```
<title>Kernel Demux API</title>
<para>The kernel demux API defines a driver-internal interface for registering
low-level,
hardware specific driver to a hardware independent demux layer. It is only of
interest for
DVB device driver writers. The header file for this API is named <emphasis
role="tt">demux.h</emphasis> and located in
<emphasis role="tt">drivers/media/dvb/dvb-core</emphasis>.
</para>
<para>Maintainer note: This section must be reviewed. It is probably out of
date.
</para>

<section id="kernel_demux_data_types">
<title>Kernel Demux Data Types</title>


<section id="dmx_success_t">
<title>dmx_success_t</title>
 <programlisting>
 typedef enum {
   DMX_OK = 0, /&#x22C6; Received Ok &#x22C6;/
   DMX_LENGTH_ERROR, /&#x22C6; Incorrect length &#x22C6;/
   DMX_OVERRUN_ERROR, /&#x22C6; Receiver ring buffer overrun &#x22C6;/
   DMX_CRC_ERROR, /&#x22C6; Incorrect CRC &#x22C6;/
   DMX_FRAME_ERROR, /&#x22C6; Frame alignment error &#x22C6;/
   DMX_FIFO_ERROR, /&#x22C6; Receiver FIFO overrun &#x22C6;/
   DMX_MISSED_ERROR /&#x22C6; Receiver missed packet &#x22C6;/
 } dmx_success_t;
</programlisting>

</section>
<section id="ts_filter_types">
<title>TS filter types</title>
 <programlisting>

/&#x22C6;----------------------------------------------------------------------
---&#x22C6;/
 /&#x22C6; TS packet reception &#x22C6;/

/&#x22C6;----------------------------------------------------------------------
---&#x22C6;/

 /&#x22C6; TS filter type for set_type() &#x22C6;/

 #define TS_PACKET       1   /&#x22C6; send TS packets (188 bytes) to callback
(default) &#x22C6;/
 #define TS_PAYLOAD_ONLY 2   /&#x22C6; in case TS_PACKET is set, only send the
TS
                               payload (&#x003C;=184 bytes per packet) to
callback &#x22C6;/
 #define TS_DECODER      4   /&#x22C6; send stream to built-in decoder (if
present) &#x22C6;/
</programlisting>

</section>
```

```
<section id="dmx_ts_pes_t">
<title>dmx_ts_pes_t</title>
<para>The structure
</para>
<programlisting>
 typedef enum
 {
         DMX_TS_PES_AUDIO,    /&#x22C6; also send packets to audio decoder (if it
exists) &#x22C6;/
         DMX_TS_PES_VIDEO,    /&#x22C6; ... &#x22C6;/
         DMX_TS_PES_TELETEXT,
         DMX_TS_PES_SUBTITLE,
         DMX_TS_PES_PCR,
         DMX_TS_PES_OTHER,
 } dmx_ts_pes_t;
</programlisting>
<para>describes the PES type for filters which write to a built-in decoder. The
correspond (and
should be kept identical) to the types in the demux device.
</para>
<programlisting>
 struct dmx_ts_feed_s {
         int is_filtering; /&#x22C6; Set to non-zero when filtering in progress
&#x22C6;/
         struct dmx_demux_s&#x22C6; parent; /&#x22C6; Back-pointer &#x22C6;/
         void&#x22C6; priv; /&#x22C6; Pointer to private data of the API client
&#x22C6;/
         int (&#x22C6;set) (struct dmx_ts_feed_s&#x22C6; feed,
                     __u16 pid,
                     size_t callback_length,
                     size_t circular_buffer_size,
                     int descramble,
                     struct timespec timeout);
         int (&#x22C6;start_filtering) (struct dmx_ts_feed_s&#x22C6; feed);
         int (&#x22C6;stop_filtering) (struct dmx_ts_feed_s&#x22C6; feed);
         int (&#x22C6;set_type) (struct dmx_ts_feed_s&#x22C6; feed,
                         int type,
                         dmx_ts_pes_t pes_type);
 };

 typedef struct dmx_ts_feed_s dmx_ts_feed_t;
</programlisting>
 <programlisting>

/&#x22C6;----------------------------------------------------------------------
---&#x22C6;/
 /&#x22C6; PES packet reception (not supported yet) &#x22C6;/

/&#x22C6;----------------------------------------------------------------------
---&#x22C6;/

 typedef struct dmx_pes_filter_s {
         struct dmx_pes_s&#x22C6; parent; /&#x22C6; Back-pointer &#x22C6;/
         void&#x22C6; priv; /&#x22C6; Pointer to private data of the API client
&#x22C6;/
 } dmx_pes_filter_t;
```

```
</programlisting>
 <programlisting>
 typedef struct dmx_pes_feed_s {
         int is_filtering; /&#x22C6; Set to non-zero when filtering in progress
&#x22C6;/
         struct dmx_demux_s&#x22C6; parent; /&#x22C6; Back-pointer &#x22C6;/
         void&#x22C6; priv; /&#x22C6; Pointer to private data of the API client
&#x22C6;/
         int (&#x22C6;set) (struct dmx_pes_feed_s&#x22C6; feed,
                     __u16 pid,
                     size_t circular_buffer_size,
                     int descramble,
                     struct timespec timeout);
         int (&#x22C6;start_filtering) (struct dmx_pes_feed_s&#x22C6; feed);
         int (&#x22C6;stop_filtering) (struct dmx_pes_feed_s&#x22C6; feed);
         int (&#x22C6;allocate_filter) (struct dmx_pes_feed_s&#x22C6; feed,
                                 dmx_pes_filter_t&#x22C6;&#x22C6; filter);
         int (&#x22C6;release_filter) (struct dmx_pes_feed_s&#x22C6; feed,
                                dmx_pes_filter_t&#x22C6; filter);
 } dmx_pes_feed_t;
</programlisting>
 <programlisting>
 typedef struct {
         __u8 filter_value [DMX_MAX_FILTER_SIZE];
         __u8 filter_mask [DMX_MAX_FILTER_SIZE];
         struct dmx_section_feed_s&#x22C6; parent; /&#x22C6; Back-pointer
&#x22C6;/
         void&#x22C6; priv; /&#x22C6; Pointer to private data of the API client
&#x22C6;/
 } dmx_section_filter_t;
</programlisting>
 <programlisting>
 struct dmx_section_feed_s {
         int is_filtering; /&#x22C6; Set to non-zero when filtering in progress
&#x22C6;/
         struct dmx_demux_s&#x22C6; parent; /&#x22C6; Back-pointer &#x22C6;/
         void&#x22C6; priv; /&#x22C6; Pointer to private data of the API client
&#x22C6;/
         int (&#x22C6;set) (struct dmx_section_feed_s&#x22C6; feed,
                     __u16 pid,
                     size_t circular_buffer_size,
                     int descramble,
                     int check_crc);
         int (&#x22C6;allocate_filter) (struct dmx_section_feed_s&#x22C6; feed,
                                 dmx_section_filter_t&#x22C6;&#x22C6; filter);
         int (&#x22C6;release_filter) (struct dmx_section_feed_s&#x22C6; feed,
                                dmx_section_filter_t&#x22C6; filter);
         int (&#x22C6;start_filtering) (struct dmx_section_feed_s&#x22C6; feed);
         int (&#x22C6;stop_filtering) (struct dmx_section_feed_s&#x22C6; feed);
 };
 typedef struct dmx_section_feed_s dmx_section_feed_t;


/&#x22C6;-----------------------------------------------------------------------
---&#x22C6;/
 /&#x22C6; Callback functions &#x22C6;/
```

```
/&#x22C6;------------------------------------------------------------------------
---&#x22C6;/

 typedef int (&#x22C6;dmx_ts_cb) ( __u8 &#x22C6; buffer1,
                                size_t buffer1_length,
                                __u8 &#x22C6; buffer2,
                                size_t buffer2_length,
                                dmx_ts_feed_t&#x22C6; source,
                                dmx_success_t success);

 typedef int (&#x22C6;dmx_section_cb) ( __u8 &#x22C6; buffer1,
                                    size_t buffer1_len,
                                    __u8 &#x22C6; buffer2,
                                    size_t buffer2_len,
                                    dmx_section_filter_t &#x22C6; source,
                                    dmx_success_t success);

 typedef int (&#x22C6;dmx_pes_cb) ( __u8 &#x22C6; buffer1,
                                size_t buffer1_len,
                                __u8 &#x22C6; buffer2,
                                size_t buffer2_len,
                                dmx_pes_filter_t&#x22C6; source,
                                dmx_success_t success);


/&#x22C6;------------------------------------------------------------------------
---&#x22C6;/
 /&#x22C6; DVB Front-End &#x22C6;/

/&#x22C6;------------------------------------------------------------------------
---&#x22C6;/

 typedef enum {
         DMX_OTHER_FE = 0,
         DMX_SATELLITE_FE,
         DMX_CABLE_FE,
         DMX_TERRESTRIAL_FE,
         DMX_LVDS_FE,
         DMX_ASI_FE, /&#x22C6; DVB-ASI interface &#x22C6;/
         DMX_MEMORY_FE
 } dmx_frontend_source_t;

 typedef struct {
         /&#x22C6; The following char&#x22C6; fields point to NULL terminated
strings &#x22C6;/
         char&#x22C6; id;                    /&#x22C6; Unique front-end
identifier &#x22C6;/
         char&#x22C6; vendor;                /&#x22C6; Name of the front-end
vendor &#x22C6;/
         char&#x22C6; model;                 /&#x22C6; Name of the front-end
model &#x22C6;/
         struct list_head connectivity_list; /&#x22C6; List of front-ends that
can
                                                be connected to a particular
                                                demux &#x22C6;/
```

```
        void&#x22C6; priv;     /&#x22C6; Pointer to private data of the API
client &#x22C6;/
        dmx_frontend_source_t source;
 } dmx_frontend_t;


/&#x22C6;----------------------------------------------------------------------
---&#x22C6;/
 /&#x22C6; MPEG-2 TS Demux &#x22C6;/

/&#x22C6;----------------------------------------------------------------------
---&#x22C6;/

 /&#x22C6;
  &#x22C6; Flags OR'ed in the capabilites field of struct dmx_demux_s.
  &#x22C6;/

 #define DMX_TS_FILTERING                        1
 #define DMX_PES_FILTERING                       2
 #define DMX_SECTION_FILTERING                   4
 #define DMX_MEMORY_BASED_FILTERING              8    /&#x22C6; write()
available &#x22C6;/
 #define DMX_CRC_CHECKING                        16
 #define DMX_TS_DESCRAMBLING                     32
 #define DMX_SECTION_PAYLOAD_DESCRAMBLING        64
 #define DMX_MAC_ADDRESS_DESCRAMBLING            128
</programlisting>

</section>
<section id="demux_demux_t">
<title>demux_demux_t</title>
 <programlisting>
 /&#x22C6;
  &#x22C6; DMX_FE_ENTRY(): Casts elements in the list of registered
  &#x22C6; front-ends from the generic type struct list_head
  &#x22C6; to the type &#x22C6; dmx_frontend_t
  &#x22C6;.
 &#x22C6;/

 #define DMX_FE_ENTRY(list) list_entry(list, dmx_frontend_t, connectivity_list)

 struct dmx_demux_s {
        /&#x22C6; The following char&#x22C6; fields point to NULL terminated
strings &#x22C6;/
        char&#x22C6; id;                    /&#x22C6; Unique demux identifier
&#x22C6;/
        char&#x22C6; vendor;                /&#x22C6; Name of the demux vendor
&#x22C6;/
        char&#x22C6; model;                 /&#x22C6; Name of the demux model
&#x22C6;/
        __u32 capabilities;          /&#x22C6; Bitfield of capability flags
&#x22C6;/
        dmx_frontend_t&#x22C6; frontend;    /&#x22C6; Front-end connected to
the demux &#x22C6;/
        struct list_head reg_list;   /&#x22C6; List of registered demuxes
&#x22C6;/
```

```
        void&#x22C6; priv;                  /&#x22C6; Pointer to private data
of the API client &#x22C6;/
        int users;                  /&#x22C6; Number of users &#x22C6;/
        int (&#x22C6;open) (struct dmx_demux_s&#x22C6; demux);
        int (&#x22C6;close) (struct dmx_demux_s&#x22C6; demux);
        int (&#x22C6;write) (struct dmx_demux_s&#x22C6; demux, const
char&#x22C6; buf, size_t count);
        int (&#x22C6;allocate_ts_feed) (struct dmx_demux_s&#x22C6; demux,
                                dmx_ts_feed_t&#x22C6;&#x22C6; feed,
                                dmx_ts_cb callback);
        int (&#x22C6;release_ts_feed) (struct dmx_demux_s&#x22C6; demux,
                               dmx_ts_feed_t&#x22C6; feed);
        int (&#x22C6;allocate_pes_feed) (struct dmx_demux_s&#x22C6; demux,
                                dmx_pes_feed_t&#x22C6;&#x22C6; feed,
                                dmx_pes_cb callback);
        int (&#x22C6;release_pes_feed) (struct dmx_demux_s&#x22C6; demux,
                               dmx_pes_feed_t&#x22C6; feed);
        int (&#x22C6;allocate_section_feed) (struct dmx_demux_s&#x22C6; demux,
                                    dmx_section_feed_t&#x22C6;&#x22C6; feed,
                                    dmx_section_cb callback);
        int (&#x22C6;release_section_feed) (struct dmx_demux_s&#x22C6; demux,
                                   dmx_section_feed_t&#x22C6; feed);
        int (&#x22C6;descramble_mac_address) (struct dmx_demux_s&#x22C6; demux,
                                     __u8&#x22C6; buffer1,
                                     size_t buffer1_length,
                                     __u8&#x22C6; buffer2,
                                     size_t buffer2_length,
                                     __u16 pid);
        int (&#x22C6;descramble_section_payload) (struct dmx_demux_s&#x22C6;
demux,
                                         __u8&#x22C6; buffer1,
                                         size_t buffer1_length,
                                         __u8&#x22C6; buffer2, size_t
buffer2_length,
                                         __u16 pid);
        int (&#x22C6;add_frontend) (struct dmx_demux_s&#x22C6; demux,
                              dmx_frontend_t&#x22C6; frontend);
        int (&#x22C6;remove_frontend) (struct dmx_demux_s&#x22C6; demux,
                                 dmx_frontend_t&#x22C6; frontend);
        struct list_head&#x22C6; (&#x22C6;get_frontends) (struct
dmx_demux_s&#x22C6; demux);
        int (&#x22C6;connect_frontend) (struct dmx_demux_s&#x22C6; demux,
                                dmx_frontend_t&#x22C6; frontend);
        int (&#x22C6;disconnect_frontend) (struct dmx_demux_s&#x22C6; demux);


        /&#x22C6; added because js cannot keep track of these himself &#x22C6;/
        int (&#x22C6;get_pes_pids) (struct dmx_demux_s&#x22C6; demux, __u16
&#x22C6;pids);
 };
 typedef struct dmx_demux_s dmx_demux_t;
</programlisting>

</section>
<section id="demux_directory">
<title>Demux directory</title>
```

```
 <programlisting>
 /&#x22C6;
  &#x22C6; DMX_DIR_ENTRY(): Casts elements in the list of registered
  &#x22C6; demuxes from the generic type struct list_head&#x22C6; to the type
dmx_demux_t
  &#x22C6;.
  &#x22C6;/

 #define DMX_DIR_ENTRY(list) list_entry(list, dmx_demux_t, reg_list)

 int dmx_register_demux (dmx_demux_t&#x22C6; demux);
 int dmx_unregister_demux (dmx_demux_t&#x22C6; demux);
 struct list_head&#x22C6; dmx_get_demuxes (void);
</programlisting>
 </section></section>
<section id="demux_directory_api">
<title>Demux Directory API</title>
<para>The demux directory is a Linux kernel-wide facility for registering and
accessing the
MPEG-2 TS demuxes in the system. Run-time registering and unregistering of demux
drivers
is possible using this API.
</para>
<para>All demux drivers in the directory implement the abstract interface
dmx_demux_t.
</para>

<section
role="subsection"><title>dmx_register_demux()</title>
<para>DESCRIPTION
</para>
<informaltable><tgroup cols="1"><tbody><row><entry
 align="char">
<para>This function makes a demux driver interface available to the Linux
kernel. It is
 usually called by the init_module() function of the kernel module that contains
 the demux driver. The caller of this function is responsible for allocating
 dynamic or static memory for the demux structure and for initializing its
fields
 before calling this function. The memory allocated for the demux structure
 must not be freed before calling dmx_unregister_demux(),</para>
</entry>
 </row></tbody></tgroup></informaltable>
<para>SYNOPSIS
</para>
<informaltable><tgroup cols="1"><tbody><row><entry
 align="char">
<para>int dmx_register_demux ( dmx_demux_t &#x22C6;demux )</para>
</entry>
 </row></tbody></tgroup></informaltable>
<para>PARAMETERS
</para>
<informaltable><tgroup cols="2"><tbody><row><entry
 align="char">
<para>dmx_demux_t*
 demux</para>
```

```
</entry><entry
 align="char">
<para>Pointer to the demux structure.</para>
</entry>
 </row></tbody></tgroup></informaltable>
<para>RETURNS
</para>
<informaltable><tgroup cols="2"><tbody><row><entry
 align="char">
<para>0</para>
</entry><entry
 align="char">
<para>The function was completed without errors.</para>
</entry>
 </row><row><entry
 align="char">
<para>-EEXIST</para>
</entry><entry
 align="char">
<para>A demux with the same value of the id field already stored
 in the directory.</para>
</entry>
 </row><row><entry
 align="char">
<para>-ENOSPC</para>
</entry><entry
 align="char">
<para>No space left in the directory.</para>
</entry>
 </row></tbody></tgroup></informaltable>

</section><section
role="subsection"><title>dmx_unregister_demux()</title>
<para>DESCRIPTION
</para>
<informaltable><tgroup cols="1"><tbody><row><entry
 align="char">
<para>This function is called to indicate that the given demux interface is no
 longer available. The caller of this function is responsible for freeing the
 memory of the demux structure, if it was dynamically allocated before calling
 dmx_register_demux(). The cleanup_module() function of the kernel module
 that contains the demux driver should call this function. Note that this
function
 fails if the demux is currently in use, i.e., release_demux() has not been
called
 for the interface.</para>
</entry>
 </row></tbody></tgroup></informaltable>
<para>SYNOPSIS
</para>
<informaltable><tgroup cols="1"><tbody><row><entry
 align="char">
<para>int dmx_unregister_demux ( dmx_demux_t &amp;#x22C6;demux )</para>
</entry>
 </row></tbody></tgroup></informaltable>
<para>PARAMETERS
```

```
</para>
<informaltable><tgroup cols="2"><tbody><row><entry
 align="char">
<para>dmx_demux_t*
 demux</para>
</entry><entry
 align="char">
<para>Pointer to the demux structure which is to be
 unregistered.</para>
</entry>
 </row></tbody></tgroup></informaltable>
<para>RETURNS
</para>
<informaltable><tgroup cols="2"><tbody><row><entry
 align="char">
<para>0</para>
</entry><entry
 align="char">
<para>The function was completed without errors.</para>
</entry>
 </row><row><entry
 align="char">
<para>ENODEV</para>
</entry><entry
 align="char">
<para>The specified demux is not registered in the demux
 directory.</para>
</entry>
 </row><row><entry
 align="char">
<para>EBUSY</para>
</entry><entry
 align="char">
<para>The specified demux is currently in use.</para>
</entry>
 </row></tbody></tgroup></informaltable>

</section><section
role="subsection"><title>dmx_get_demuxes()</title>
<para>DESCRIPTION
</para>
<informaltable><tgroup cols="1"><tbody><row><entry
 align="char">
<para>Provides the caller with the list of registered demux interfaces, using
the
 standard list structure defined in the include file linux/list.h. The include
file
 demux.h defines the macro DMX_DIR_ENTRY() for converting an element of
 the generic type struct list_head* to the type dmx_demux_t*. The caller must
 not free the memory of any of the elements obtained via this function
call.</para>
</entry>
 </row></tbody></tgroup></informaltable>
<para>SYNOPSIS
</para>
<informaltable><tgroup cols="1"><tbody><row><entry
```

