLY). Each logical demux device can only be opened once with write permissions.

This I/O-Control connects the demux to an already opened frontend or memory input through the filedescriptor.

There is no way to select other sources like ASI, LVDS or directly connected firewire inputs. These inputs must be faked using a so-called dummy frontend.

```
#define DVB_DEMUX_SET_SOURCE _IOW(DVB_IOCTL_BASE, 0x21, int /* input device fd */)
```

Return codes:

- EBADF: demux file descriptor was not opened for writing
- EINVAL: the filedescriptor doesn't belong to any DVB device
- ENOSYS: the input device doesn't belong to the same adapter as the demux

6.3 MPEG-2 TS filters

MPEG-2 TS PID filters filter a TS on the PID value only. Many DVB hardwares support three varieties of PID filters:

1. decoder feeds: data is delivered internally to the decoder
2. general purpose data filters: single PID output to memory
3. stream recording filters: output of multiple PIDs to one common memory

6.3.1 TS decoder feeds

Most demuxes can be configured to directly (ie. internally in the hardware) deliver specific TS packets to specified hardware decoding facilities (ie. video or audio decoders). The demux file descriptor can then be passed to the decoder's SET_SOURCE ioctl, so the decoder actually gets the TS packets.

This I/O-Control sets a decoder feed filter on this demux open to deliver specific TS packets to the decoder (which needs to be already connected to the demux open)

```
#define DVB_DEMUX_SET_TS_DECODER_FEED _IOW(DVB_IOCTL_BASE, 0x23, uint16_t /* pid */)
```

Return codes:

- EBUSY: another filter has already been set
- ENODEV: the demux device doesn't have any decoder feeds
- EINVAL: the decoder_type parameter is invalid

6.3.2 Pid filters

This filter types filters a single PID to a memory buffer, which can be retrieved by the read() systemcall. O_NONBLOCK opens and the poll() systemcall are fully supported.

The buffer_threshold is used to limit irq load and must be ¡= buffer_size (both members are specified in bytes).

This enum describes all possible capabilities of a pid filter. Some of the options are mutually exclusive, for example DVB_DMX_PAYLOAD_ONLY and DVB_DMX_ADAPTATION_ONLY.

```
enum dvb_demux_pid_filter_flags {
  DVB_DEMUX_FULL_TS         = (1 << 0), /*!< don't filter on a specific pid, output the whole TS */
  DVB_DEMUX_PAYLOAD_ONLY    = (1 << 1), /*!< only deliver the payload (ie. strip off the TS header) */
  DVB_DEMUX_ADAPTATION_ONLY = (1 << 2), /*!< only deliver the TS header and any adaptation fields if present */
  DVB_DEMUX_WAIT_FOR_PUSI   = (1 << 3), /*!< wait for the payload unit start indicator before starting to filter */
  DVB_DEMUX_HIGH_PRIO_ONLY  = (1 << 4), /*!< only deliver high priority packets on the specified pid */
  DVB_DEMUX_LOW_PRIO_ONLY   = (1 << 5), /*!< only deliver low priority packets on the specified pid*/
  DVB_DEMUX_OUTPUT_DUPES    = (1 << 6), /*!< deliver duplicated packets, too (if the hardware delivers them at all) */
  DVB_DEMUX_OUTPUT_ERRPKTS  = (1 << 7), /*!< deliver erroneous packets, too (if the hardware delivers them at all) */
};
```

This struct is used to configure a PID filter. The member sbuffer_size and buffer_threshold are hints to the driver; the real buffer size and threshold might be different due to hardware restriction and are reported back by the driver.

```
struct dvb_demux_pid_filter {
  uint16_t pid;                          /*!< PID to filter (unless DVB_DEMUX_FULL_TS is specified for the flags) */
  enum dvb_demux_pid_filter_flags flags; /*!< special filtering flags */
  uint32_t buffer_size;                  /*!< in bytes, size of internal buffer */
  uint32_t buffer_threshold;             /*!< in bytes, notify threshold, must be <= buffer_size */
};
```

This I/O-Control sets a simple pid filter on the demux open, which delivers all TS packets with matching pid to a memory buffer

If the DVB_DMX_FULL_TS flag is specified the pid value and the other flags are irrelevant and the full TS is output.

```
#define DVB_DEMUX_SET_PID_FILTER _IOWR(DVB_IOCTL_BASE, 0x24, struct dvb_demux_pid_filter)
```

Return codes:

- EBUSY: another filter has already been set
- ENODEV: the demux device doesn't have any pid filters
- ENOSYS: the demux doesn't support the requested dvb_demux_pid_filter_flags
- EINVAL: either buffer_size < 188 or buffer_threshold > buffer_size
- E2BIG: the buffer size exceeds DVB_MAX_BUFFER_SIZE

6.3.3 Recording filters

The output of multiple PIDs goes to a common memory buffer. The recording for a number of PIDs can be started and stopped in one atomic operation.

If event logging is specified, the specified events that might be triggered by the different pids that are being recorded are written to a separate memory buffer for futher processing by the application, for example to build an index file for the recording.

This enum describes the desired type of a recording filter. DVB_DMX_REC_TYPE_SIMPLE_READ and DVB_DMX_REC_TYPE_EVENT_LOGGING are mutually exclusive. Specifying DVB_DMX_REC_TY without DVB_DMX_REC_TYPE_BUFFERED is not allowed obviously.

```
enum dvb_demux_recording_type {
  DVB_DEMUX_REC_TYPE_SIMPLE_READ   = (1 << 0), /*!< simple \c read() based recording, no event logging possible */
  DVB_DEMUX_REC_TYPE_BUFFERED      = (1 << 1), /*!< buffer based recording */
  DVB_DEMUX_REC_TYPE_EVENT_LOGGING = (1 << 2), /*!< event logging for buffer based recording */
};
```

This struct is used to set a recording filter on a demux device open.

buffer_size and buffer_threshold are hints to the driver; the real buffer size and threshold can differ because of hardware restriction and are written back by the driver.

The members mmap_size and data_offset are returned by the driver and are only valid if DVB_DMX_REC_TYPE_BUFFERED was specified.

log_offset is returned by the driver and is only valid if event logging was requested by specifying DVB_DMX_REC_TYPE_EVENT_LOGGING.

```
struct dvb_demux_recording_filter {
  /* in */
  enum dvb_demux_recording_type type; /*!< type of this recording filter */
  size_t  buffer_size;                /*!< in bytes, size of the buffer to allocate */
  size_t  buffer_threshold;           /*!< in bytes, notify threshold, must be $<=$ buffer_size */
  /* out, only valid for type $!=$ DVB_DEMUX_REC_TYPE_SIMPLE_READ */
  size_t  mmap_size;                  /*!< in bytes, size of memory area to mmap() */
  size_t  data_offset;                /*!< in bytes, offset into mmap()ed memory for data buffer */
  size_t  log_offset;                 /*!< in bytes, offset into mmap()ed memory for event logging buffer */
};
```

This enum describes all possible recording events which might be written to the event log if specified.

```
enum dvb_demux_rec_event {
  DVB_DEMUX_REC_EVENT_NONE                  = 0,         /*!< none of the events below */
  DVB_DEMUX_REC_EVENT_PUSI                  = (1 <<  0), /*!< the payload unit start indicator is set */
  DVB_DEMUX_REC_EVENT_TEI                   = (1 <<  1), /*!< a discontinuity in the PCR has occured */
  DVB_DEMUX_REC_EVENT_DISCONT_INDICATOR     = (1 <<  2), /*!< a discontinuity in the PCR has occured */
  DVB_DEMUX_REC_EVENT_RANDOM_ACCESS         = (1 <<  3), /*!< this TS packet is a valid location from which to decode
  DVB_DEMUX_REC_EVENT_ESPI                  = (1 <<  4), /*!< this is a high priority packet for this PID stream */
  DVB_DEMUX_REC_EVENT_PCR                   = (1 <<  5), /*!< this packet contains a pcr */
  DVB_DEMUX_REC_EVENT_OPCR                  = (1 <<  6), /*!< this packet contains an opcr */
  DVB_DEMUX_REC_EVENT_TRANSPORT_PRIVATE_DATA = (1 <<  7), /*!< this packet contains private data in the adaptation fiel
  DVB_DEMUX_REC_EVENT_AF_EXTENSION          = (1 <<  8), /*!< an adaptation field extension exists within this TS pacl
  DVB_DEMUX_REC_EVENT_SEQ_HEADER            = (1 <<  9), /*!< this packet contains the sequence header code 0xb3 */
  DVB_DEMUX_REC_EVENT_GROUP_START           = (1 << 10), /*!< this packet contains the group start code 0xb8 */
  DVB_DEMUX_REC_EVENT_I_FRAME               = (1 << 11), /*!< this packet contains the beginning of an i-frame */
  DVB_DEMUX_REC_EVENT_P_FRAME               = (1 << 12), /*!< this packet contains the beginning of an p-frame */
  DVB_DEMUX_REC_EVENT_B_FRAME               = (1 << 13), /*!< this packet contains the beginning of an n-frame */
  DVB_DEMUX_REC_EVENT_ALL                   = 0x3fff,    /*!< all of the events above */
};
```

