```
<title>DVB Frontend API</title>

<para>The DVB frontend device controls the tuner and DVB demodulator
hardware. It can be accessed through <emphasis
role="tt">/dev/dvb/adapter0/frontend0</emphasis>. Data types and and
ioctl definitions can be accessed by including <emphasis
role="tt">linux/dvb/frontend.h</emphasis> in your application.</para>

<para>DVB frontends come in three varieties: DVB-S (satellite), DVB-C
(cable) and DVB-T (terrestrial). Transmission via the internet (DVB-IP)
is not yet handled by this API but a future extension is possible. For
DVB-S the frontend device also supports satellite equipment control
(SEC) via DiSEqC and V-SEC protocols. The DiSEqC (digital SEC)
specification is available from
<ulink
url="http://www.eutelsat.com/satellites/4_5_5.html">Eutelsat</ulink>.</para>

<para>Note that the DVB API may also be used for MPEG decoder-only PCI
cards, in which case there exists no frontend device.</para>

<section id="frontend_types">
<title>Frontend Data Types</title>

<section id="frontend_type">
<title>frontend type</title>

<para>For historical reasons frontend types are named after the type of
modulation used in
transmission.</para>
<programlisting>
        typedef enum fe_type {
        FE_QPSK,   /&#x22C6; DVB-S &#x22C6;/
        FE_QAM,    /&#x22C6; DVB-C &#x22C6;/
        FE_OFDM    /&#x22C6; DVB-T &#x22C6;/
        } fe_type_t;
</programlisting>

</section>

<section id="frontend_caps">
<title>frontend capabilities</title>

<para>Capabilities describe what a frontend can do. Some capabilities can only
be supported for
a specific frontend type.</para>
<programlisting>
        typedef enum fe_caps {
        FE_IS_STUPID                 = 0,
        FE_CAN_INVERSION_AUTO        = 0x1,
        FE_CAN_FEC_1_2               = 0x2,
        FE_CAN_FEC_2_3               = 0x4,
        FE_CAN_FEC_3_4               = 0x8,
        FE_CAN_FEC_4_5               = 0x10,
        FE_CAN_FEC_5_6               = 0x20,
        FE_CAN_FEC_6_7               = 0x40,
        FE_CAN_FEC_7_8               = 0x80,
        FE_CAN_FEC_8_9               = 0x100,
        FE_CAN_FEC_AUTO              = 0x200,
        FE_CAN_QPSK                  = 0x400,
        FE_CAN_QAM_16                = 0x800,
        FE_CAN_QAM_32                = 0x1000,
        FE_CAN_QAM_64                = 0x2000,
        FE_CAN_QAM_128               = 0x4000,
        FE_CAN_QAM_256               = 0x8000,
        FE_CAN_QAM_AUTO              = 0x10000,
        FE_CAN_TRANSMISSION_MODE_AUTO = 0x20000,
        FE_CAN_BANDWIDTH_AUTO        = 0x40000,
        FE_CAN_GUARD_INTERVAL_AUTO   = 0x80000,
        FE_CAN_HIERARCHY_AUTO        = 0x100000,
        FE_CAN_MUTE_TS               = 0x80000000,
        FE_CAN_CLEAN_SETUP           = 0x40000000
        } fe_caps_t;
</programlisting>
</section>

<section id="frontend_info">
<title>frontend information</title>

<para>Information about the frontend ca be queried with
        <link linkend="FE_GET_INFO">FE_GET_INFO</link>.</para>

<programlisting>
        struct dvb_frontend_info {
        char       name[128];
        fe_type_t  type;
        uint32_t   frequency_min;
        uint32_t   frequency_max;
        uint32_t   frequency_stepsize;
        uint32_t   frequency_tolerance;
        uint32_t   symbol_rate_min;
        uint32_t   symbol_rate_max;
        uint32_t   symbol_rate_tolerance;     /&#x22C6; ppm &#x22C6;/
        uint32_t   notifier_delay;            /&#x22C6; ms &#x22C6;/