 align="char">
<para>struct list_head ⋆dmx_get_demuxes ()</para>
</entry>
 </row></tbody></tgroup></informaltable>
<para>PARAMETERS
</para>
<informaltable><tgroup cols="2"><tbody><row><entry
 align="char">
<para>none</para>
</entry>
 </row></tbody></tgroup></informaltable>
<para>RETURNS
</para>
<informaltable><tgroup cols="2"><tbody><row><entry
 align="char">
<para>struct list_head *</para>
</entry><entry
 align="char">
<para>A list of demux interfaces, or NULL in the case of an
 empty list.</para>
</entry>
 </row></tbody></tgroup></informaltable>
 </section></section>
<section id="demux_api">
<title>Demux API</title>
<para>The demux API should be implemented for each demux in the system. It is used to select
the TS source of a demux and to manage the demux resources. When the demux client allocates a resource via the demux API, it receives a pointer to the API of that
resource.
</para>
<para>Each demux receives its TS input from a DVB front-end or from memory, as set via the
demux API. In a system with more than one front-end, the API can be used to select one of
the DVB front-ends as a TS source for a demux, unless this is fixed in the HW platform. The
demux API only controls front-ends regarding their connections with demuxes; the APIs
used to set the other front-end parameters, such as tuning, are not defined in this
document.
</para>
<para>The functions that implement the abstract interface demux should be defined static or
module private and registered to the Demux Directory for external access. It is not necessary
to implement every function in the demux_t struct, however (for example, a demux interface
might support Section filtering, but not TS or PES filtering). The API client is expected to
check the value of any function pointer before calling the function: the value of NULL means
“function not available”.
</para>