This struct is used to specify one recording pid and the desired events it should trigger in the event log. If DVB_DEMUX_REC_EVENT_ALL is specified for flags, the driver will generate all flags supported (see DVB_DMX_GET_CAPS).

```
struct dvb_rec_pid {
  uint16_t                pid;   /*!< pid to capture */
  enum dvb_demux_rec_event flags; /*!< desired events this item should trigger */
};
```

This struct is used to specify n_pids pids for recording at once.

```
struct dvb_demux_recording_pids {
  uint32_t           n_pids; /*!< number of pids specified */
  struct dvb_rec_pid *pids;  /*!< array of \ref dvb_rec_pid */
};
```

This I/O-Control sets a recording filter on the demux open.

```
#define DVB_DEMUX_SET_RECORDING_FILTER  _IOWR(DVB_IOCTL_BASE, 0x25, struct dvb_demux_recording_filter)
```

Return codes:

- `EBUSY`: another filter has already been set
- `ENODEV`: the demux device doesn't have any recording filters
- `ENOSYS`: the demux doesn't support the requested combination in dvb_demux_recording_type
- `EINVAL`: either buffer_size < PAGE_SIZE or buffer_threshold > buffer_size
- `E2BIG`: the buffer size exceeds DVB_MAX_BUFFER_SIZE

This I/O-Control adds the recording pids specified by dvb_demux_recording_pids to the active recording filter. The operation is atomic, ie. all pids are added or none.

```
#define DVB_DEMUX_ADD_RECORDING_PIDS  _IOW (DVB_IOCTL_BASE, 0x26, struct dvb_demux_recording_pids)
```

Return codes:

- EBUSY: no recording filter has been set
- EINVAL: the number of pids is 0
- EEXIST: one of the specified pids has already been added
- ENOSYS: the demux doesn't support the requested combination in ??vb_demux_rec_event

This I/O-Control deletes the recording pids specified by dvb_demux_recording_pids from the active recording filter. The operation is atomic, ie. no pids are deleted or all.

```
#define DVB_DEMUX_DEL_RECORDING_PIDS  _IOW (DVB_IOCTL_BASE, 0x27, struct dvb_demux_recording_pids)
```

Return codes:

- EBUSY: no recording filter has been set
- EINVAL: the number of pids is 0
- ENOENT: one of the specified pids wasn't set at all

This struct is used as a return structure when recording data is retrieved with the DVB_DEMUX_RETRIEV ioctl.

```
struct dvb_demux_recording_data {
  size_t data_offset; /*!< in bytes, offset of confirmed data into buffer */
  size_t data_len;    /*!< in bytes, length of confirmed data */

  size_t log_offset;  /*!< in bytes, offset of confirmed logging events data into buffer */
  size_t log_len;     /*!< in bytes, length of confirmed logging events data */
};
```

This struct describes one item in the event logging buffer.

```
struct dvb_demux_recording_log_item {
  uint64_t counter;               /*!< counter, is increased for every TS packet processed (wrap around)*/
  uint16_t pid;                   /*!< pid which triggered this entry in the event logging buffer */
  enum dvb_demux_rec_event event; /*!< event bitfield for the TS packet */
};
```

This I/O-Control retrieves informations about the area of the recording buffer where new recording data is available. If event logging was requested, informations about the area of the event logging buffer where the accompanying event loggging informations can be found are supplied, too.

The area won't be touched by the driver until it is confirmed with DVB_DEMUX_CONFIRM_RECORDING

```
#define DVB_DEMUX_RETRIEVE_RECORDING_DATA _IOWR(DVB_IOCTL_BASE, 0x28, struct dvb_demux_recording_data)
```

Return codes:

- EBUSY: no recording filter has been set
- EINVAL: ţead() based capture was requested

- ENOENT: informations were already retrieved, but haven't been confirmed yet
- EAGAIN: data in the buffer hasn't reached the threshold yet (Q_NONBLOCK opens only)
- EINTR: waiting for data interrupted by user (not Q_NONBLOCK opens)

This I/O-Control confirms the area of the recording buffer which has been retrieved with DVB_DEMUX_RETRIEVE_RECORDING_DATA. If event logging was requested, the corresponding area of the event logging buffer is confirmed, too.

```
#define DVB_DEMUX_CONFIRM_RECORDING_DATA  _IO  (DVB_IOCTL_BASE, 0x29)
```

Return codes:

- EBUSY: no recording filter has been set
- EINVAL: ţead() based capture was requested
- ENOENT: nothing to confirm

6.3.4 Section filters

Many varieties of MPEG-2 PSI or DVB SI filters are supported.

This I/O-Control describes the number of bytes a section filter can handle.

```
#define DVB_DEMUX_FILTER_SIZE 16
```

This enum describes all possible flags for a section filter.

```
enum dvb_demux_section_filter_flags {
  DVB_DEMUX_SECTION_CHECK_CRC = (1 << 0), /*!< only deliver sections where the CRC check succeeded */
  DVB_DEMUX_SECTION_ONESHOT   = (1 << 1), /*!< disable the section filter after one section has been delivered*/
};
```

This struct describes the properties of a section filter.

If all neg bits are zero, the filter matches when $((data\&mask) == filter$, else it matches when $(((data\&mask\& \ neg) == (filter\& \ neg))\&\&((data\&mask\&neg)! = (filter\&neg)))$ i.e. all non-masked data bits with neg bit 0 must match and at least one non-masked data bit with neg bit 1 must differ.

```
struct dvb_demux_section_filter {
  uint16_t  pid;                            /*!< pid to filter */
  uint8_t   filter[DVB_DEMUX_FILTER_SIZE];  /*!< bytes to match */
  uint8_t   mask[DVB_DEMUX_FILTER_SIZE];    /*!< filter mask */
  uint8_t   neg[DVB_DEMUX_FILTER_SIZE];     /*!< positive or negative match */
  uint32_t  timeout;                        /*!< timeout in milliseconds */
  enum dvb_demux_section_filter_flags flags; /*!< special flags*/
  uint32_t buffer_size;                     /*!< in bytes, size of internal buffer */
  uint32_t buffer_threshold;                /*!< in bytes, notify threshold, must be <= buffer_size */
};
```

This I/O-Control sets a section filter.

```
#define DVB_DEMUX_SET_SECTION_FILTER _IOW(DVB_IOCTL_BASE, 0x28, struct dvb_demux_section_filter)
```

Return codes:

- EBUSY: another filter has already been set
- ENODEV: the demux device doesn't have any section filters
- EINVAL: buffer_size < 4096
- E2BIG: the buffer size exceeds DVB_MAX_BUFFER_SIZE
- ENOSYS: the demux doesn't support the requested flags

6.4 MPEG-2 PS/PES filters

Most demuxes support MPEG-2 PS, MPEG-2 multiplexed PES as well as MPEG-1 system streams.

The PS filters come in two varieties:

1. (MPEG) decoder feeds: data is directly fed to the decoder
2. general purpose data filters with output to memory

Filtering is done primarily on the stream_id, but with DVB_DMX_PES_PRIVATE_1/2 filtering is done on the sub_stream_id, which can be used for DVD audio.

This is much like dvb_demux_ts_decoder_feed, except that DVB_DEMUX_DECODER_TYPE_PCR is not allowed here.