        fe_caps_t  caps;
        };
</programlisting>
</section>

<section id="frontend_diseqc">
<title>diseqc master command</title>

<para>A message sent from the frontend to DiSEqC capable equipment.</para>
<programlisting>
        struct dvb_diseqc_master_cmd {
        uint8_t msg [6]; /&#x22C6;  { framing, address, command, data[3] }
&#x22C6;/
        uint8_t msg_len; /&#x22C6;  valid values are 3...6  &#x22C6;/
        };
</programlisting>
</section>
<section role="subsection">
<title>diseqc slave reply</title>
```

```
<para>A reply to the frontend from DiSEqC 2.0 capable equipment.</para>
<programlisting>
        struct dvb_diseqc_slave_reply {
        uint8_t msg [4]; /&#x22C6;  { framing, data [3] } &#x22C6;/
        uint8_t msg_len; /&#x22C6;  valid values are 0...4, 0 means no msg
&#x22C6;/
        int     timeout; /&#x22C6;  return from ioctl after timeout ms with
&#x22C6;/
        };                      /&#x22C6;  errorcode when no message was
received  &#x22C6;/
</programlisting>
</section>

<section id="frontend_diseqc_slave_reply">
<title>diseqc slave reply</title>
<para>The voltage is usually used with non-DiSEqC capable LNBs to switch the
polarzation
(horizontal/vertical). When using DiSEqC epuipment this voltage has to be
switched
consistently to the DiSEqC commands as described in the DiSEqC spec.</para>
<programlisting>
        typedef enum fe_sec_voltage {
        SEC_VOLTAGE_13,
        SEC_VOLTAGE_18
        } fe_sec_voltage_t;
</programlisting>
</section>

<section id="frontend_sec_tone">
<title>SEC continuous tone</title>

<para>The continous 22KHz tone is usually used with non-DiSEqC capable LNBs to
switch the
high/low band of a dual-band LNB. When using DiSEqC epuipment this voltage has
to
be switched consistently to the DiSEqC commands as described in the DiSEqC
spec.</para>
<programlisting>
        typedef enum fe_sec_tone_mode {
        SEC_TONE_ON,
        SEC_TONE_OFF
        } fe_sec_tone_mode_t;
</programlisting>
</section>

<section id="frontend_sec_burst">
<title>SEC tone burst</title>

<para>The 22KHz tone burst is usually used with non-DiSEqC capable switches to
select
between two connected LNBs/satellites. When using DiSEqC epuipment this voltage
has to
be switched consistently to the DiSEqC commands as described in the DiSEqC
spec.</para>
<programlisting>
```

```
        typedef enum fe_sec_mini_cmd {
        SEC_MINI_A,
        SEC_MINI_B
        } fe_sec_mini_cmd_t;
</programlisting>

<para></para>
</section>

<section id="frontend_status">
<title>frontend status</title>
<para>Several functions of the frontend device use the fe_status data type
defined
by</para>
<programlisting>
 typedef enum fe_status {
         FE_HAS_SIGNAL     = 0x01,   /&#x22C6;  found something above the noise
level &#x22C6;/
         FE_HAS_CARRIER    = 0x02,   /&#x22C6;  found a DVB signal  &#x22C6;/
         FE_HAS_VITERBI    = 0x04,   /&#x22C6;  FEC is stable  &#x22C6;/
         FE_HAS_SYNC       = 0x08,   /&#x22C6;  found sync bytes  &#x22C6;/
         FE_HAS_LOCK       = 0x10,   /&#x22C6;  everything's working...
&#x22C6;/
         FE_TIMEDOUT       = 0x20,   /&#x22C6;  no lock within the last ~2
seconds &#x22C6;/
         FE_REINIT         = 0x40    /&#x22C6;  frontend was reinitialized,
&#x22C6;/
 } fe_status_t;                      /&#x22C6;  application is recommned to
reset &#x22C6;/
</programlisting>
<para>to indicate the current state and/or state changes of the frontend
hardware.