<para>Whenever the functions of the demux API modify shared data, the possibilities of lost
update and race condition problems should be addressed, e.g. by protecting parts of code with
mutexes. This is especially important on multi-processor hosts.
</para>
<para>Note that functions called from a bottom half context must not sleep, at least in the 2.2.x
kernels. Even a simple memory allocation can result in a kernel thread being put to sleep if
swapping is needed. For example, the Linux kernel calls the functions of a network device
interface from a bottom half context. Thus, if a demux API function is called from network
device code, the function must not sleep.
</para>

<section id="kdapi_fopen">
<title>open()</title>
<para>DESCRIPTION
</para>
<informaltable><tgroup cols="1"><tbody><row><entry
 align="char">
<para>This function reserves the demux for use by the caller and, if necessary,
 initializes the demux. When the demux is no longer needed, the function close()
 should be called. It should be possible for multiple clients to access the demux
 at the same time. Thus, the function implementation should increment the
 demux usage count when open() is called and decrement it when close() is
 called.</para>
</entry>
 </row></tbody></tgroup></informaltable>
<para>SYNOPSIS
</para>
<informaltable><tgroup cols="1"><tbody><row><entry
 align="char">
<para>int open (demux_t⋆ demux);</para>
</entry>
 </row></tbody></tgroup></informaltable>
<para>PARAMETERS
</para>
<informaltable><tgroup cols="2"><tbody><row><entry
 align="char">
<para>demux_t* demux</para>
</entry><entry
 align="char">
<para>Pointer to the demux API and instance data.</para>
</entry>
 </row></tbody></tgroup></informaltable>
<para>RETURNS
</para>
<informaltable><tgroup cols="2"><tbody><row><entry
 align="char">
<para>0</para>
</entry><entry

```
 align="char">
<para>The function was completed without errors.</para>
</entry>
 </row><row><entry
 align="char">
<para>-EUSERS</para>
</entry><entry
 align="char">
<para>Maximum usage count reached.</para>
</entry>
 </row><row><entry
 align="char">
<para>-EINVAL</para>
</entry><entry
 align="char">
<para>Bad parameter.</para>
</entry>
 </row></tbody></tgroup></informaltable>

</section>
<section id="kdapi_fclose">
<title>close()</title>
<para>DESCRIPTION
</para>
<informaltable><tgroup cols="1"><tbody><row><entry
 align="char">
<para>This function reserves the demux for use by the caller and, if necessary,
 initializes the demux. When the demux is no longer needed, the function close()
 should be called. It should be possible for multiple clients to access the
demux
 at the same time. Thus, the function implementation should increment the
 demux usage count when open() is called and decrement it when close() is
 called.</para>
</entry>
 </row></tbody></tgroup></informaltable>
<para>SYNOPSIS
</para>
<informaltable><tgroup cols="1"><tbody><row><entry
 align="char">
<para>int close(demux_t&amp;#x22C6; demux);</para>
</entry>
 </row></tbody></tgroup></informaltable>
<para>PARAMETERS
</para>
<informaltable><tgroup cols="2"><tbody><row><entry
 align="char">
<para>demux_t* demux</para>
</entry><entry
 align="char">
<para>Pointer to the demux API and instance data.</para>
</entry>
 </row></tbody></tgroup></informaltable>
<para>RETURNS
</para>
<informaltable><tgroup cols="2"><tbody><row><entry
 align="char">
```

```
<para>0</para>
</entry><entry
 align="char">
<para>The function was completed without errors.</para>
</entry>
 </row><row><entry
 align="char">
<para>-ENODEV</para>
</entry><entry
 align="char">
<para>The demux was not in use.</para>
</entry>
 </row><row><entry
 align="char">
<para>-EINVAL</para>
</entry><entry
 align="char">
<para>Bad parameter.</para>
</entry>
 </row></tbody></tgroup></informaltable>

</section>
<section id="kdapi_fwrite">
<title>write()</title>
<para>DESCRIPTION
</para>
<informaltable><tgroup cols="1"><tbody><row><entry
 align="char">
<para>This function provides the demux driver with a memory buffer containing TS
 packets. Instead of receiving TS packets from the DVB front-end, the demux
 driver software will read packets from memory. Any clients of this demux
 with active TS, PES or Section filters will receive filtered data via the Demux
 callback API (see 0). The function returns when all the data in the buffer has
 been consumed by the demux. Demux hardware typically cannot read TS from
 memory. If this is the case, memory-based filtering has to be implemented
 entirely in software.</para>
</entry>
 </row></tbody></tgroup></informaltable>
<para>SYNOPSIS
</para>
<informaltable><tgroup cols="1"><tbody><row><entry
 align="char">
<para>int write(demux_t&#x22C6; demux, const char&#x22C6; buf, size_t
 count);</para>
</entry>
 </row></tbody></tgroup></informaltable>
<para>PARAMETERS
</para>
<informaltable><tgroup cols="2"><tbody><row><entry
 align="char">
<para>demux_t* demux</para>
</entry><entry
 align="char">
<para>Pointer to the demux API and instance data.</para>
</entry>
 </row><row><entry
```

```
 align="char">
<para>const char* buf</para>
</entry><entry
 align="char">
<para>Pointer to the TS data in kernel-space memory.</para>
</entry>
 </row><row><entry
 align="char">
<para>size_t length</para>
</entry><entry
 align="char">
<para>Length of the TS data.</para>
</entry>
 </row></tbody></tgroup></informaltable>
<para>RETURNS
</para>
<informaltable><tgroup cols="2"><tbody><row><entry
 align="char">
<para>0</para>
</entry><entry
 align="char">
<para>The function was completed without errors.</para>
</entry>
 </row><row><entry
 align="char">
<para>-ENOSYS</para>
</entry><entry
 align="char">
<para>The command is not implemented.</para>
</entry>
 </row><row><entry
 align="char">
<para>-EINVAL</para>
</entry><entry
 align="char">
<para>Bad parameter.</para>
</entry>
 </row></tbody></tgroup></informaltable>

</section><section
role="subsection"><title>allocate_ts_feed()</title>
<para>DESCRIPTION
</para>
<informaltable><tgroup cols="1"><tbody><row><entry
 align="char">
<para>Allocates a new TS feed, which is used to filter the TS packets carrying a
 certain PID. The TS feed normally corresponds to a hardware PID filter on the
 demux chip.</para>
</entry>
 </row></tbody></tgroup></informaltable>
<para>SYNOPSIS
</para>
<informaltable><tgroup cols="1"><tbody><row><entry
 align="char">
<para>int allocate_ts_feed(dmx_demux_t&amp;#x22C6; demux,
 dmx_ts_feed_t&amp;#x22C6;&amp;#x22C6; feed, dmx_ts_cb callback);</para>
```

```
</entry>
 </row></tbody></tgroup></informaltable>
<para>PARAMETERS
</para>
<informaltable><tgroup cols="2"><tbody><row><entry
 align="char">
<para>demux_t* demux</para>
</entry><entry
 align="char">
<para>Pointer to the demux API and instance data.</para>
</entry>
 </row><row><entry
 align="char">
<para>dmx_ts_feed_t**
 feed</para>
</entry><entry
 align="char">
<para>Pointer to the TS feed API and instance data.</para>
</entry>
 </row><row><entry
 align="char">
<para>dmx_ts_cb callback</para>
</entry><entry
 align="char">
<para>Pointer to the callback function for passing received TS
 packet</para>
</entry>
 </row></tbody></tgroup></informaltable>
<para>RETURNS
</para>
<informaltable><tgroup cols="2"><tbody><row><entry
 align="char">
<para>0</para>
</entry><entry
 align="char">
<para>The function was completed without errors.</para>
</entry>
 </row><row><entry
 align="char">
<para>-EBUSY</para>
</entry><entry
 align="char">
<para>No more TS feeds available.</para>
</entry>
 </row><row><entry
 align="char">
<para>-ENOSYS</para>
</entry><entry
 align="char">
<para>The command is not implemented.</para>
</entry>
 </row><row><entry
 align="char">
<para>-EINVAL</para>
</entry><entry
 align="char">
```