6.4.1 PES decoder feeds

This enum describes extended PES filtering flags.

```
enum dvb_demux_pes_flags {
  DVB_DEMUX_PES_PRIVATE_1 = (1 << 0), /*!< fixme */
  DVB_DEMUX_PES_PRIVATE_2 = (1 << 1), /*!< fixme */
};
```

This struct is used to configure the demux to deliver all PS/PES packets with the specified stream_id to the specified decoder feed (DVB_DEMUX_DECODER_TYPE_PCR is not allowed) If dvb_demux_pes_flags is specified, filtering is done on the corresponding substream id.

```
struct dvb_demux_pes_decoder_feed {
  uint8_t                    stream_id;     /*!< stream id */
  enum dvb_demux_pes_flags    flags;        /*!< indicates if substream_id of private stream should be filtered */
};
```

This I/O-Control sets a decoder feed filter on this demux open to deliver specific PS/PES packets to the specified hardware decoding facility.

```
#define DVB_DEMUX_SET_PES_DECODER_FEED _IOW(DVB_IOCTL_BASE, 0x29, struct dvb_demux_pes_decoder_feed)
```

Return codes:

- EBUSY: another filter has already been set
- ENODEV: the demux device doesn't have any decoder feeds
- EINVAL: the decoder_type parameter is invalid

6.4.2 PES filters

This struct is used configure a PES filter.

```
struct dvb_demux_pes_filter {
  uint8_t                  stream_id; /*!< stream_id to filter */
  enum dvb_demux_pes_flags flags;     /*!< indicates if substream_id of private stream should be filtered */

  uint32_t buffer_size;         /*!< in bytes, size of internal buffer */
  uint32_t buffer_threshold;    /*!< in bytes, notify threshold, must be <= buffer_size */
};
```

This I/O-Control sets a simple PES filter on the demux open, which delivers all PES packets with matching stream_id to a memory buffer.

fixme: return codes

```
#define DVB_DEMUX_SET_PES_FILTER _IOW(DVB_IOCTL_BASE, 0x2a, struct dvb_demux_pes_filter)
```

6.5 Synchronization

This struct describes the stc.

```
struct dvb_demux_stc {
  unsigned int base; /*!< divisor for stc to get 90kHz clock */
  uint64_t stc;      /*!< stc in base*90 kHz units */
};
```

This I/O-Control reads out the system clock recovered from the stream.

For MPEG-2 TS this ioctl must be called on a file descriptor where DVB_DEMUX_SET_TS_DECODER_FE has been called.

For MPEG-2 PS or MPEG-1 the STC is recovered from the SCR automatically, without additional filter, and this ioctl can be called on any demux file descriptor where any decoder filter has been set on the PS.

```
#define DVB_DEMUX_GET_STC _IOWR(DVB_IOCTL_BASE, 0x2e, struct dvb_demux_stc)
```

6.6 Descrambler control

HW usually has a number of key "slots", where a key pair is set. Then a number of PIDs can be descrambled using this key slot. Descrambling can operate at TS or PES packet level. The ioctls can be applied to any demux file descriptor (you can also open a new one).

This struct describes on descrambling key pair.

```
struct dvb_demux_descr_key {
  int     index;
  int     parity; /* 0 == even, 1 == odd */
  uint8_t key[8];
};
```

This I/O-Control sets a descrambling key pair.

```
#define DVB_DEMUX_SET_DESCR_KEY _IOW(DVB_IOCTL_BASE, 0x2f, struct dvb_demux_descr_key)
```

This struct describes the pid/slot combination to be descrambled.

```
struct dvb_demux_descr_pid {
  int      index;  /* -1 == disable */
  int      flags;
#define DVB_DEMUX_DESCR_PES (1 << 0)
  uint16_t pid;
};
```

This I/O-Control sets a descrambling pid.

```
#define DVB_DEMUX_SET_DESCR_PID _IOW(DVB_IOCTL_BASE, 0x30, struct dvb_demux_descr_pid)
```

6.7 Demux status

This I/O-Control is used to set a trigger for the specified stc. Only one trigger can be set at a time. If the filedescriptor has been opened in blocking mode, then the ioctl will sleep until the time has been reached.

```
#define DVB_DEMUX_SET_STC_TRIGGER _IOW(DVB_IOCTL_BASE, 0x2b, uint64_t /* 33-bit / 90kHz */)
```

This enum describes the possible demux status items.

```
enum dvb_demux_status {
  DVB_DEMUX_PRIVATE     = (1 << 0), /*!< some private data from the hw driver */
  DVB_DEMUX_STC_COMPARE = (1 << 1),  /*!< stc trigger fired */
  DVB_DEMUX_PS_TIMEOUT  = (1 << 2),  /*!< ps timeout happened */
  DVB_DEMUX_TS_SYNC     = (1 << 3),  /*!< ts sync status changed */
};
```

This struct is used to query the status of the demux.

```
struct dvb_demux_status_query {
  uint32_t priv[16];

  uint64_t stc_compare; /*!< stc value when event happened */
  uint64_t ps_timeout;  /*!< stc value when event happened */
  int      ts_sync;     /*!< if the demux has a valid ts sync */

  enum dvb_demux_status status;
};
```

This I/O-Control queries the demux status.

```
#define DVB_DEMUX_GET_STATUS _IOWR(DVB_IOCTL_BASE, 0x6b, struct dvb_demux_status_query)
```

7 Common interface API

We create one "ci" device node per CI controller, i.e. each "ci" device serves all slots of that controller.

The protocol units used by this API are raw, unfragmented TPDUs. I.e. the transport layer must be implemented in userspace, but link level fragmentation is handled entirely within the driver.

The API supports two protocols: ETSI EN 50221 PCMCIA link layer packets (LPDUs, i.e. fragmented TPDUs) with a maximum size of 64KiB defragemented (but otherwise unprocessed) TPDUs, i.e. the transport layer must be implemented in userspace, but link level fragmentation is handled entirely within the driver. Note: According to EN 50221 TPDUs can be of any size, so defragmentation cannot be implemented inside the driver in a standard conforming way. However, this protocol is proven to be useful in practise.

The poll() systemcall can be used in the following way: changes in slot status will be signaled by POLLPRI (module inserted / ready) or POLLHUP (module removed) available space in the send queue is signaled by POLLOUT available data from module is signaled by POLLIN other errors are signaled by POLLERR (this usually means the slot needs to be reset)

7.1 Capabilites

This enum describes the possible capabilities of a controller.

```
enum dvb_ci_capability {
  DVB_CI_CAP_PROTOCOL,    /*!< supported protocols */
  DVB_CI_CAP_NUM_SLOTS,   /*!< number of slots */
  DVB_CI_CAP_MAX_TPDU_SIZE, /*!< maximum TPDU size for DVB_CI_PROTOCOL_LINK_DEFRAG */
};
```

This enum describes the available protocols.

```
enum dvb_ci_protocol {
  DVB_CI_PROTOCOL_LINK,   /*!< EN 50221 PCMCIA link layer */
  DVB_CI_PROTOCOL_LINK_DEFRAG /*!< defragmented links layer packets */
};
```

This struct is used to query the capabilities of a controller.

```
struct dvb_ci_caps {
  enum dvb_ci_capability cap; /*!< capabiltiy to query*/
  unsigned int           val; /*!< result */
};
```

This I/O-Control queries a specific capability of a controller.

```
#define DVB_CI_GET_CAPS _IOWR(DVB_IOCTL_BASE, 0xc0, struct dvb_ci_caps)
```

7.2 CI slot handling

This I/O-Control resets a slot.

```
#define DVB_CI_RESET_SLOT _IOW(DVB_IOCTL_BASE, 0xc1, int /* slot number */)
```

This enum describes the ci slot status the device might be in.

```
enum dvb_ci_cam_status {
  DVB_CI_CAM_PRESENT = (1 << 0),  /*!< CAM inserted */
  DVB_CI_CAM_READY   = (1 << 1),  /*!< CAM is initialized */
  DVB_CI_CAM_ERROR   = (1 << 2),  /*!< communication with CAM not possible */
};
```

This struct is used to query the current slot status.

```
struct dvb_ci_slot_status {
        int     slot;   /*!< slot number to query */
        enum dvb_ci_cam_status  status;   /*!< current status */
  unsigned int    fragment_size;  /*!< negotiated link level fragment size */
};
```

This I/O-Control queries the current slot status.

```
#define DVB_CI_GET_SLOT_STATUS _IOWR(DVB_IOCTL_BASE, 0xc2, struct dvb_ci_slot_status)
```

7.3 Message interface

Messages with the CAM are exchanged via the read() and write() systemcalls. For the DVB_CI_PROTOCOL_LINK_DEFRAG protocol, each message contains one complete, unfragmented TPDU in the following format:

```
struct {
u8 slot;  // slot number
u8 tc_id; // transport connection id
u8 tpdu[];
};
```

Each write() call must write exactly one complete message. If the message is larger than the value returned by DVB_CI_CAP_MAX_TPDU_SIZE, ENOBUFS is returned. Each read() call will return at maximum one complete message, even if there are more messages pending. If the buffer is too small to read the complete message, ENOBUFS is returned.

For the DVB_CI_PROTOCOL_LINK protocol, each message contains one complete LPDU (containing one TPDU fragment) in the following format:

```
struct {
u8 slot;   // slot number
u8 lpdu[];
};
```

Each write() call must write exactly one complete message. If the message is larger than the fragment_size returned by DVB_CI_GET_SLOT_STATUS ENOBUFS is returned. Each read() call will return at maximum one complete message, even if there are more messages pending. If the buffer is too small to read the complete message, ENOBUFS is returned.