</para>

</section>

<section id="frontend_params">
<title>frontend parameters</title>
<para>The kind of parameters passed to the frontend device for tuning depend on
the kind of hardware you are using. All kinds of parameters are combined as an
union in the FrontendParameters structure:</para>
<programlisting>
 struct dvb_frontend_parameters {
         uint32_t frequency;       /&#x22C6; (absolute) frequency in Hz for
QAM/OFDM &#x22C6;/
                                   /&#x22C6; intermediate frequency in kHz for
QPSK &#x22C6;/
         fe_spectral_inversion_t inversion;
         union {
                 struct dvb_qpsk_parameters qpsk;
                 struct dvb_qam_parameters  qam;
                 struct dvb_ofdm_parameters ofdm;
         } u;
 };
</programlisting>
<para>For satellite QPSK frontends you have to use the
```

```
<constant>QPSKParameters</constant> member defined by</para>
<programlisting>
 struct dvb_qpsk_parameters {
         uint32_t        symbol_rate;  /&#x22C6; symbol rate in Symbols per
second &#x22C6;/
         fe_code_rate_t  fec_inner;    /&#x22C6; forward error correction (see
above) &#x22C6;/
 };
</programlisting>
<para>for cable QAM frontend you use the <constant>QAMParameters</constant>
structure</para>
<programlisting>
 struct dvb_qam_parameters {
         uint32_t         symbol_rate; /&#x22C6; symbol rate in Symbols per
second &#x22C6;/
         fe_code_rate_t   fec_inner;   /&#x22C6; forward error correction (see
above) &#x22C6;/
         fe_modulation_t  modulation;  /&#x22C6; modulation type (see above)
&#x22C6;/
 };
</programlisting>
<para>DVB-T frontends are supported by the <constant>OFDMParamters</constant>
structure
</para>
<programlisting>
 struct dvb_ofdm_parameters {
         fe_bandwidth_t      bandwidth;
         fe_code_rate_t      code_rate_HP;  /&#x22C6; high priority stream code
rate &#x22C6;/
         fe_code_rate_t      code_rate_LP;  /&#x22C6; low priority stream code
rate &#x22C6;/
         fe_modulation_t     constellation; /&#x22C6; modulation type (see
above) &#x22C6;/
         fe_transmit_mode_t  transmission_mode;
         fe_guard_interval_t guard_interval;
         fe_hierarchy_t      hierarchy_information;
 };
</programlisting>
<para>In the case of QPSK frontends the <constant>Frequency</constant> field
specifies the intermediate
frequency, i.e. the offset which is effectively added to the local oscillator
frequency (LOF) of
the LNB. The intermediate frequency has to be specified in units of kHz. For QAM
and
OFDM frontends the Frequency specifies the absolute frequency and is given in
Hz.
</para>
<para>The Inversion field can take one of these values:
</para>
<programlisting>
 typedef enum fe_spectral_inversion {
         INVERSION_OFF,
         INVERSION_ON,
         INVERSION_AUTO
 } fe_spectral_inversion_t;
</programlisting>
```

```
<para>It indicates if spectral inversion should be presumed or not. In the
automatic setting
(<constant>INVERSION_AUTO</constant>) the hardware will try to figure out the
correct setting by
itself.
</para>
<para>The possible values for the <constant>FEC_inner</constant> field are
</para>
<programlisting>
 typedef enum fe_code_rate {
         FEC_NONE = 0,
         FEC_1_2,
         FEC_2_3,
         FEC_3_4,
         FEC_4_5,
         FEC_5_6,
         FEC_6_7,
         FEC_7_8,
         FEC_8_9,
         FEC_AUTO
 } fe_code_rate_t;
</programlisting>
<para>which correspond to error correction rates of 1/2, 2/3, etc., no error
correction or auto
detection.