<para>Bad parameter.</para>
</entry>
 </row></tbody></tgroup></informaltable>

</section><section
role="subsection"><title>release_ts_feed()</title>
<para>DESCRIPTION
</para>
<informaltable><tgroup cols="1"><tbody><row><entry
 align="char">
<para>Releases the resources allocated with allocate_ts_feed(). Any filtering in
 progress on the TS feed should be stopped before calling this function.</para>
</entry>
 </row></tbody></tgroup></informaltable>
<para>SYNOPSIS
</para>
<informaltable><tgroup cols="1"><tbody><row><entry
 align="char">
<para>int release_ts_feed(dmx_demux_t⋆ demux,
 dmx_ts_feed_t⋆ feed);</para>
</entry>
 </row></tbody></tgroup></informaltable>
<para>PARAMETERS
</para>
<informaltable><tgroup cols="2"><tbody><row><entry
 align="char">
<para>demux_t* demux</para>
</entry><entry
 align="char">
<para>Pointer to the demux API and instance data.</para>
</entry>
 </row><row><entry
 align="char">
<para>dmx_ts_feed_t* feed</para>
</entry><entry
 align="char">
<para>Pointer to the TS feed API and instance data.</para>
</entry>
 </row></tbody></tgroup></informaltable>
<para>RETURNS
</para>
<informaltable><tgroup cols="2"><tbody><row><entry
 align="char">
<para>0</para>
</entry><entry
 align="char">
<para>The function was completed without errors.</para>
</entry>
 </row><row><entry
 align="char">
<para>-EINVAL</para>
</entry><entry
 align="char">
<para>Bad parameter.</para>
</entry>
 </row></tbody></tgroup></informaltable>

```
</section><section
role="subsection"><title>allocate_section_feed()</title>
<para>DESCRIPTION
</para>
<informaltable><tgroup cols="1"><tbody><row><entry
 align="char">
<para>Allocates a new section feed, i.e. a demux resource for filtering and
receiving
 sections. On platforms with hardware support for section filtering, a section
 feed is directly mapped to the demux HW. On other platforms, TS packets are
 first PID filtered in hardware and a hardware section filter then emulated in
 software. The caller obtains an API pointer of type dmx_section_feed_t as an
 out parameter. Using this API the caller can set filtering parameters and start
 receiving sections.</para>
</entry>
 </row></tbody></tgroup></informaltable>
<para>SYNOPSIS
</para>
<informaltable><tgroup cols="1"><tbody><row><entry
 align="char">
<para>int allocate_section_feed(dmx_demux_t&#x22C6; demux,
 dmx_section_feed_t &#x22C6;&#x22C6;feed, dmx_section_cb callback);</para>
</entry>
 </row></tbody></tgroup></informaltable>
<para>PARAMETERS
</para>
<informaltable><tgroup cols="2"><tbody><row><entry
 align="char">
<para>demux_t *demux</para>
</entry><entry
 align="char">
<para>Pointer to the demux API and instance data.</para>
</entry>
 </row><row><entry
 align="char">
<para>dmx_section_feed_t
 **feed</para>
</entry><entry
 align="char">
<para>Pointer to the section feed API and instance data.</para>
</entry>
 </row><row><entry
 align="char">
<para>dmx_section_cb
 callback</para>
</entry><entry
 align="char">
<para>Pointer to the callback function for passing received
 sections.</para>
</entry>
 </row></tbody></tgroup></informaltable>
<para>RETURNS
</para>
<informaltable><tgroup cols="2"><tbody><row><entry
 align="char">
```

<para>0</para>
</entry><entry
 align="char">
<para>The function was completed without errors.</para>
</entry>
 </row><row><entry
 align="char">
<para>-EBUSY</para>
</entry><entry
 align="char">
<para>No more section feeds available.</para>
</entry>
 </row><row><entry
 align="char">
<para>-ENOSYS</para>
</entry><entry
 align="char">
<para>The command is not implemented.</para>
</entry>
 </row><row><entry
 align="char">
<para>-EINVAL</para>
</entry><entry
 align="char">
<para>Bad parameter.</para>
</entry>
 </row></tbody></tgroup></informaltable>

</section><section
role="subsection"><title>release_section_feed()</title>
<para>DESCRIPTION
</para>
<informaltable><tgroup cols="1"><tbody><row><entry
 align="char">
<para>Releases the resources allocated with allocate_section_feed(), including
 allocated filters. Any filtering in progress on the section feed should be
stopped
 before calling this function.</para>
</entry>
 </row></tbody></tgroup></informaltable>
<para>SYNOPSIS
</para>
<informaltable><tgroup cols="1"><tbody><row><entry
 align="char">
<para>int release_section_feed(dmx_demux_t⋆ demux,
 dmx_section_feed_t ⋆feed);</para>
</entry>
 </row></tbody></tgroup></informaltable>
<para>PARAMETERS
</para>
<informaltable><tgroup cols="2"><tbody><row><entry
 align="char">
<para>demux_t *demux</para>
</entry><entry
 align="char">
<para>Pointer to the demux API and instance data.</para>

```
</entry>
 </row><row><entry
 align="char">
<para>dmx_section_feed_t
 *feed</para>
</entry><entry
 align="char">
<para>Pointer to the section feed API and instance data.</para>
</entry>
 </row></tbody></tgroup></informaltable>
<para>RETURNS
</para>
<informaltable><tgroup cols="2"><tbody><row><entry
 align="char">
<para>0</para>
</entry><entry
 align="char">
<para>The function was completed without errors.</para>
</entry>
 </row><row><entry
 align="char">
<para>-EINVAL</para>
</entry><entry
 align="char">
<para>Bad parameter.</para>
</entry>
 </row></tbody></tgroup></informaltable>

</section><section
role="subsection"><title>descramble_mac_address()</title>
<para>DESCRIPTION
</para>
<informaltable><tgroup cols="1"><tbody><row><entry
 align="char">
<para>This function runs a descrambling algorithm on the destination MAC
 address field of a DVB Datagram Section, replacing the original address
 with its un-encrypted version. Otherwise, the description on the function
 descramble_section_payload() applies also to this function.</para>
</entry>
 </row></tbody></tgroup></informaltable>
<para>SYNOPSIS
</para>
<informaltable><tgroup cols="1"><tbody><row><entry
 align="char">
<para>int descramble_mac_address(dmx_demux_t&amp;#x22C6; demux, __u8
 &amp;#x22C6;buffer1, size_t buffer1_length, __u8 &amp;#x22C6;buffer2,
 size_t buffer2_length, __u16 pid);</para>
</entry>
 </row></tbody></tgroup></informaltable>
<para>PARAMETERS
</para>
<informaltable><tgroup cols="2"><tbody><row><entry
 align="char">
<para>dmx_demux_t
 *demux</para>
</entry><entry
```

```
 align="char">
<para>Pointer to the demux API and instance data.</para>
</entry>
 </row><row><entry
 align="char">
<para>__u8 *buffer1</para>
</entry><entry
 align="char">
<para>Pointer to the first byte of the section.</para>
</entry>
 </row><row><entry
 align="char">
<para>size_t buffer1_length</para>
</entry><entry
 align="char">
<para>Length of the section data, including headers and CRC,
 in buffer1.</para>
</entry>
 </row><row><entry
 align="char">
<para>__u8* buffer2</para>
</entry><entry
 align="char">
<para>Pointer to the tail of the section data, or NULL. The
 pointer has a non-NULL value if the section wraps past
 the end of a circular buffer.</para>
</entry>
 </row><row><entry
 align="char">
<para>size_t buffer2_length</para>
</entry><entry
 align="char">
<para>Length of the section data, including headers and CRC,
 in buffer2.</para>
</entry>
 </row><row><entry
 align="char">
<para>__u16 pid</para>
</entry><entry
 align="char">
<para>The PID on which the section was received. Useful
 for obtaining the descrambling key, e.g. from a DVB
 Common Access facility.</para>
</entry>
 </row></tbody></tgroup></informaltable>
<para>RETURNS
</para>
<informaltable><tgroup cols="2"><tbody><row><entry
 align="char">
<para>0</para>
</entry><entry
 align="char">
<para>The function was completed without errors.</para>
</entry>
 </row><row><entry
 align="char">
```

<para>-ENOSYS</para>
</entry><entry
 align="char">
<para>No descrambling facility available.</para>
</entry>
 </row><row><entry
 align="char">
<para>-EINVAL</para>
</entry><entry
 align="char">
<para>Bad parameter.</para>
</entry>
 </row></tbody></tgroup></informaltable>