8 Audio API

MPEG hardware audio decoders can be found on most set-top-box chipsets. They can either retrieve the audio data from the demux or they can be fed directly from userspace.

The audio API is split in separate devices for decoding + post-processing, This enum describes the available audio source formats.

```
enum dvb_audio_source_format {
  DVB_AUDIO_FORMAT_PES   = (1 << 0), /*!< MPEG-2 packetized elementary stream */
  DVB_AUDIO_FORMAT_MPEG1 = (1 << 1), /*!< MPEG-1 system stream */
  DVB_AUDIO_FORMAT_ES    = (1 << 2), /*!< MPEG-2 elementary stream */
  DVB_AUDIO_FORMAT_DVD   = (1 << 3), /*!< MPEG-2 PES with private header processing */
  DVB_AUDIO_FORMAT_RAW   = (1 << 4), /*!< RAW data */
};
```

This enum describes the available audio encodings.

```
enum dvb_audio_encoding {
  DVB_AUDIO_ENC_AUTO  = (1 << 0), /*!< the underlying mechanism figures out the real encoding scheme */
  DVB_AUDIO_ENC_PCM   = (1 << 1), /*!< PCM, fixme: pass-through to post-processing (?) */
  DVB_AUDIO_ENC_LPCM  = (1 << 2), /*!< LPCM */
  DVB_AUDIO_ENC_MPEG1 = (1 << 3), /*!< MPEG1 layer 1+2 */
  DVB_AUDIO_ENC_MPEG2 = (1 << 4), /*!< MPEG2 layer 1+2 */
  DVB_AUDIO_ENC_MP3   = (1 << 5), /*!< MPEG2 layer 3 */
  DVB_AUDIO_ENC_AC3   = (1 << 6), /*!< AC3 */
  DVB_AUDIO_ENC_DTS   = (1 << 7), /*!< DTS */
  DVB_AUDIO_ENC_AAC   = (1 << 8), /*!< AAC */
};
```

This enum describes the available audio capabilities.

```
enum dvb_audio_capability {
  DVB_AUDIO_CAP_SOURCE_FORMATS,   /*!< available source formats */
  DVB_AUDIO_CAP_ENCODINGS,        /*!< available encodings */
  DVB_AUDIO_CAP_NUM_SPDIF_INPUTS, /*!< available spdif inputs */
  DVB_AUDIO_CAP_NUM_I2S_INPUTS,   /*!< available i2c inputs */
};
```

This struct is used to query the MPEG audio decoder capabilities.

```
struct dvb_audio_caps {
  enum dvb_audio_capability cap; /*!< cabapility to query  */
  unsigned int val;              /*!< output value by the driver */
};
```

This I/O-Control requests a capability information from the driver. Drivers are expected to deliver valid informations for all capabilities defined.

```
#define DVB_AUDIO_GET_CAPS _IOWR(DVB_IOCTL_BASE, 0x60, struct dvb_audio_caps)
```

Return codes:

- `EINVAL`: the requested capability is invalid

8.1 Input routing and syncronisation

The decoder device needs to be opened with mode O_RDWR in order for DVB_AUDIO_SET_SOURCE to succeed. Each audio device can only be opened once with O_RDWR.

For DVB_AUDIO_SOURCE_MEMORY the stream is passed into the audio device via write().

For DVB_AUDIO_SOURCE_DEMUX the file descriptor of the demux has to be passed in. This enum describes the source type of the audio data.

```
enum dvb_audio_source_type {
  DVB_AUDIO_SOURCE_DEMUX,  /*!< pass through the corresponding demux device */
  DVB_AUDIO_SOURCE_MEMORY, /*!< directly into the decoder via \c write() system call */
  DVB_AUDIO_SOURCE_I2S,    /*!< from an external i2c interface */
  DVB_AUDIO_SOURCE_SPDIF,  /*!< from an external spdif interface */
};
```

This struct describes the audio source of the audio data.

```
struct dvb_audio_source {
  enum dvb_audio_source_format format; /*!< input source type */
  enum dvb_audio_source_type type;     /*!< desired source format to be delivered (DVB_AUDIO_SOURCE_MEMORY only) */
  enum dvb_audio_encoding enc;         /*!< audio encoding scheme */
  int input;                           /*!< demux fd, or source id (for I2S or SPDIF) */
};
```

This I/O-Control

- either prepares the decoder to accept audio data through the write() system call
- or configures a connection between a demux device, an i2c or an spdif interface and the decoder.

```
#define DVB_AUDIO_SET_SOURCE _IOW(DVB_IOCTL_BASE, 0x61, struct dvb_audio_source)
```

Return codes:

- `EINVAL`: input source type is unknown.
- `EFIXME`: desired source format cannot be delivered by the demux.

This I/O-Control binds the syncronisation to a demux specified by a filedescriptor, where a filter of type DVB_DEMUX_FILTER_TYPE_PCR has been set or, for PS playback, the demux file descriptor where the audio decoder filter has been set.

```
#define DVB_AUDIO_SET_REF_STC _IOW(DVB_IOCTL_BASE, 0x62, int /* demux fd */)
```

This I/O-Control controls the audio syncronisation of the decoder.

```
#define DVB_AUDIO_SET_SYNC _IOW(DVB_IOCTL_BASE, 0x63, int /* 0 == unsynced, != 0 sync to STC */ )
```

8.2 Decoder control

This I/O-Control starts playback immediately.

```
#define DVB_AUDIO_START _IO(DVB_IOCTL_BASE, 0x64)
```

This I/O-Control stops playback immediately.

```
#define DVB_AUDIO_STOP _IO(DVB_IOCTL_BASE, 0x65)
```

This I/O-Control clear decoder buffers (necessary when stream input is not continuous, e.g. seeking in stream or reverse playback)

```
#define DVB_AUDIO_CLEAR_BUFFER _IO(DVB_IOCTL_BASE, 0x66)
```

This enum describes the available audio decode channels.

```
enum dvb_audio_decode_channel {
  DVB_AUDIO_STEREO,
  DVB_AUDIO_MONO,
  DVB_AUDIO_DUAL_LEFT,
  DVB_AUDIO_DUAL_RIGHT,
};
```

This I/O-Control sets the channel output mode (not available for 5.1ch decoding)

```
#define DVB_AUDIO_SET_CHANNEL _IOW(DVB_IOCTL_BASE, 0x68, enum dvb_audio_decode_channel)
```

This enum describes the available 5.1 channel decoding mode.

```
enum dvb_audio_dec_mode {
  DVB_AUDIO_DEC_MODE_STEREO,
  DVB_AUDIO_DEC_MODE_CENTRE,
  DVB_AUDIO_DEC_MODE_LCR,
  DVB_AUDIO_DEC_MODE_LRS,
  DVB_AUDIO_DEC_MODE_LCRS,
  DVB_AUDIO_DEC_MODE_LRSLSR,
  DVB_AUDIO_DEC_MODE_LCRSLSR,
};
```

This I/O-Control sets the channel decoding mode (not available for stereo/mono/dual) (fixme?)

```
#define DVB_AUDIO_SET_DECODE_MODE _IOW(DVB_IOCTL_BASE, 0x69, enum dvb_audio_dec_mode)
```

This enum describes the available audio karaoke modes, fixme: what's this?.

```
enum dvb_audio_karaoke_mode {
  DVB_AUDIO_KARAOKE_OFF             = 0,
  DVB_AUDIO_KARAOKE_ON              = (1 << 0),
  DVB_AUDIO_KARAOKE_2_0             = 0,        /* left/right */
  DVB_AUDIO_KARAOKE_3_0             = (1 << 1), /* left, middle/melody, right */
  DVB_AUDIO_KARAOKE_NO_VOCALS       = 0,
  DVB_AUDIO_KARAOKE_VOCALS_1        = (1 << 2),
  DVB_AUDIO_KARAOKE_VOCALS_2        = (2 << 2),
  DVB_AUDIO_KARAOKE_VOCALS_BOTH     = (3 << 2),
};
```

This I/O-Control sets the audio karaoke mode

```
#define DVB_AUDIO_SET_KARAOKE_MODE _IOW(DVB_IOCTL_BASE, 0x6a, enum dvb_audio_karaoke_mode)
```