</para>
<para>For cable and terrestrial frontends (QAM and OFDM) one also has to specify
the quadrature
modulation mode which can be one of the following:
</para>
<programlisting>
 typedef enum fe_modulation {
 QPSK,
         QAM_16,
         QAM_32,
         QAM_64,
         QAM_128,
         QAM_256,
         QAM_AUTO
 } fe_modulation_t;
</programlisting>
<para>Finally, there are several more parameters for OFDM:
</para>
<programlisting>
 typedef enum fe_transmit_mode {
         TRANSMISSION_MODE_2K,
         TRANSMISSION_MODE_8K,
         TRANSMISSION_MODE_AUTO
 } fe_transmit_mode_t;
</programlisting>
 <programlisting>
 typedef enum fe_bandwidth {
         BANDWIDTH_8_MHZ,
         BANDWIDTH_7_MHZ,
         BANDWIDTH_6_MHZ,
         BANDWIDTH_AUTO
 } fe_bandwidth_t;
</programlisting>
 <programlisting>
 typedef enum fe_guard_interval {
         GUARD_INTERVAL_1_32,
         GUARD_INTERVAL_1_16,
         GUARD_INTERVAL_1_8,
         GUARD_INTERVAL_1_4,
         GUARD_INTERVAL_AUTO
 } fe_guard_interval_t;
</programlisting>
 <programlisting>
 typedef enum fe_hierarchy {
         HIERARCHY_NONE,
         HIERARCHY_1,
         HIERARCHY_2,
         HIERARCHY_4,
         HIERARCHY_AUTO
 } fe_hierarchy_t;
</programlisting>

</section>

<section id="frontend_events">
<title>frontend events</title>
 <programlisting>
 struct dvb_frontend_event {
         fe_status_t status;
         struct dvb_frontend_parameters parameters;
 };
</programlisting>
 </section>
</section>


<section id="frontend_fcalls">
<title>Frontend Function Calls</title>

<section id="frontend_f_open">
<title>open()</title>
<para>DESCRIPTION</para>
<informaltable><tgroup cols="1"><tbody><row>
<entry align="char">
<para>This system call opens a named frontend device
(/dev/dvb/adapter0/frontend0)
 for subsequent use. Usually the first thing to do after a successful open is to
 find out the frontend type with <link
linkend="FE_GET_INFO">FE_GET_INFO</link>.</para>
<para>The device can be opened in read-only mode, which only allows monitoring
of
 device status and statistics, or read/write mode, which allows any kind of use
 (e.g. performing tuning operations.)
</para>
<para>In a system with multiple front-ends, it is usually the case that multiple
devices
 cannot be open in read/write mode simultaneously. As long as a front-end
 device is opened in read/write mode, other open() calls in read/write mode will
 either fail or block, depending on whether non-blocking or blocking mode was
 specified. A front-end device opened in blocking mode can later be put into
 non-blocking mode (and vice versa) using the F_SETFL command of the fcntl
 system call. This is a standard system call, documented in the Linux manual
 page for fcntl. When an open() call has succeeded, the device will be ready
 for use in the specified mode. This implies that the corresponding hardware is
 powered up, and that other front-ends may have been powered down to make
 that possible.</para>
</entry>
 </row></tbody></tgroup></informaltable>

<para>SYNOPSIS</para>
<informaltable><tgroup cols="1"><tbody><row><entry
 align="char">
<para>int open(const char &#x22C6;deviceName, int flags);</para>
</entry>
 </row></tbody></tgroup></informaltable>
<para>PARAMETERS
</para>
<informaltable><tgroup cols="2"><tbody><row><entry
 align="char">
<para>const char
 *deviceName</para>
</entry><entry
 align="char">
<para>Name of specific video device.</para>
</entry>
 </row><row><entry
 align="char">
<para>int flags</para>
</entry><entry
 align="char">
<para>A bit-wise OR of the following flags:</para>
</entry>
 </row><row><entry
 align="char">
</entry><entry
 align="char">
<para>O_RDONLY read-only access</para>
</entry>
 </row><row><entry
 align="char">
</entry><entry
 align="char">
<para>O_RDWR read/write access</para>
</entry>
 </row><row><entry
 align="char">
</entry><entry
 align="char">
<para>O_NONBLOCK open in non-blocking mode</para>
</entry>
 </row><row><entry
 align="char">
</entry><entry