</section><section
role="subsection"><title>descramble_section_payload()</title>
<para>DESCRIPTION
</para>
<informaltable><tgroup cols="1"><tbody><row><entry
 align="char">
<para>This function runs a descrambling algorithm on the payload of a DVB
 Datagram Section, replacing the original payload with its un-encrypted
 version. The function will be called from the demux API implementation;
 the API client need not call this function directly. Section-level scrambling
 algorithms are currently standardized only for DVB-RCC (return channel
 over 2-directional cable TV network) systems. For all other DVB networks,
 encryption schemes are likely to be proprietary to each data broadcaster. Thus,
 it is expected that this function pointer will have the value of NULL (i.e.,
 function not available) in most demux API implementations. Nevertheless, it
 should be possible to use the function pointer as a hook for dynamically adding
 a “plug-in” descrambling facility to a demux driver.</para>
</entry>
 </row><row><entry
 align="char">
<para>While this function is not needed with hardware-based section
descrambling,
 the descramble_section_payload function pointer can be used to override the
 default hardware-based descrambling algorithm: if the function pointer has a
 non-NULL value, the corresponding function should be used instead of any
 descrambling hardware.</para>
</entry>
 </row></tbody></tgroup></informaltable>
<para>SYNOPSIS
</para>
<informaltable><tgroup cols="1"><tbody><row><entry
 align="char">
<para>int descramble_section_payload(dmx_demux_t⋆ demux,
 __u8 ⋆buffer1, size_t buffer1_length, __u8 ⋆buffer2,
 size_t buffer2_length, __u16 pid);</para>
</entry>
 </row></tbody></tgroup></informaltable>
<para>PARAMETERS
</para>
<informaltable><tgroup cols="2"><tbody><row><entry
 align="char">
<para>dmx_demux_t

```
 *demux</para>
</entry><entry
 align="char">
<para>Pointer to the demux API and instance data.</para>
</entry>
 </row><row><entry
 align="char">
<para>__u8 *buffer1</para>
</entry><entry
 align="char">
<para>Pointer to the first byte of the section.</para>
</entry>
 </row><row><entry
 align="char">
<para>size_t buffer1_length</para>
</entry><entry
 align="char">
<para>Length of the section data, including headers and CRC,
 in buffer1.</para>
</entry>
 </row><row><entry
 align="char">
<para>__u8 *buffer2</para>
</entry><entry
 align="char">
<para>Pointer to the tail of the section data, or NULL. The
 pointer has a non-NULL value if the section wraps past
 the end of a circular buffer.</para>
</entry>
 </row><row><entry
 align="char">
<para>size_t buffer2_length</para>
</entry><entry
 align="char">
<para>Length of the section data, including headers and CRC,
 in buffer2.</para>
</entry>
 </row><row><entry
 align="char">
<para>__u16 pid</para>
</entry><entry
 align="char">
<para>The PID on which the section was received. Useful
 for obtaining the descrambling key, e.g. from a DVB
 Common Access facility.</para>
</entry>
 </row></tbody></tgroup></informaltable>
<para>RETURNS
</para>
<informaltable><tgroup cols="2"><tbody><row><entry
 align="char">
<para>0</para>
</entry><entry
 align="char">
<para>The function was completed without errors.</para>
</entry>
```

```
 </row><row><entry
 align="char">
<para>-ENOSYS</para>
</entry><entry
 align="char">
<para>No descrambling facility available.</para>
</entry>
 </row><row><entry
 align="char">
<para>-EINVAL</para>
</entry><entry
 align="char">
<para>Bad parameter.</para>
</entry>
 </row></tbody></tgroup></informaltable>

</section><section
role="subsection"><title>add_frontend()</title>
<para>DESCRIPTION
</para>
<informaltable><tgroup cols="1"><tbody><row><entry
 align="char">
<para>Registers a connectivity between a demux and a front-end, i.e., indicates
that
 the demux can be connected via a call to connect_frontend() to use the given
 front-end as a TS source. The client of this function has to allocate dynamic
or
 static memory for the frontend structure and initialize its fields before
calling
 this function. This function is normally called during the driver
initialization.
 The caller must not free the memory of the frontend struct before successfully
 calling remove_frontend().</para>
</entry>
 </row></tbody></tgroup></informaltable>
<para>SYNOPSIS
</para>
<informaltable><tgroup cols="1"><tbody><row><entry
 align="char">
<para>int add_frontend(dmx_demux_t &amp;#x22C6;demux, dmx_frontend_t
 &amp;#x22C6;frontend);</para>
</entry>
 </row></tbody></tgroup></informaltable>
<para>PARAMETERS
</para>
<informaltable><tgroup cols="2"><tbody><row><entry
 align="char">
<para>dmx_demux_t*
 demux</para>
</entry><entry
 align="char">
<para>Pointer to the demux API and instance data.</para>
</entry>
 </row><row><entry
 align="char">
<para>dmx_frontend_t*
```

```
 frontend</para>
</entry><entry
 align="char">
<para>Pointer to the front-end instance data.</para>
</entry>
 </row></tbody></tgroup></informaltable>
<para>RETURNS
</para>
<informaltable><tgroup cols="2"><tbody><row><entry
 align="char">
<para>0</para>
</entry><entry
 align="char">
<para>The function was completed without errors.</para>
</entry>
 </row><row><entry
 align="char">
<para>-EEXIST</para>
</entry><entry
 align="char">
<para>A front-end with the same value of the id field already
 registered.</para>
</entry>
 </row><row><entry
 align="char">
<para>-EINUSE</para>
</entry><entry
 align="char">
<para>The demux is in use.</para>
</entry>
 </row><row><entry
 align="char">
<para>-ENOMEM</para>
</entry><entry
 align="char">
<para>No more front-ends can be added.</para>
</entry>
 </row><row><entry
 align="char">
<para>-EINVAL</para>
</entry><entry
 align="char">
<para>Bad parameter.</para>
</entry>
 </row></tbody></tgroup></informaltable>

</section><section
role="subsection"><title>remove_frontend()</title>
<para>DESCRIPTION
</para>
<informaltable><tgroup cols="1"><tbody><row><entry
 align="char">
<para>Indicates that the given front-end, registered by a call to
add_frontend(), can
 no longer be connected as a TS source by this demux. The function should be
 called when a front-end driver or a demux driver is removed from the system.
```

```
 If the front-end is in use, the function fails with the return value of -EBUSY.
 After successfully calling this function, the caller can free the memory of
 the frontend struct if it was dynamically allocated before the add_frontend()
 operation.</para>
</entry>
 </row></tbody></tgroup></informaltable>
<para>SYNOPSIS
</para>
<informaltable><tgroup cols="1"><tbody><row><entry
 align="char">
<para>int remove_frontend(dmx_demux_t&#x22C6; demux,
 dmx_frontend_t&#x22C6; frontend);</para>
</entry>
 </row></tbody></tgroup></informaltable>
<para>PARAMETERS
</para>
<informaltable><tgroup cols="2"><tbody><row><entry
 align="char">
<para>dmx_demux_t*
 demux</para>
</entry><entry
 align="char">
<para>Pointer to the demux API and instance data.</para>
</entry>
 </row><row><entry
 align="char">
<para>dmx_frontend_t*
 frontend</para>
</entry><entry
 align="char">
<para>Pointer to the front-end instance data.</para>
</entry>
 </row></tbody></tgroup></informaltable>
<para>RETURNS
</para>
<informaltable><tgroup cols="2"><tbody><row><entry
 align="char">
<para>0</para>
</entry><entry
 align="char">
<para>The function was completed without errors.</para>
</entry>
 </row><row><entry
 align="char">
<para>-EINVAL</para>
</entry><entry
 align="char">
<para>Bad parameter.</para>
</entry>
 </row><row><entry
 align="char">
<para>-EBUSY</para>
</entry><entry
 align="char">
<para>The front-end is in use, i.e. a call to connect_frontend()
 has not been followed by a call to disconnect_frontend().</para>
```

```
</entry>
 </row></tbody></tgroup></informaltable>

</section><section
role="subsection"><title>get_frontends()</title>
<para>DESCRIPTION
</para>
<informaltable><tgroup cols="1"><tbody><row><entry
 align="char">
<para>Provides the APIs of the front-ends that have been registered for this
demux.
 Any of the front-ends obtained with this call can be used as a parameter for
 connect_frontend().</para>
</entry>
 </row><row><entry
 align="char">
<para>The include file demux.h contains the macro DMX_FE_ENTRY() for
 converting an element of the generic type struct list_head* to the type
 dmx_frontend_t*. The caller must not free the memory of any of the elements
 obtained via this function call.</para>
</entry>
 </row></tbody></tgroup></informaltable>
<para>SYNOPSIS
</para>
<informaltable><tgroup cols="1"><tbody><row><entry
 align="char">
<para>struct list_head&#x22C6; get_frontends(dmx_demux_t&#x22C6; demux);</para>
</entry>
 </row></tbody></tgroup></informaltable>
<para>PARAMETERS
</para>
<informaltable><tgroup cols="2"><tbody><row><entry
 align="char">
<para>dmx_demux_t*
 demux</para>
</entry><entry
 align="char">
<para>Pointer to the demux API and instance data.</para>
</entry>
 </row></tbody></tgroup></informaltable>
<para>RETURNS
</para>
<informaltable><tgroup cols="2"><tbody><row><entry
 align="char">
<para>dmx_demux_t*</para>
</entry><entry
 align="char">
<para>A list of front-end interfaces, or NULL in the case of an
 empty list.</para>
</entry>
 </row></tbody></tgroup></informaltable>

</section><section
role="subsection"><title>connect_frontend()</title>
<para>DESCRIPTION
</para>
```