This enum describes the possible sampling frequencies.

```
enum dvb_audio_sampling_frequency {
  DVB_AUDIO_FREQUENCY_8000,    /*!< 8 KHz */
  DVB_AUDIO_FREQUENCY_11025,   /*!< 11.025 KHz */
  DVB_AUDIO_FREQUENCY_12000,   /*!< 12 KHz */
  DVB_AUDIO_FREQUENCY_16000,   /*!< 16 KHz */
  DVB_AUDIO_FREQUENCY_22050,   /*!< 22.05 KHz */
  DVB_AUDIO_FREQUENCY_24000,   /*!< 24.00 KHz */
  DVB_AUDIO_FREQUENCY_32000,   /*!< 32.00 KHz */
  DVB_AUDIO_FREQUENCY_44100,   /*!< 44.1 KHz */
  DVB_AUDIO_FREQUENCY_48000,   /*!< 48.00 KHz */
  DVB_AUDIO_FREQUENCY_64000,   /*!< 64.00 KHz */
  DVB_AUDIO_FREQUENCY_88200,   /*!< 88.2 KHz */
  DVB_AUDIO_FREQUENCY_96000,   /*!< 96.00 KHz */
  DVB_AUDIO_FREQUENCY_128000,  /*!< 128 KHz */
  DVB_AUDIO_FREQUENCY_176400,  /*!< 176.4 KHz */
  DVB_AUDIO_FREQUENCY_192000,  /*!< 192 KHz */
  DVB_AUDIO_FREQUENCY_INVALID, /*!< invalid value */
};
```

8.3 Raw PCM data

If source format DVB_AUDIO_FORMAT_RAW is used, then further set up is necessary for processing. They are sampling rate, resolution, signedness, endianness and number of channels

This enum describes the resolution (no of bits used to represent) of the PCM data.

```
enum dvb_audio_pcm_bits {
  DVB_AUDIO_PCM_BITS_16,
  DVB_AUDIO_PCM_BITS_18,
  DVB_AUDIO_PCM_BITS_20,
  DVB_AUDIO_PCM_BITS_24,
  DVB_AUDIO_PCM_BITS_32,
};
```

This enum describes the endianess.

```
enum dvb_audio_endianess {
  DVB_AUDIO_BIG_ENDIAN,
  DVB_AUDIO_LITTLE_ENDIAN,
};
```

This struct is used to provide informations about the raw PCM data.

```
struct dvb_audio_pcm_info {
  enum dvb_audio_sampling_frequency sampling_frequency; /*!< sampling frequency */
  enum dvb_audio_pcm_bits bits;                         /*!< no of bits */
  enum dvb_audio_endianess endianess;                   /* big or little endian */
  int is_signed;        /*!< 0 == unsigned, != signed */
  int no_of_channels;   /*!< 1 for mono, 2 stereo  */
};
```

This I/O-Control is used to provide the necessary informations about the PCM data to the decoder.

```
#define DVB_AUDIO_SET_PCM_INFO _IOW(DVB_IOCTL_BASE, 0x6d, struct dvb_audio_pcm_info)
```

8.4 Mixer and output control

The mixer has two stages:

First, a number of inputs are mixed to a number of internal sub-groups (L, R, C, SL, SR, W, Laux, Raux, ...) Note: Not all input channels can be mixed to every sub-group, e.g. R-in cannot be mixed to L-out. This is hardware dependent.

Then, the sub-groups are fed through tone control and master volume to the outputs (TV, VCR, aux, ...) Note: Not all subgroups can be routed to every output, e.g. the aux subgroup may only be mixed to the AUX output. This is hardware dependent.

Additionally, test tones (beeps) can be generated and mixed to the outputs.

The "aux" channel is used e.g. for separate headphone outputs.

All levels are in the range 0..1000. The driver will take internal measures to prevent clipping if the hardware requires it. Bass and treble gains are in the range -1000..1000. They may not be available for every output/channel.

I don't know any reasonable way to describe the hardware restrictions for the mixer. Simple API use cases should be portable, though.

The mixer has a number of outputs. They are categorised as main outputs and auxiliary outputs. Auxiliary outputs are always stereo while main outputs could be stereo or multichannel.

This enum describes the output channels.

```
enum dvb_mixer_output_channel {
  DVB_MIXER_OUTPUT_CH_L   = (1 << 0), /*!< left channel */
  DVB_MIXER_OUTPUT_CH_R   = (1 << 1), /*!< right channel */
  DVB_MIXER_OUTPUT_CH_C   = (1 << 2), /*!< center channel */
  DVB_MIXER_OUTPUT_CH_LFE = (1 << 3), /*!< low freq effects channel */
  DVB_MIXER_OUTPUT_CH_LS  = (1 << 4), /*!< left surround channel */
  DVB_MIXER_OUTPUT_CH_RS  = (1 << 5), /*!< right surround channel */
  DVB_MIXER_OUTPUT_CH_S   = (1 << 6), /*!< single surround channel */
  DVB_MIXER_OUTPUT_CH_ALL = (1 << 7), /*!< all channels */
};
```

This enum describes the different mixer outputs.

```
enum dvb_mixer_output {
  DVB_MIXER_OUTPUT_MAIN0, /*!< main output 0 */
  DVB_MIXER_OUTPUT_MAIN1, /*!< main output 1 */
  DVB_MIXER_OUTPUT_AUX0,  /*!< auxiliary output 0 */
  DVB_MIXER_OUTPUT_AUX1,  /*!< auxiliary output 1 */
};
```

This struct describes the mixer source for a mixer output.

```
struct dvb_mixer_source {
  enum dvb_mixer_output output; /*!< output to configure */
  int decoder_fd;               /*!< file desc. of decoder/post-proc/PCM-pass-thru */
};
```

This I/O-Control sets the source for a mixer output. Depending on the hardware restrictions only certain outputs may be assigned to a decoder.

```
#define DVB_MIXER_SET_SOURCE _IOW(DVB_IOCTL_BASE, 0x70, struct dvb_mixer_source)
```

Return codes:

- EPERM: an impossible case has been selected (see above)

This struct describes with output is to be muted or unmuted

```
struct dvb_mixer_mute {
  enum dvb_mixer_output output; /*!< output to configure */
  int mute;  /*!< 0 == open, != mute */
};
```

This I/O-Control mutes or unmutes an output.

```
#define DVB_MIXER_SET_MUTE _IOW(DVB_IOCTL_BASE, 0x71, struct dvb_mixer_mute)
```

This struct is used to configure the output levels.

```
struct dvb_mixer_level {
  enum dvb_mixer_output output;           /*!< output to configure */
  enum dvb_mixer_output_channel channels; /*!< channel to configure */
  int level;                              /*!< level left channel */
};
```

This I/O-Control configures the output levels of one output.

```
#define DVB_MIXER_SET_OUTPUT_LEVEL _IOW(DVB_IOCTL_BASE, 0x72, struct dvb_mixer_level)
```

This struct is used to configure the tone controls.

```
struct dvb_mixer_tone {
  enum dvb_mixer_output output;           /*!< output to configure */
  enum dvb_mixer_output_channel channels; /*!< channel to configure */
  int bass;    /*!< desired bass */
  int treble;  /*!< desired treble */
};
```

This I/O-Control configures the tone controls of one output

```
#define DVB_MIXER_SET_TONE _IOW(DVB_IOCTL_BASE, 0x73, struct dvb_mixer_tone)
```

This struct is used to configure the balance.

```
struct dvb_mixer_balance {
  enum dvb_mixer_output output;           /*!< output to configure */
  enum dvb_mixer_output_channel channels; /*!< channel to configure */
  int balance;                            /*!< desired balance */
};
```

This I/O-Control configures the tone controls of one output.

```
#define DVB_MIXER_SET_BALANCE _IOW(DVB_IOCTL_BASE, 0x74, struct dvb_mixer_balance)
```

This struct is used to control test tones.

```
struct dvb_audio_beep_param {
  unsigned int frequency; /*!< frequency of the test tone */
  int level;              /*!< level, set to 0 to disable */
};
```

This I/O-Control configures and enables test tones.

```
#define DVB_MIXER_SET_BEEP _IOW(DVB_IOCTL_BASE, 0x75, struct dvb_audio_beep_param)
```

8.5 S/P-DIF output

The S/P-DIF output can be fed from encoded stream data (the audio decode just performs synchronization), from one decoder/post-proc or from mixer output (2ch only).