```

```
 align="char">
<para>(blocking mode is the default)</para>
</entry>
 </row></tbody></tgroup></informaltable>
<para>ERRORS
</para>
<informaltable><tgroup cols="2"><tbody><row><entry
 align="char">
<para>ENODEV</para>
</entry><entry
 align="char">
<para>Device driver not loaded/available.</para>
</entry>
 </row><row><entry
 align="char">
<para>EINTERNAL</para>
</entry><entry
 align="char">
<para>Internal error.</para>
</entry>
 </row><row><entry
 align="char">
<para>EBUSY</para>
</entry><entry
 align="char">
<para>Device or resource busy.</para>
</entry>
 </row><row><entry
 align="char">
<para>EINVAL</para>
</entry><entry
 align="char">
<para>Invalid argument.</para>
</entry>
 </row></tbody></tgroup></informaltable>
</section>

<section id="frontend_f_close">
<title>close()</title>
<para>DESCRIPTION
</para>
<informaltable><tgroup cols="1"><tbody><row><entry
 align="char">
<para>This system call closes a previously opened front-end device. After
closing
 a front-end device, its corresponding hardware might be powered down
 automatically.</para>
</entry>
 </row></tbody></tgroup></informaltable>
<para>SYNOPSIS
</para>
<informaltable><tgroup cols="1"><tbody><row><entry
 align="char">
<para>int close(int fd);</para>
</entry>
 </row></tbody></tgroup></informaltable>
```

```
<para>PARAMETERS
</para>
<informaltable><tgroup cols="2"><tbody><row><entry
 align="char">
<para>int fd</para>
</entry><entry
 align="char">
<para>File descriptor returned by a previous call to open().</para>
</entry>
 </row></tbody></tgroup></informaltable>
<para>ERRORS
</para>
<informaltable><tgroup cols="2"><tbody><row><entry
 align="char">
<para>EBADF</para>
</entry><entry
 align="char">
<para>fd is not a valid open file descriptor.</para>
</entry>
 </row></tbody></tgroup></informaltable>
</section>

<section id="FE_READ_STATUS">
<title>FE_READ_STATUS</title>
<para>DESCRIPTION
</para>
<informaltable><tgroup cols="1"><tbody><row><entry
 align="char">
<para>This ioctl call returns status information about the front-end. This call
only
 requires read-only access to the device.</para>
</entry>
 </row></tbody></tgroup></informaltable>
<para>SYNOPSIS
</para>
<informaltable><tgroup cols="1"><tbody><row><entry
 align="char">
<para>int ioctl(int fd, int request = <link
linkend="FE_READ_STATUS">FE_READ_STATUS</link>,
 fe_status_t &#x22C6;status);</para>
</entry>
 </row></tbody></tgroup></informaltable>
<para>PARAMETERS
</para>

<informaltable><tgroup cols="2"><tbody><row><entry
 align="char">
<para>int fd</para>
</entry><entry
 align="char">
<para>File descriptor returned by a previous call to open().</para>
</entry>
 </row><row><entry
 align="char">
<para>int request</para>
</entry><entry
```

```
 align="char">
<para>Equals <link linkend="FE_READ_STATUS">FE_READ_STATUS</link> for this
command.</para>
</entry>
 </row><row><entry
 align="char">
<para>struct fe_status_t
 *status</para>
</entry><entry
 align="char">
<para>Points to the location where the front-end status word is
 to be stored.</para>
</entry>
 </row></tbody></tgroup></informaltable>
<para>ERRORS
</para>
<informaltable><tgroup cols="2"><tbody><row><entry
 align="char">
<para>EBADF</para>
</entry><entry
 align="char">
<para>fd is not a valid open file descriptor.</para>
</entry>
 </row><row><entry
 align="char">
<para>EFAULT</para>
</entry><entry
 align="char">
<para>status points to invalid address.</para>
</entry>
 </row></tbody></tgroup></informaltable>
</section>

<section id="FE_READ_BER">
<title>FE_READ_BER</title>
<para>DESCRIPTION
</para>
<informaltable><tgroup cols="1"><tbody><row><entry
 align="char">
<para>This ioctl call returns the bit error rate for the signal currently
 received/demodulated by the front-end. For this command, read-only access to
 the device is sufficient.</para>
</entry>
 </row></tbody></tgroup></informaltable>
<para>SYNOPSIS
</para>
<informaltable><tgroup cols="1"><tbody><row><entry
 align="char">
<para>int ioctl(int fd, int request = <link