<informaltable><tgroup cols="1"><tbody><row><entry
 align="char">
<para>Connects the TS output of the front-end to the input of the demux. A demux
 can only be connected to a front-end registered to the demux with the function
 add_frontend().</para>
</entry>
 </row><row><entry
 align="char">
<para>It may or may not be possible to connect multiple demuxes to the same
 front-end, depending on the capabilities of the HW platform. When not used,
 the front-end should be released by calling disconnect_frontend().</para>
</entry>
 </row></tbody></tgroup></informaltable>
<para>SYNOPSIS
</para>
<informaltable><tgroup cols="1"><tbody><row><entry
 align="char">
<para>int connect_frontend(dmx_demux_t⋆ demux,
 dmx_frontend_t⋆ frontend);</para>
</entry>
 </row></tbody></tgroup></informaltable>
<para>PARAMETERS
</para>
<informaltable><tgroup cols="2"><tbody><row><entry
 align="char">
<para>dmx_demux_t*
 demux</para>
</entry><entry
 align="char">
<para>Pointer to the demux API and instance data.</para>
</entry>
 </row><row><entry
 align="char">
<para>dmx_frontend_t*
 frontend</para>
</entry><entry
 align="char">
<para>Pointer to the front-end instance data.</para>
</entry>
 </row></tbody></tgroup></informaltable>
<para>RETURNS
</para>
<informaltable><tgroup cols="2"><tbody><row><entry
 align="char">
<para>0</para>
</entry><entry
 align="char">
<para>The function was completed without errors.</para>
</entry>
 </row><row><entry
 align="char">
<para>-EINVAL</para>
</entry><entry
 align="char">
<para>Bad parameter.</para>
</entry>

```
 </row><row><entry
 align="char">
<para>-EBUSY</para>
</entry><entry
 align="char">
<para>The front-end is in use.</para>
</entry>
 </row></tbody></tgroup></informaltable>

</section><section
role="subsection"><title>disconnect_frontend()</title>
<para>DESCRIPTION
</para>
<informaltable><tgroup cols="1"><tbody><row><entry
 align="char">
<para>Disconnects the demux and a front-end previously connected by a
 connect_frontend() call.</para>
</entry>
 </row></tbody></tgroup></informaltable>
<para>SYNOPSIS
</para>
<informaltable><tgroup cols="1"><tbody><row><entry
 align="char">
<para>int disconnect_frontend(dmx_demux_t&amp;#x22C6; demux);</para>
</entry>
 </row></tbody></tgroup></informaltable>
<para>PARAMETERS
</para>
<informaltable><tgroup cols="2"><tbody><row><entry
 align="char">
<para>dmx_demux_t*
 demux</para>
</entry><entry
 align="char">
<para>Pointer to the demux API and instance data.</para>
</entry>
 </row></tbody></tgroup></informaltable>
<para>RETURNS
</para>
<informaltable><tgroup cols="2"><tbody><row><entry
 align="char">
<para>0</para>
</entry><entry
 align="char">
<para>The function was completed without errors.</para>
</entry>
 </row><row><entry
 align="char">
<para>-EINVAL</para>
</entry><entry
 align="char">
<para>Bad parameter.</para>
</entry>
 </row></tbody></tgroup></informaltable>
 </section></section>
<section id="demux_callback_api">
```

<title>Demux Callback API</title>
<para>This kernel-space API comprises the callback functions that deliver filtered data to the
demux client. Unlike the other APIs, these API functions are provided by the client and called
from the demux code.
</para>
<para>The function pointers of this abstract interface are not packed into a structure as in the
other demux APIs, because the callback functions are registered and used independent
of each other. As an example, it is possible for the API client to provide several
callback functions for receiving TS packets and no callbacks for PES packets or sections.
</para>
<para>The functions that implement the callback API need not be re-entrant: when a demux
driver calls one of these functions, the driver is not allowed to call the function again before
the original call returns. If a callback is triggered by a hardware interrupt, it is recommended
to use the Linux “bottom half” mechanism or start a tasklet instead of making the callback
function call directly from a hardware interrupt.
</para>

<section
role="subsection"><title>dmx_ts_cb()</title>
<para>DESCRIPTION
</para>
<informaltable><tgroup cols="1"><tbody><row><entry
 align="char">
<para>This function, provided by the client of the demux API, is called from the
 demux code. The function is only called when filtering on this TS feed has
 been enabled using the start_filtering() function.</para>
</entry>
 </row><row><entry
 align="char">
<para>Any TS packets that match the filter settings are copied to a circular buffer. The
 filtered TS packets are delivered to the client using this callback function. The
 size of the circular buffer is controlled by the circular_buffer_size parameter
 of the set() function in the TS Feed API. It is expected that the buffer1 and
 buffer2 callback parameters point to addresses within the circular buffer, but
 other implementations are also possible. Note that the called party should not
 try to free the memory the buffer1 and buffer2 parameters point to.</para>
</entry>
 </row><row><entry
 align="char">
<para>When this function is called, the buffer1 parameter typically points to the
 start of the first undelivered TS packet within a circular buffer. The buffer2
 buffer parameter is normally NULL, except when the received TS packets have
 crossed the last address of the circular buffer and ”wrapped” to

the beginning
 of the buffer. In the latter case the buffer1 parameter would contain an
address
 within the circular buffer, while the buffer2 parameter would contain the first
 address of the circular buffer.</para>
</entry>
 </row><row><entry
 align="char">
<para>The number of bytes delivered with this function (i.e. buffer1_length +
 buffer2_length) is usually equal to the value of callback_length parameter
 given in the set() function, with one exception: if a timeout occurs before
 receiving callback_length bytes of TS data, any undelivered packets are
 immediately delivered to the client by calling this function. The timeout
 duration is controlled by the set() function in the TS Feed API.</para>
</entry>
 </row><row><entry
 align="char">
<para>If a TS packet is received with errors that could not be fixed by the
TS-level
 forward error correction (FEC), the Transport_error_indicator flag of the TS
 packet header should be set. The TS packet should not be discarded, as
 the error can possibly be corrected by a higher layer protocol. If the called
 party is slow in processing the callback, it is possible that the circular
buffer
 eventually fills up. If this happens, the demux driver should discard any TS
 packets received while the buffer is full. The error should be indicated to the
 client on the next callback by setting the success parameter to the value of
 DMX_OVERRUN_ERROR.</para>
</entry>
 </row><row><entry
 align="char">
<para>The type of data returned to the callback can be selected by the new
 function int (*set_type) (struct dmx_ts_feed_s* feed, int type, dmx_ts_pes_t
 pes_type) which is part of the dmx_ts_feed_s struct (also cf. to the
 include file ost/demux.h) The type parameter decides if the raw TS packet
 (TS_PACKET) or just the payload (TS_PACKET—TS_PAYLOAD_ONLY)
 should be returned. If additionally the TS_DECODER bit is set the stream
 will also be sent to the hardware MPEG decoder. In this case, the second
 flag decides as what kind of data the stream should be interpreted. The
 possible choices are one of DMX_TS_PES_AUDIO, DMX_TS_PES_VIDEO,
 DMX_TS_PES_TELETEXT, DMX_TS_PES_SUBTITLE,
 DMX_TS_PES_PCR, or DMX_TS_PES_OTHER.</para>
</entry>
 </row></tbody></tgroup></informaltable>
<para>SYNOPSIS
</para>
<informaltable><tgroup cols="1"><tbody><row><entry
 align="char">
<para>int dmx_ts_cb(__u8⋆ buffer1, size_t buffer1_length,
 __u8⋆ buffer2, size_t buffer2_length, dmx_ts_feed_t⋆
 source, dmx_success_t success);</para>
</entry>
 </row></tbody></tgroup></informaltable>
<para>PARAMETERS
</para>
<informaltable><tgroup cols="2"><tbody><row><entry

align="char">
<para>__u8* buffer1</para>
</entry><entry
align="char">
<para>Pointer to the start of the filtered TS packets.</para>
</entry>
</row><row><entry
align="char">
<para>size_t buffer1_length</para>
</entry><entry
align="char">
<para>Length of the TS data in buffer1.</para>
</entry>
</row><row><entry
align="char">
<para>__u8* buffer2</para>
</entry><entry
align="char">
<para>Pointer to the tail of the filtered TS packets, or NULL.</para>
</entry>
</row><row><entry
align="char">
<para>size_t buffer2_length</para>
</entry><entry
align="char">
<para>Length of the TS data in buffer2.</para>
</entry>
</row><row><entry
align="char">
<para>dmx_ts_feed_t*
source</para>
</entry><entry
align="char">
<para>Indicates which TS feed is the source of the callback.</para>
</entry>
</row><row><entry
align="char">
<para>dmx_success_t
success</para>
</entry><entry
align="char">
<para>Indicates if there was an error in TS reception.</para>
</entry>
</row></tbody></tgroup></informaltable>
<para>RETURNS
</para>
<informaltable><tgroup cols="2"><tbody><row><entry
align="char">
<para>0</para>
</entry><entry
align="char">
<para>Continue filtering.</para>
</entry>
</row><row><entry
align="char">
<para>-1</para>

</entry><entry
 align="char">
<para>Stop filtering - has the same effect as a call to
 stop_filtering() on the TS Feed API.</para>
</entry>
 </row></tbody></tgroup></informaltable>