This enum describes the possible sources for a S/P-DIF signal

```
enum dvb_audio_spdif_source {
  DVB_AUDIO_SPDIF_SOURCE_PP,  /*!< deocder output */
  DVB_AUDIO_SPDIF_SOURCE_DEC, /*!< decoder output w/o post-proc */
  DVB_AUDIO_SPDIF_SOURCE_ES,  /*!< raw elementary stream data */
};
```

This struct is used to configure the S/P-DIF output

```
struct dvb_audio_spdif_config {
  int source_fd;                      /*!< dec/post-proc or mixer file descriptor */
  enum dvb_audio_spdif_source source;  /*!< signal source */
  unsigned int fs;                    /*!< sampling frequency 32000/44100/48000 Hz */
  unsigned int word_length;           /*!< 16...24 bit */
};
```

This I/O-Control configures an S/P-DIF output

```
#define DVB_AUDIO_SET_SPDIF _IOW(DVB_IOCTL_BASE, 0x80, struct dvb_audio_spdif_config)
```

8.6 Audio decoder status

This enum describes the play state the decoder is in.

```
enum dvb_audio_play_state {
  DVB_AUDIO_STOPPED, /*!< the decoder is idle */
  DVB_AUDIO_PLAYING, /*!< the decoder is playing */
  DVB_AUDIO_INVALID, /*!< the decoder state is invalid */
};
```

This enum describes the possible audio status items.

```
enum dvb_audio_status {
  DVB_AUDIO_PRIVATE           = (1 << 0), /*!< some private data from the hw driver */
  DVB_AUDIO_PLAY_STATE        = (1 << 1), /*!< play state changed */
  DVB_AUDIO_SAMPLING_FREQUENCY = (1 << 2), /*!< sampling frequency changed */
};
```

This struct is used query status of the audio decoder.

```
struct dvb_audio_status_query {
  uint32_t priv[16];

  enum dvb_audio_play_state play_state;                /*!< if the decoder is currently decoding */
  enum dvb_audio_sampling_frequency sampling_frequency; /*!< the current sampling frequency */

  enum dvb_audio_status status;
};
```

This I/O-Control queries the current audio deocder status.

```
#define DVB_AUDIO_GET_STATUS _IOWR(DVB_IOCTL_BASE, 0x6b, struct dvb_audio_status_query)
```

8.7 Post processing

Output of the decoder can optionally be routed through a post-processor. Common post-processing algorithms include stereo downmix, Dolby Prologic or SRS decoding. However, the capabilities of different hardware vary too much to address this in a standard API.

9 Video API

MPEG hardware video decoders can be found on most set-top-box chipsets. They can either retrieve the video data from the demux or they can be fed directly from userspace. Userspace applications can control the playback and can be notified about video decoder events.

Video decoders may take as input a single ES (elementary stream), PES (packetized elementary stream) or stillpictures in various formats. Depending on hardware capabilities the input stream can be MPEG-2 or MPEG-1 video.

Each video device can only be opened once with write permissions (mode O_WRONLY or O_RDWR).

Video decoders can usually be connected to to a demux using the DVB_VIDEO_SET_SOURCE on a device open with write permissions by providing the filedescriptor of the demux open. If an appropriate filter for the video stream is set on the demux, then the video data is usually transmitted from the demux to the video decoder internally. Multiplexed streams (MPEG-2 TS/PS/PES, MPEG-1) can only be passed through the demux.

Non-packetized formats or stillpicture data can usually be written directly to the video decoder using the write() system call, after DVB_VIDEO_SET_SOURCE has

9.1 Capabilities

General capability handling is explained in section **??**. Drivers are expected to deliver valid informations for all possible capabilities.

This enum describes all possibly capabilities a video MPEG decoder might support.

```
enum dvb_video_capability {
  DVB_VIDEO_CAP_SOURCE_FORMATS,      /*!< bitmask of the supported dvb_video_source_format formats */
  DVB_VIDEO_CAP_PROFILE_LEVEL,       /*!< video upper decoding capability */
  DVB_VIDEO_CAP_PRESENTATION_FORMAT, /*!< bitmask of the supported dvb_video_presentation_format presentation_formats
};
```

This struct is used to query the decoder capabilities.

```
struct dvb_video_caps {
  enum dvb_video_capability cap; /*!< cabapility to query  */
  unsigned int              val; /*!< output value by the driver */
};
```

This I/O-Control requests a capability information from the driver.

```
#define DVB_VIDEO_GET_CAPS _IOWR(DVB_IOCTL_BASE, 0x40, struct dvb_video_caps)
```

Return codes:

- EINVAL: the requested capability is invalid

9.2 Input routing and synchronisation

Video decoders can be connected to a demux device or can accept data from a user application through the write() system call. Setting up the video decoder requires write permissions on the file descriptor.

This enum describes the input source of the video data.

```
enum dvb_video_source_type {
  DVB_VIDEO_SOURCE_DEMUX,  /*!< pass through the corresponding demux device */
  DVB_VIDEO_SOURCE_MEMORY, /*!< directly into the decoder via \c write() system call */
};
```

This enum describes the different video source formats supported by the MPEG decoder.

Depending on the hardware capabilities, video decoders might support packetized and non-packetized MPEG-2 or MPEG-1 formats as well as different stillpicture formats.

```
enum dvb_video_source_format {
  DVB_VIDEO_MPEG1_PES    = (1 << 0), /*!< MPEG1 packetized elementary stream */
  DVB_VIDEO_MPEG1_ES     = (1 << 1), /*!< MPEG1 elementary stream */
  DVB_VIDEO_MPEG2_PES    = (1 << 2), /*!< MPEG2 packetized elementary stream */
  DVB_VIDEO_MPEG2_ES     = (1 << 3), /*!< MPEG2 elementary stream */
  DVB_VIDEO_STILL_FRAME  = (1 << 4), /*!< MPEG2 es frames (I frame starting with sequence_header) */
  DVB_VIDEO_STILL_JPEG   = (1 << 5), /*!< JPEG frame */
  DVB_VIDEO_STILL_YUV    = (1 << 6), /*!< YUV frame */
};
```

This struct is used to set the input source of a demux device.

```
struct dvb_video_source {
  enum dvb_video_source_type   type;   /*!< input source type */
  enum dvb_video_source_format format; /*!< desired source format to be used */
  /*!< type == DVB_VIDEO_SOURCE_DEMUX only */
  int  fd;     /*!< file descriptor of the demux device */
};
```

This I/O-Control either configures a connection between a demux device and the decoder or prepares the MPEG video decoder to accept video data through the write() system call.

```
#define DVB_VIDEO_SET_SOURCE _IOW(DVB_IOCTL_BASE, 0x41, struct dvb_video_source)
```

Return codes:

- EINVAL: input source type is unknown.
- EFIXME: desired source format cannot be delivered by the demux.

This I/O-Control is used to provide a file descriptor for a demux that is responsible for video synchronization.

For TS playback, this has to be a demux filedescriptor where a filter of type DVB_DEMUX_FILTER_TYPE_
has been set.

For PS playback, the demux filedescriptor where the PS is passed through has to be provided.

```
#define DVB_VIDEO_SET_REF_STC _IOW(DVB_IOCTL_BASE, 0x42, int /* demux fd */)
```

Return codes:

- EINVAL: the filedescriptor doesn't belong to a valid DVB device

For further informations about audio/video synchronization have a look at the audio API section.

9.3 Presentation and auto scaling

There is limited support for presentation and auto scaling to ease application development. Everything which is more complex should be done through other means like DirectFB.

This enum describes the possible video presentation formats

```
enum dvb_video_presentation_format {
  DVB_VIDEO_UNSCALED        = (1 << 0),  /*!< unscaled or unknown presentation format */
  DVB_VIDEO_LETTER_BOX_16_9 = (1 << 1),  /*!< Display 16:9 letterbox on 4:3 screen */
  DVB_VIDEO_LETTER_BOX_14_9 = (1 << 2),  /*!< Display 14:9 letterbox on 4:3 screen */
  DVB_VIDEO_PAN_SCAN        = (1 << 3),  /*!< Display cut out (with pan-scan vectors) on 4:3 screen */
  DVB_VIDEO_CENTER_CUT_OUT  = (1 << 4),  /*!< Display center cut out on 4:3 screen */
  DVB_VIDEO_PILLARBOX       = (1 << 5),  /*!< Display 4:3 pillarbox on 16:9 screen */
  DVB_VIDEO_SCALE_16_9      = (1 << 6),  /*!< Display scaled 16:9 letterbox in 4:3 frame on a 16:9 screen */
  DVB_VIDEO_SCALE_14_9      = (1 << 7),  /*!< Display scaled 16:9 letterbox (shoot & protect 14:9) in a 4:3 frame on a
  DVB_VIDEO_SCALE_4_3       = (1 << 8),  /*!< Display scaled 16:9 letterbox (shoot & protect 4:3) in a 4:3 frame on a 4
  DVB_VIDEO_SCALE_UP        = (1 << 9),  /*!< Display full size scaled */
};
```

This I/O-Control sets the presentation format

```
#define DVB_VIDEO_SET_PRESENTATION_FORMAT _IOW(DVB_IOCTL_BASE, 0x50, enum dvb_video_presentation_format)
```

Return codes:

- ENODEV: the video source hasn't been set
- ENOSYS: the decoder isn't in a state to handle this command (ie. state is stopped, but freeze is called)

9.4 Decoder control

This I/O-Control starts video playback with specified speed.