linkend="FE_READ_BER">FE_READ_BER</link>,
 uint32_t &#x22C6;ber);</para>
</entry>
 </row></tbody></tgroup></informaltable>
<para>PARAMETERS
</para>
<informaltable><tgroup cols="2"><tbody><row><entry
```

```
 align="char">
<para>int fd</para>
</entry><entry
 align="char">
<para>File descriptor returned by a previous call to open().</para>
</entry>
 </row><row><entry
 align="char">
<para>int request</para>
</entry><entry
 align="char">
<para>Equals <link linkend="FE_READ_BER">FE_READ_BER</link> for this
command.</para>
</entry>
 </row><row><entry
 align="char">
<para>uint32_t *ber</para>
</entry><entry
 align="char">
<para>The bit error rate is stored into *ber.</para>
</entry>
 </row></tbody></tgroup></informaltable>
<para>ERRORS
</para>
<informaltable><tgroup cols="2"><tbody><row><entry
 align="char">
<para>EBADF</para>
</entry><entry
 align="char">
<para>fd is not a valid open file descriptor.</para>
</entry>
 </row><row><entry
 align="char">
<para>EFAULT</para>
</entry><entry
 align="char">
<para>ber points to invalid address.</para>
</entry>
 </row><row><entry
 align="char">
<para>ENOSIGNAL</para>
</entry><entry
 align="char">
<para>There is no signal, thus no meaningful bit error rate. Also
 returned if the front-end is not turned on.</para>
</entry>
 </row><row><entry
 align="char">
<para>ENOSYS</para>
</entry><entry
 align="char">
<para>Function not available for this device.</para>
</entry>
 </row></tbody></tgroup></informaltable>
</section>
```

```
<section id="FE_READ_SNR">
<title>FE_READ_SNR</title>

<para>DESCRIPTION
</para>
<informaltable><tgroup cols="1"><tbody><row><entry
 align="char">
<para>This ioctl call returns the signal-to-noise ratio for the signal currently
received
 by the front-end. For this command, read-only access to the device is
sufficient.</para>
</entry>
 </row></tbody></tgroup></informaltable>
<para>SYNOPSIS
</para>
<informaltable><tgroup cols="1"><tbody><row><entry
 align="char">
<para>int ioctl(int fd, int request = <link
linkend="FE_READ_SNR">FE_READ_SNR</link>, int16_t
 &#x22C6;snr);</para>
</entry>
 </row></tbody></tgroup></informaltable>
<para>PARAMETERS
</para>
<informaltable><tgroup cols="2"><tbody><row><entry
 align="char">
<para>int fd</para>
</entry><entry
 align="char">
<para>File descriptor returned by a previous call to open().</para>
</entry>
 </row><row><entry
 align="char">
<para>int request</para>
</entry><entry
 align="char">
<para>Equals <link linkend="FE_READ_SNR">FE_READ_SNR</link> for this
command.</para>
</entry>
 </row><row><entry
 align="char">
<para>int16_t *snr</para>
</entry><entry
 align="char">
<para>The signal-to-noise ratio is stored into *snr.</para>
</entry>
 </row></tbody></tgroup></informaltable>

<para>ERRORS
</para>
<informaltable><tgroup cols="2"><tbody><row><entry
 align="char">
<para>EBADF</para>
</entry><entry
 align="char">
<para>fd is not a valid open file descriptor.</para>
```

```
</entry>
 </row><row><entry
 align="char">
<para>EFAULT</para>
</entry><entry
 align="char">
<para>snr points to invalid address.</para>
</entry>
 </row><row><entry
 align="char">
<para>ENOSIGNAL</para>
</entry><entry
 align="char">
<para>There is no signal, thus no meaningful signal strength
 value. Also returned if front-end is not turned on.</para>
</entry>
 </row><row><entry
 align="char">
<para>ENOSYS</para>
</entry><entry
 align="char">
<para>Function not available for this device.</para>
</entry>
 </row></tbody></tgroup></informaltable>
</section>

<section id="FE_READ_SIGNAL_STRENGTH">
<title>FE_READ_SIGNAL_STRENGTH</title>
<para>DESCRIPTION
</para>
<informaltable><tgroup cols="1"><tbody><row><entry
 align="char">
<para>This ioctl call returns the signal strength value for the signal currently
received
 by the front-end. For this command, read-only access to the device is
sufficient.</para>
</entry>
 </row></tbody></tgroup></informaltable>
<para>SYNOPSIS
</para>
<informaltable><tgroup cols="1"><tbody><row><entry
 align="char">