</section><section
role="subsection"><title>dmx_section_cb()</title>
<para>DESCRIPTION
</para>
<informaltable><tgroup cols="1"><tbody><row><entry
 align="char">
<para>This function, provided by the client of the demux API, is called from the
 demux code. The function is only called when filtering of sections has been
 enabled using the function start_filtering() of the section feed API. When the
 demux driver has received a complete section that matches at least one section
 filter, the client is notified via this callback function. Normally this
function is
 called for each received section; however, it is also possible to deliver
multiple
 sections with one callback, for example when the system load is high. If an
 error occurs while receiving a section, this function should be called with
 the corresponding error type set in the success field, whether or not there is
 data to deliver. The Section Feed implementation should maintain a circular
 buffer for received sections. However, this is not necessary if the Section
Feed
 API is implemented as a client of the TS Feed API, because the TS Feed
 implementation then buffers the received data. The size of the circular buffer
 can be configured using the set() function in the Section Feed API. If there
 is no room in the circular buffer when a new section is received, the section
 must be discarded. If this happens, the value of the success parameter should
 be DMX_OVERRUN_ERROR on the next callback.</para>
</entry>
 </row></tbody></tgroup></informaltable>
<para>SYNOPSIS
</para>
<informaltable><tgroup cols="1"><tbody><row><entry
 align="char">
<para>int dmx_section_cb(__u8⋆ buffer1, size_t
 buffer1_length, __u8⋆ buffer2, size_t
 buffer2_length, dmx_section_filter_t⋆ source,
 dmx_success_t success);</para>
</entry>
 </row></tbody></tgroup></informaltable>
<para>PARAMETERS
</para>
<informaltable><tgroup cols="2"><tbody><row><entry
 align="char">
<para>__u8* buffer1</para>
</entry><entry
 align="char">
<para>Pointer to the start of the filtered section, e.g. within the
 circular buffer of the demux driver.</para>
</entry>
 </row><row><entry

```
 align="char">
<para>size_t buffer1_length</para>
</entry><entry
 align="char">
<para>Length of the filtered section data in buffer1, including
 headers and CRC.</para>
</entry>
 </row><row><entry
 align="char">
<para>__u8* buffer2</para>
</entry><entry
 align="char">
<para>Pointer to the tail of the filtered section data, or NULL.
 Useful to handle the wrapping of a circular buffer.</para>
</entry>
 </row><row><entry
 align="char">
<para>size_t buffer2_length</para>
</entry><entry
 align="char">
<para>Length of the filtered section data in buffer2, including
 headers and CRC.</para>
</entry>
 </row><row><entry
 align="char">
<para>dmx_section_filter_t*
 filter</para>
</entry><entry
 align="char">
<para>Indicates the filter that triggered the callback.</para>
</entry>
 </row><row><entry
 align="char">
<para>dmx_success_t
 success</para>
</entry><entry
 align="char">
<para>Indicates if there was an error in section reception.</para>
</entry>
 </row></tbody></tgroup></informaltable>
<para>RETURNS
</para>
<informaltable><tgroup cols="2"><tbody><row><entry
 align="char">
<para>0</para>
</entry><entry
 align="char">
<para>Continue filtering.</para>
</entry>
 </row><row><entry
 align="char">
<para>-1</para>
</entry><entry
 align="char">
<para>Stop filtering - has the same effect as a call to
 stop_filtering() on the Section Feed API.</para>
```

```
</entry>
 </row></tbody></tgroup></informaltable>
 </section></section>
<section id="ts_feed_api">
<title>TS Feed API</title>
<para>A TS feed is typically mapped to a hardware PID filter on the demux chip.
Using this API, the client can set the filtering properties to start/stop
filtering TS
packets on a particular TS feed. The API is defined as an abstract interface of
the type
dmx_ts_feed_t.
</para>
<para>The functions that implement the interface should be defined static or
module private. The
client can get the handle of a TS feed API by calling the function
allocate_ts_feed() in the
demux API.
</para>

<section
role="subsection"><title>set()</title>
<para>DESCRIPTION
</para>
<informaltable><tgroup cols="1"><tbody><row><entry
 align="char">
<para>This function sets the parameters of a TS feed. Any filtering in progress
on the
 TS feed must be stopped before calling this function.</para>
</entry>
 </row></tbody></tgroup></informaltable>
<para>SYNOPSIS
</para>
<informaltable><tgroup cols="1"><tbody><row><entry
 align="char">
<para>int set ( dmx_ts_feed_t&amp;#x22C6; feed, __u16 pid, size_t
 callback_length, size_t circular_buffer_size, int
 descramble, struct timespec timeout);</para>
</entry>
 </row></tbody></tgroup></informaltable>
<para>PARAMETERS
</para>
<informaltable><tgroup cols="2"><tbody><row><entry
 align="char">
<para>dmx_ts_feed_t* feed</para>
</entry><entry
 align="char">
<para>Pointer to the TS feed API and instance data.</para>
</entry>
 </row><row><entry
 align="char">
<para>__u16 pid</para>
</entry><entry
 align="char">
<para>PID value to filter. Only the TS packets carrying the
 specified PID will be passed to the API client.</para>
</entry>
```

```
 </row><row><entry
 align="char">
<para>size_t
 callback_length</para>
</entry><entry
 align="char">
<para>Number of bytes to deliver with each call to the
 dmx_ts_cb() callback function. The value of this
 parameter should be a multiple of 188.</para>
</entry>
 </row><row><entry
 align="char">
<para>size_t
 circular_buffer_size</para>
</entry><entry
 align="char">
<para>Size of the circular buffer for the filtered TS packets.</para>
</entry>
 </row><row><entry
 align="char">
<para>int descramble</para>
</entry><entry
 align="char">
<para>If non-zero, descramble the filtered TS packets.</para>
</entry>
 </row><row><entry
 align="char">
<para>struct timespec
 timeout</para>
</entry><entry
 align="char">
<para>Maximum time to wait before delivering received TS
 packets to the client.</para>
</entry>
 </row></tbody></tgroup></informaltable>
<para>RETURNS
</para>
<informaltable><tgroup cols="2"><tbody><row><entry
 align="char">
<para>0</para>
</entry><entry
 align="char">
<para>The function was completed without errors.</para>
</entry>
 </row><row><entry
 align="char">
<para>-ENOMEM</para>
</entry><entry
 align="char">
<para>Not enough memory for the requested buffer size.</para>
</entry>
 </row><row><entry
 align="char">
<para>-ENOSYS</para>
</entry><entry
 align="char">
```

<para>No descrambling facility available for TS.</para>
</entry>
 </row><row><entry
 align="char">
<para>-EINVAL</para>
</entry><entry
 align="char">
<para>Bad parameter.</para>
</entry>
 </row></tbody></tgroup></informaltable>

</section><section
role="subsection"><title>start_filtering()</title>
<para>DESCRIPTION
</para>
<informaltable><tgroup cols="1"><tbody><row><entry
 align="char">
<para>Starts filtering TS packets on this TS feed, according to its settings.
The PID
 value to filter can be set by the API client. All matching TS packets are
 delivered asynchronously to the client, using the callback function registered
 with allocate_ts_feed().</para>
</entry>
 </row></tbody></tgroup></informaltable>
<para>SYNOPSIS
</para>
<informaltable><tgroup cols="1"><tbody><row><entry
 align="char">
<para>int start_filtering(dmx_ts_feed_t⋆ feed);</para>
</entry>
 </row></tbody></tgroup></informaltable>
<para>PARAMETERS
</para>
<informaltable><tgroup cols="2"><tbody><row><entry
 align="char">
<para>dmx_ts_feed_t* feed</para>
</entry><entry
 align="char">
<para>Pointer to the TS feed API and instance data.</para>
</entry>
 </row></tbody></tgroup></informaltable>
<para>RETURNS
</para>
<informaltable><tgroup cols="2"><tbody><row><entry
 align="char">
<para>0</para>
</entry><entry
 align="char">
<para>The function was completed without errors.</para>
</entry>
 </row><row><entry
 align="char">
<para>-EINVAL</para>
</entry><entry
 align="char">
<para>Bad parameter.</para>

</entry>
 </row></tbody></tgroup></informaltable>

</section><section
role="subsection"><title>stop_filtering()</title>
<para>DESCRIPTION
</para>
<informaltable><tgroup cols="1"><tbody><row><entry
 align="char">
<para>Stops filtering TS packets on this TS feed.</para>
</entry>
 </row></tbody></tgroup></informaltable>
<para>SYNOPSIS
</para>
<informaltable><tgroup cols="1"><tbody><row><entry
 align="char">
<para>int stop_filtering(dmx_ts_feed_t⋆ feed);</para>
</entry>
 </row></tbody></tgroup></informaltable>
<para>PARAMETERS
</para>
<informaltable><tgroup cols="2"><tbody><row><entry
 align="char">
<para>dmx_ts_feed_t* feed</para>
</entry><entry
 align="char">
<para>Pointer to the TS feed API and instance data.</para>
</entry>
 </row></tbody></tgroup></informaltable>
<para>RETURNS
</para>
<informaltable><tgroup cols="2"><tbody><row><entry
 align="char">
<para>0</para>
</entry><entry
 align="char">
<para>The function was completed without errors.</para>
</entry>
 </row><row><entry
 align="char">
<para>-EINVAL</para>
</entry><entry
 align="char">
<para>Bad parameter.</para>
</entry>
 </row></tbody></tgroup></informaltable>
 </section></section>
<section id="section_feed_api">
<title>Section Feed API</title>
<para>A section feed is a resource consisting of a PID filter and a set of
section filters. Using this
API, the client can set the properties of a section feed and to start/stop
filtering. The API is
defined as an abstract interface of the type dmx_section_feed_t. The functions
that implement
the interface should be defined static or module private. The client can get the

handle of
a section feed API by calling the function allocate_section_feed() in the demux
API.
</para>
<para>On demux platforms that provide section filtering in hardware, the Section
Feed API
implementation provides a software wrapper for the demux hardware. Other
platforms may
support only PID filtering in hardware, requiring that TS packets are converted
to sections in
software. In the latter case the Section Feed API implementation can be a client
of the TS
Feed API.
</para>