- speed = 1000 : normal play
- 0 < speed < 1000 : slow forward
- speed > 1000 : fast forward
- speed = -1000 : reverse play
- -1000 < speed < 0: slow reverse
- speed < -1000 : fast reverse

```
#define DVB_VIDEO_PLAY _IOW(DVB_IOCTL_BASE, 0x43, int /* speed */)
```

This I/O-Control stops playback immediately.

```
#define DVB_VIDEO_STOP _IO(DVB_IOCTL_BASE, 0x44)
```

This I/O-Control freezes playback after next frame has been decoded, play state is set to frozen.

```
#define DVB_VIDEO_STEP _IO(DVB_IOCTL_BASE, 0x45)
```

This I/O-Control freezes playback immediately.

```
#define DVB_VIDEO_FREEZE _IO(DVB_IOCTL_BASE, 0x49)
```

This I/O-Control continues playback.

```
#define DVB_VIDEO_CONTINUE _IO(DVB_IOCTL_BASE, 0x4a)
```

This enum describes the available video decoding modes.

```
enum dvb_video_decode_mode {
  DVB_VIDEO_FRAME_ANY    = (1 << 0), /*!< all frames */
  DVB_VIDEO_FRAME_I      = (1 << 1), /*!< I-frames only */
  DVB_VIDEO_FRAME_IP     = (1 << 2), /*!< I- and P-frames only*/
  DVB_VIDEO_FIELD_TOP    = (1 << 3), /*!< only top fields */
  DVB_VIDEO_FIELD_BOTTOM = (1 << 4)  /*!< only bottom fields*/
};
```

This I/O-Control sets the video decoding modes, only valid when decoder is currently playing.

```
#define DVB_VIDEO_SET_DECODE_MODE _IOW(DVB_IOCTL_BASE, 0x46, enum dvb_video_decode_mode)
```

This I/O-Control clears decoder buffers (necessary when stream input is not continuous, e.g. seeking in stream or reverse playback).

```
#define DVB_VIDEO_CLEAR_BUFFER _IO(DVB_IOCTL_BASE,  0x47)
```

Return codes:

- `ENODEV`: the video source hasn't been set

This I/O-Control clears the frame buffer(s) so that only a black image is presented.

```
#define DVB_VIDEO_CLEAR_FRAME_BUF _IO(DVB_IOCTL_BASE, 0x48)
```

9.5 Stillpicture display

Stillpicture playback is achieved by setting the video decoder source to memory and specifying the dvb_video_source_format as one of the DVB_VIDEO_STILL_xxx members.

Subsequent calls to write() will then the instruct the video decoder to show the desired stillpicture(s).

9.6 ES header information

This enum describes the available video aspect ratios.

```
enum dvb_video_aspect_ratio {
  DVB_VIDEO_ASPECT_RATIO_INVALID, /*!< unknown or invalid aspect ration */
  DVB_VIDEO_ASPECT_RATIO_4_3,     /*!< 4:3 aspect ratio*/
  DVB_VIDEO_ASPECT_RATIO_16_9,    /*!< 16:9 aspect ratio*/
  DVB_VIDEO_ASPECT_RATIO_221,     /*!< 2.21:1 aspect ratio */
  DVB_VIDEO_ASPECT_RATIO_1        /*!< source aspect ratio 1:1 (square pixels) */
};
```

This enum describes the available video chroma formats.

```
enum dvb_video_chroma_format {
  DVB_VIDEO_CHROMA_FORMAT_INVALID, /*!< invalid or unknown chroma format */
  DVB_VIDEO_CHROMA_FORMAT_420,     /*!< YUV420 */
  DVB_VIDEO_CHROMA_FORMAT_422,     /*!< YUV422 */
  DVB_VIDEO_CHROMA_FORMAT_444,     /*!< YUV444 */
};
```

This enum describes the available video formats.

```
enum dvb_video_video_format {
  DVB_VIDEO_VIDEO_FORMAT_COMPONENT,  /*!< Component Video Format */
  DVB_VIDEO_VIDEO_FORMAT_PAL,        /*!< PAL Video Format */
  DVB_VIDEO_VIDEO_FORMAT_NTSC,       /*!< NTSC Video Format */
  DVB_VIDEO_VIDEO_FORMAT_SECAM,      /*!< SECAM Video Format */
  DVB_VIDEO_VIDEO_FORMAT_MAC,        /*!< MAC Video Format */
  DVB_VIDEO_VIDEO_FORMAT_UNSPECIFIED,/*!< UNSPECIFIED Video Format */
};
```

This enum describes if certain header extensions are present.

```
enum dvb_video_header_extensions {
  DVB_VIDEO_SEQUENCE_HDR                 = (1 << 0), /*!< sequence header present */
  DVB_VIDEO_SEQUENCE_EXTENSION    = (1 << 1), /*!< sequence extension present */
  DVB_VIDEO_SEQUENCE_DISPLAY_EXTENSION  = (1 << 2), /*!< sequence display extension present */
  DVB_VIDEO_SEQUENCE_USER_DATA           = (1 << 3), /*!< user data present */
};
```

This enum describes the decoder profile level capabilities.

```
enum dvb_video_profile_level {
  DVB_VIDEO_HP_AT_HL   = (1 <<  0), /*!< HIGH Profile @ HIGH Level */
  DVB_VIDEO_HP_AT_H14  = (1 <<  1), /*!< HIGH Profile @ HIGH-1440 Level */
  DVB_VIDEO_HP_AT_ML   = (1 <<  2), /*!< HIGH Profile @ MAIN Level */
  DVB_VIDEO_SPT_AT_H14 = (1 <<  3), /*!< SPAT Profile @ HIGH-1440 Level */
  DVB_VIDEO_SNR_AT_ML  = (1 <<  4), /*!< SNR Profile  @ MAIN Level */
  DVB_VIDEO_SNR_AT_LL  = (1 <<  5), /*!< SNR Profile  @ LOW Level */
  DVB_VIDEO_MP_AT_HL   = (1 <<  6), /*!< MAIN Profile @ HIGH Level */
  DVB_VIDEO_MP_AT_H14  = (1 <<  7), /*!< MAIN Profile @ HIGH-1440 Level */
  DVB_VIDEO_MP_AT_ML   = (1 <<  8), /*!< MAIN Profile @ MAIN Level */
  DVB_VIDEO_MP_AT_LL   = (1 <<  9), /*!< MAIN Profile @ LOW Level */
  DVB_VIDEO_SP_AT_ML   = (1 << 10), /*!< SIMPLE Profile @ MAIN Level */
  DVB_VIDEO_422_AT_HL  = (1 << 11), /*!< 4:2:2 Profile  @ HIGH Level */
  DVB_VIDEO_422_AT_ML  = (1 << 12), /*!< 4:2:2 Profile  @ MAIN Level */
  DVB_VIDEO_MVP_AT_HL  = (1 << 13), /*!< MVP Profile  @ HIGH Level */
  DVB_VIDEO_MVP_AT_H14 = (1 << 14), /*!< MVP Profile  @ HIGH-1440 Level */
  DVB_VIDEO_MVP_AT_ML  = (1 << 15), /*!< MVP Profile  @ MAIN Level */
  DVB_VIDEO_MVP_AT_LL  = (1 << 16), /*!< MVP Profile  @ LOW Level */
};
```

This struct is used to provide informations about the sequence header

```
struct dvb_video_sequence_header {
  enum dvb_video_header_extensions extensions; /*!< bitfield, available extensions */
  /*! if DVB_VIDEO_SEQUENCE_HDR */
  unsigned int w;                              /*!< width */
  unsigned int h;                              /*!< height */
  enum dvb_video_aspect_ratio ar;              /*!< aspect ratio */
  unsigned int frame_rate;                     /*!< in frames per 1000sec */
  unsigned int bit_rate;                       /*!< in bit/sec */
  unsigned int vbv_buffer_size;                /*!< vbv buffer size */

  /*! if DVB_VIDEO_SEQUENCE_EXTENSION */
  enum dvb_video_profile_level profile_level; /*!< profile level */
  uint8_t progressive;                        /*!< boolean, true if progressive */
  enum dvb_video_chroma_format cf;            /*!< chroma format */

  /*! if DVB_VIDEO_SEQUENCE_DISPLAY_EXTENSION */
  unsigned int display_vertical_size;          /*!< fixme */
  unsigned int display_horizontal_size;        /*!< fixme */
  unsigned int frame_center_vertical_offset;   /*!< fixme */
  unsigned int frame_center_horizontal_offset; /*!< fixme */
  enum dvb_video_video_format vf;              /*!< video format type */

  /*! if DVB_VIDEO_SEQUENCE_USER_DATA */
  unsigned int afd;                      /* Active Format Descriptor */
};
```

This struct is used to provide informations about the pes header

```
struct dvb_video_pes_header {
  uint8_t trick_mode_control; /*!< the trick mode control byte from the header */
};
```

This I/O-Control retrieves the last decoded sequence header from the video decoder

```
#define DVB_VIDEO_GET_SEQHDR _IOR(DVB_IOCTL_BASE, 0x4e, struct dvb_video_sequence_header)
```

9.7 Video decoder status

This enum describes the play state the decoder is in

```
enum dvb_video_play_state {
  DVB_VIDEO_STOPPED, /*!< the decoder is idle */
  DVB_VIDEO_PLAYING, /*!< the decoder is playing */
  DVB_VIDEO_FROZEN,  /*!< the playback is currently frozen */
  DVB_VIDEO_PICTURE, /*!< the playback is currently showing a single I-Frame or a dripfeed */
};
```

This enum is used to enumerate the different status that can be queried

```
enum dvb_video_status {
  DVB_VIDEO_PRIVATE                = (1 <<  0), /*!< some private data from the hw driver */
  DVB_VIDEO_PLAY_STATE             = (1 <<  1), /*!< decoder play state */
  DVB_VIDEO_DECODE_MODE            = (1 <<  2), /*!< decode mode changed */
  DVB_VIDEO_PRESENTATION_FORMAT    = (1 <<  3), /*!< presentation format changed */
  DVB_VIDEO_ASPECT_RATIO           = (1 <<  4), /*!< aspect ratio changed*/
  DVB_VIDEO_PES_HEADER_CHANGED     = (1 <<  5), /*!< the pes header has changes */
  DVB_VIDEO_SEQUENCE_HEADER_CHANGED = (1 <<  6), /*!< the sequence header has changed (including extension, if present)
  DVB_VIDEO_ERROR_COUNT            = (1 <<  7), /*!< count of errors since the last read (e.g. number of sequence head
  DVB_VIDEO_IFRAME_COUNT           = (1 <<  8), /*!< count of errors since the last read (e.g. number of sequence head
  DVB_VIDEO_PLAY_SPEED             = (1 <<  9), /*!< decoder play speed */
};
```

This struct is used query the status of the video decoder

```
struct dvb_video_status_query {
  uint32_t     priv[16];

  enum dvb_video_play_state          play_state;          /*!< */
  enum dvb_video_decode_mode         decode_mode;         /*!< */
  enum dvb_video_presentation_format presentation_format; /*!< */
  enum dvb_video_aspect_ratio        aspect_ratio;        /*!< */

  struct dvb_video_pes_header pes_header;           /*!< */
  struct dvb_video_sequence_header sequence_header; /*!< */

  size_t       error_count;  /*!< number of defective frame since last start*/
  size_t       iframe_count; /*!< number of iframes decoded since last start*/
  int      playspeed;    /*!< current playspeed */

  enum dvb_video_status status; /*!< mask/unmask desired status members */
};
```

This I/O-Control queries the video deocder status items specified by 'status'. All video decoders are expected to support that alle items mentioned above can be queried. If used on a blocking fd, only the specified status bits wake up the sleep.

```
#define DVB_VIDEO_GET_STATUS _IOWR(DVB_IOCTL_BASE, 0x4b, struct dvb_video_status_query)
```

Return codes:

- EFIXME: fixme

10 Network API

Broadcasting IP over DVB is a common practise for Internet downstreams ("SkyDSL"). This struct is used to set up a network interface

```
struct dvb_net_if {
  __u16 pid;  /*!< pid which is carrying the data */
  __u16 if_num;   /*!< logical interface number */
};
```

This I/O-Control adds a DVB network interface dvbM_N, fed by MPE packets from 'pid' M is the DVB adapter number, N is the interface number, counting from 0.

```
#define NET_ADD_IF    _IOWR(DVB_IOCTL_BASE, 0xa0, struct dvb_net_if)
```

Return codes:

- EBUSY: if if_num is already in use, or no filter for pid is available
- EPERM: if the caller is not root

This I/O-Control removes a DVB network interface.

```
#define NET_REMOVE_IF _IOW(DVB_IOCTL_BASE, 0xa1, int /* if_num */)
```

Return codes:

- ENODEV: if if_num not present
- EPERM: if the caller is not root

This I/O-Control retrieves informations about a network interface, if_num is input, pid is output

```
#define NET_GET_IF    _IOWR(DVB_IOCTL_BASE, 0xa2, struct dvb_net_if)
```

Return codes:

- ENODEV: if if_num not present

11 Abbreviations

- API = application programming interface
- CI/ CA= common interface, common access
- CVS = concurrent versioning system
- DMA = direct memory access
- DSM- CC = digital storage media command and control
- DVB = digital video broadcast
- HDD = hard disk drive
- IDTV = integrated digital television
- MHP = multimedia home platform
- OSD = on- screen display
- PES = packetized elementary stream
- PS = program stream
- SPU = subtitle processing unit
- S/ P- DIF = Sony/ Philips digital interface
- STB = set top box
- TS = transport stream

12 GNU Free Documentation License

Preamble

The purpose of this License is to make a manual, textbook, or other functional and useful 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.

1. APPLICABILITY AND DEFINITIONS

This License applies to any manual or other work, in any medium, that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. Such a notice grants a world-wide, royalty-free license, unlimited in duration, to use that work under the conditions stated herein. The **"Document"**, below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as **"you"**. You accept the license if you copy, modify or distribute the work in a way requiring permission under copyright law.

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. (Thus, 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. If a section does not fit the above definition of Secondary then it is not allowed to be designated as Invariant. The Document may contain zero Invariant Sections. If the Document does not identify any Invariant Sections then there are none.

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 Front-Cover Text may be at most 5 words, and a Back-Cover Text may be at most 25 words.

A **"Transparent"** copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, that is suitable for revising the document 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, or absence of markup, has been arranged to thwart or discourage subsequent modification by readers is not Transparent. An image format is not Transparent if used for any substantial amount of text. 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, PostScript or PDF designed for human modification. Examples of transparent image formats include PNG, XCF and JPG. Opaque formats include 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, PostScript or PDF 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.

A section **"Entitled XYZ"** means a named subunit of the Document whose title either is precisely XYZ or contains XYZ in parentheses following text that translates XYZ in another language. (Here XYZ stands for a specific section name mentioned below, such as **"Acknowledgements"**, **"Dedications"**, **"Endorsements"**, or **"History"**.) To **"Preserve the Title"** of such a section when you modify the Document means that it remains a section "Entitled XYZ" according to this definition.

The Document may include Warranty Disclaimers next to the notice which states that this License applies to the Document. These Warranty Disclaimers are considered to be included by reference in this License, but only as regards disclaiming warranties: any other implication that these Warranty Disclaimers may have is void and has no effect on the meaning of this License.

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.

3. COPYING IN QUANTITY

If you publish printed copies (or copies in media that commonly have printed covers) 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 computer-network location from which the general network-using public has access to download using public-standard network protocols a complete Transparent copy of the Document, free of added material. 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.

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 fewer than five), unless they release you from this requirement.

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", Preserve 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. For any section Entitled "Acknowledgements" or "Dedications", Preserve the Title of the section, 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 to be Entitled "Endorsements" or to conflict in title with any Invariant Section.

O. Preserve any Warranty Disclaimers.

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.

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, and that you preserve all their Warranty Disclaimers.

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".

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.

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, is called an "aggregate" if the copyright resulting from the compilation is not used to limit the legal rights of the compilation's users beyond what the individual works permit. When the Document is included in an aggregate, this License does not apply to the other works in the aggregate which 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 half of the entire aggregate, the Document's Cover Texts may be placed on covers that bracket the Document within the aggregate, or the electronic equivalent of covers if the Document is in electronic form. Otherwise they must appear on printed covers that bracket the whole aggregate.

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, and all the license notices in the Document, and any Warranty Disclaimers, provided that you also include the original English version of this License and the original versions of those notices and disclaimers. In case of a disagreement between the translation and the original version of this License or a notice or disclaimer, the original version will prevail.

If a section in the Document is Entitled "Acknowledgements", "Dedications", or "History", the requirement (section 4) to Preserve its Title (section 1) will typically require changing the actual title.

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.

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.