</section>
<section id="kdapi_set">
<title>set()</title>
<para>DESCRIPTION
</para>
<informaltable><tgroup cols="1"><tbody><row><entry
 align="char">
<para>This function sets the parameters of a section feed. Any filtering in
progress on
 the section feed must be stopped before calling this function. If descrambling
 is enabled, the payload_scrambling_control and address_scrambling_control
 fields of received DVB datagram sections should be observed. If either one is
 non-zero, the section should be descrambled either in hardware or using the
 functions descramble_mac_address() and descramble_section_payload() of the
 demux API. Note that according to the MPEG-2 Systems specification, only
 the payloads of private sections can be scrambled while the rest of the section
 data must be sent in the clear.</para>
</entry>
 </row></tbody></tgroup></informaltable>
<para>SYNOPSIS
</para>
<informaltable><tgroup cols="1"><tbody><row><entry
 align="char">
<para>int set(dmx_section_feed_t⋆ feed, __u16 pid, size_t
 circular_buffer_size, int descramble, int
 check_crc);</para>
</entry>
 </row></tbody></tgroup></informaltable>
<para>PARAMETERS
</para>
<informaltable><tgroup cols="2"><tbody><row><entry
 align="char">
<para>dmx_section_feed_t*
 feed</para>
</entry><entry
 align="char">
<para>Pointer to the section feed API and instance data.</para>
</entry>
 </row><row><entry
 align="char">
<para>__u16 pid</para>

```
</entry><entry
 align="char">
<para>PID value to filter; only the TS packets carrying the
 specified PID will be accepted.</para>
</entry>
 </row><row><entry
 align="char">
<para>size_t
 circular_buffer_size</para>
</entry><entry
 align="char">
<para>Size of the circular buffer for filtered sections.</para>
</entry>
 </row><row><entry
 align="char">
<para>int descramble</para>
</entry><entry
 align="char">
<para>If non-zero, descramble any sections that are scrambled.</para>
</entry>
 </row><row><entry
 align="char">
<para>int check_crc</para>
</entry><entry
 align="char">
<para>If non-zero, check the CRC values of filtered sections.</para>
</entry>
 </row></tbody></tgroup></informaltable>
<para>RETURNS
</para>
<informaltable><tgroup cols="2"><tbody><row><entry
 align="char">
<para>0</para>
</entry><entry
 align="char">
<para>The function was completed without errors.</para>
</entry>
 </row><row><entry
 align="char">
<para>-ENOMEM</para>
</entry><entry
 align="char">
<para>Not enough memory for the requested buffer size.</para>
</entry>
 </row><row><entry
 align="char">
<para>-ENOSYS</para>
</entry><entry
 align="char">
<para>No descrambling facility available for sections.</para>
</entry>
 </row><row><entry
 align="char">
<para>-EINVAL</para>
</entry><entry
 align="char">
```

<para>Bad parameters.</para>
</entry>
 </row></tbody></tgroup></informaltable>

</section><section
role="subsection"><title>allocate_filter()</title>
<para>DESCRIPTION
</para>
<informaltable><tgroup cols="1"><tbody><row><entry
 align="char">
<para>This function is used to allocate a section filter on the demux. It should only be
 called when no filtering is in progress on this section feed. If a filter cannot be
 allocated, the function fails with -ENOSPC. See in section ?? for the format of
 the section filter.</para>
</entry>
 </row><row><entry
 align="char">
<para>The bitfields filter_mask and filter_value should only be modified when no
 filtering is in progress on this section feed. filter_mask controls which bits of
 filter_value are compared with the section headers/payload. On a binary value
 of 1 in filter_mask, the corresponding bits are compared. The filter only accepts
 sections that are equal to filter_value in all the tested bit positions. Any changes
 to the values of filter_mask and filter_value are guaranteed to take effect only
 when the start_filtering() function is called next time. The parent pointer in
 the struct is initialized by the API implementation to the value of the feed
 parameter. The priv pointer is not used by the API implementation, and can
 thus be freely utilized by the caller of this function. Any data pointed to by the
 priv pointer is available to the recipient of the dmx_section_cb() function call.</para>
</entry>
 </row><row><entry
 align="char">
<para>While the maximum section filter length (DMX_MAX_FILTER_SIZE) is
 currently set at 16 bytes, hardware filters of that size are not available on all
 platforms. Therefore, section filtering will often take place first in hardware,
 followed by filtering in software for the header bytes that were not covered
 by a hardware filter. The filter_mask field can be checked to determine how
 many bytes of the section filter are actually used, and if the hardware filter will
 suffice. Additionally, software-only section filters can optionally be allocated
 to clients when all hardware section filters are in use. Note that on most demux
 hardware it is not possible to filter on the section_length field of the section
 header – thus this field is ignored, even though it is included in filter_value and

filter_mask fields.</para>
</entry>
</row></tbody></tgroup></informaltable>
<para>SYNOPSIS
</para>
<informaltable><tgroup cols="1"><tbody><row><entry
align="char">
<para>int allocate_filter(dmx_section_feed_t⋆ feed,
dmx_section_filter_t⋆⋆ filter);</para>
</entry>
</row></tbody></tgroup></informaltable>
<para>PARAMETERS
</para>
<informaltable><tgroup cols="2"><tbody><row><entry
align="char">
<para>dmx_section_feed_t*
feed</para>
</entry><entry
align="char">
<para>Pointer to the section feed API and instance data.</para>
</entry>
</row><row><entry
align="char">
<para>dmx_section_filter_t**
filter</para>
</entry><entry
align="char">
<para>Pointer to the allocated filter.</para>
</entry>
</row></tbody></tgroup></informaltable>
<para>RETURNS
</para>
<informaltable><tgroup cols="2"><tbody><row><entry
align="char">
<para>0</para>
</entry><entry
align="char">
<para>The function was completed without errors.</para>
</entry>
</row><row><entry
align="char">
<para>-ENOSPC</para>
</entry><entry
align="char">
<para>No filters of given type and length available.</para>
</entry>
</row><row><entry
align="char">
<para>-EINVAL</para>
</entry><entry
align="char">
<para>Bad parameters.</para>
</entry>
</row></tbody></tgroup></informaltable>

</section><section

role="subsection"><title>release_filter()</title>
<para>DESCRIPTION
</para>
<informaltable><tgroup cols="1"><tbody><row><entry
 align="char">
<para>This function releases all the resources of a previously allocated section
filter.
 The function should not be called while filtering is in progress on this
section
 feed. After calling this function, the caller should not try to dereference the
 filter pointer.</para>
</entry>
 </row></tbody></tgroup></informaltable>
<para>SYNOPSIS
</para>
<informaltable><tgroup cols="1"><tbody><row><entry
 align="char">
<para>int release_filter (dmx_section_feed_t⋆ feed,
 dmx_section_filter_t⋆ filter);</para>
</entry>
 </row></tbody></tgroup></informaltable>
<para>PARAMETERS
</para>
<informaltable><tgroup cols="2"><tbody><row><entry
 align="char">
<para>dmx_section_feed_t*
 feed</para>
</entry><entry
 align="char">
<para>Pointer to the section feed API and instance data.</para>
</entry>
 </row><row><entry
 align="char">
<para>dmx_section_filter_t*
 filter</para>
</entry><entry
 align="char">
<para>I/O Pointer to the instance data of a section filter.</para>
</entry>
 </row></tbody></tgroup></informaltable>
<para>RETURNS
</para>
<informaltable><tgroup cols="2"><tbody><row><entry
 align="char">
<para>0</para>
</entry><entry
 align="char">
<para>The function was completed without errors.</para>
</entry>
 </row><row><entry
 align="char">
<para>-ENODEV</para>
</entry><entry
 align="char">
<para>No such filter allocated.</para>
</entry>

```
 </row><row><entry
 align="char">
<para>-EINVAL</para>
</entry><entry
 align="char">
<para>Bad parameter.</para>
</entry>
 </row></tbody></tgroup></informaltable>

</section><section
role="subsection"><title>start_filtering()</title>
<para>DESCRIPTION
</para>
<informaltable><tgroup cols="1"><tbody><row><entry
 align="char">
<para>Starts filtering sections on this section feed, according to its settings.
Sections
 are first filtered based on their PID and then matched with the section
 filters allocated for this feed. If the section matches the PID filter and
 at least one section filter, it is delivered to the API client. The section
 is delivered asynchronously using the callback function registered with
 allocate_section_feed().</para>
</entry>
 </row></tbody></tgroup></informaltable>
<para>SYNOPSIS
</para>
<informaltable><tgroup cols="1"><tbody><row><entry
 align="char">
<para>int start_filtering ( dmx_section_feed_t&amp;#x22C6; feed );</para>
</entry>
 </row></tbody></tgroup></informaltable>
<para>PARAMETERS
</para>
<informaltable><tgroup cols="2"><tbody><row><entry
 align="char">
<para>dmx_section_feed_t*
 feed</para>
</entry><entry
 align="char">
<para>Pointer to the section feed API and instance data.</para>
</entry>
 </row></tbody></tgroup></informaltable>
<para>RETURNS
</para>
<informaltable><tgroup cols="2"><tbody><row><entry
 align="char">
<para>0</para>
</entry><entry
 align="char">
<para>The function was completed without errors.</para>
</entry>
 </row><row><entry
 align="char">
<para>-EINVAL</para>
</entry><entry
 align="char">
```

```
<para>Bad parameter.</para>
</entry>
 </row></tbody></tgroup></informaltable>

</section><section
role="subsection"><title>stop_filtering()</title>
<para>DESCRIPTION
</para>
<informaltable><tgroup cols="1"><tbody><row><entry
 align="char">
<para>Stops filtering sections on this section feed. Note that any changes to
the
 filtering parameters (filter_value, filter_mask, etc.) should only be made when
 filtering is stopped.</para>
</entry>
 </row></tbody></tgroup></informaltable>
<para>SYNOPSIS
</para>
<informaltable><tgroup cols="1"><tbody><row><entry
 align="char">
<para>int stop_filtering ( dmx_section_feed_t&amp;#x22C6; feed );</para>
</entry>
 </row></tbody></tgroup></informaltable>
<para>PARAMETERS
</para>
<informaltable><tgroup cols="2"><tbody><row><entry
 align="char">
<para>dmx_section_feed_t*
 feed</para>
</entry><entry
 align="char">
<para>Pointer to the section feed API and instance data.</para>
</entry>
 </row></tbody></tgroup></informaltable>
<para>RETURNS
</para>
<informaltable><tgroup cols="2"><tbody><row><entry
 align="char">
<para>0</para>
</entry><entry
 align="char">
<para>The function was completed without errors.</para>
</entry>
 </row><row><entry
 align="char">
<para>-EINVAL</para>
</entry><entry
 align="char">
<para>Bad parameter.</para>
</entry>
 </row></tbody></tgroup></informaltable>

</section>
```