<para>int ioctl( int fd, int request =
 <link linkend="FE_READ_SIGNAL_STRENGTH">FE_READ_SIGNAL_STRENGTH</link>, int16_t
&#x22C6;strength);</para>
</entry>
 </row></tbody></tgroup></informaltable>

<para>PARAMETERS
</para>
<informaltable><tgroup cols="2"><tbody><row><entry
 align="char">
<para>int fd</para>
</entry><entry
 align="char">
<para>File descriptor returned by a previous call to open().</para>
```

```
</entry>
 </row><row><entry
 align="char">
<para>int request</para>
</entry><entry
 align="char">
<para>Equals <link
linkend="FE_READ_SIGNAL_STRENGTH">FE_READ_SIGNAL_STRENGTH</link> for this
 command.</para>
</entry>
 </row><row><entry
 align="char">
<para>int16_t *strength</para>
</entry><entry
 align="char">
<para>The signal strength value is stored into *strength.</para>
</entry>
 </row></tbody></tgroup></informaltable>
<para>ERRORS
</para>
<informaltable><tgroup cols="2"><tbody><row><entry
 align="char">
<para>EBADF</para>
</entry><entry
 align="char">
<para>fd is not a valid open file descriptor.</para>
</entry>
 </row><row><entry
 align="char">
<para>EFAULT</para>
</entry><entry
 align="char">
<para>status points to invalid address.</para>
</entry>
 </row><row><entry
 align="char">
<para>ENOSIGNAL</para>
</entry><entry
 align="char">
<para>There is no signal, thus no meaningful signal strength
 value. Also returned if front-end is not turned on.</para>
</entry>
 </row><row><entry
 align="char">
<para>ENOSYS</para>
</entry><entry
 align="char">
<para>Function not available for this device.</para>
</entry>
 </row></tbody></tgroup></informaltable>
</section>

<section id="FE_READ_UNCORRECTED_BLOCKS">
<title>FE_READ_UNCORRECTED_BLOCKS</title>
<para>DESCRIPTION
</para>
```

```
<informaltable><tgroup cols="1"><tbody><row><entry
 align="char">
<para>This ioctl call returns the number of uncorrected blocks detected by the
device
 driver during its lifetime. For meaningful measurements, the increment in block
 count during a specific time interval should be calculated. For this command,
 read-only access to the device is sufficient.</para>
</entry>
 </row><row><entry
 align="char">
<para>Note that the counter will wrap to zero after its maximum count has been
 reached.</para>
</entry>
 </row></tbody></tgroup></informaltable>
<para>SYNOPSIS
</para>
<informaltable><tgroup cols="1"><tbody><row><entry
 align="char">
<para>int ioctl( int fd, int request =
 <link linkend="FE_READ_UNCORRECTED_BLOCKS">FE_READ_UNCORRECTED_BLOCKS</link>,
uint32_t &#x22C6;ublocks);</para>
</entry>
 </row></tbody></tgroup></informaltable>
<para>PARAMETERS
</para>
<informaltable><tgroup cols="2"><tbody><row><entry
 align="char">
<para>int fd</para>
</entry><entry
 align="char">
<para>File descriptor returned by a previous call to open().</para>
</entry>
 </row><row><entry
 align="char">
<para>int request</para>
</entry><entry
 align="char">
<para>Equals <link
linkend="FE_READ_UNCORRECTED_BLOCKS">FE_READ_UNCORRECTED_BLOCKS</link> for this
 command.</para>
</entry>
 </row><row><entry
 align="char">
<para>uint32_t *ublocks</para>
</entry><entry
 align="char">
<para>The total number of uncorrected blocks seen by the driver
 so far.</para>
</entry>
 </row></tbody></tgroup></informaltable>
<para>ERRORS
</para>
<informaltable><tgroup cols="2"><tbody><row><entry
 align="char">
<para>EBADF</para>
</entry><entry
```

```
 align="char">
<para>fd is not a valid open file descriptor.</para>
</entry>
 </row><row><entry
 align="char">
<para>EFAULT</para>
</entry><entry
 align="char">
<para>ublocks points to invalid address.</para>
</entry>
 </row><row><entry
 align="char">
<para>ENOSYS</para>
</entry><entry
 align="char">
<para>Function not available for this device.</para>
</entry>
 </row></tbody></tgroup></informaltable>
</section>

<section id="FE_SET_FRONTEND">
<title>FE_SET_FRONTEND</title>
<para>DESCRIPTION
</para>
<informaltable><tgroup cols="1"><tbody><row><entry
 align="char">
<para>This ioctl call starts a tuning operation using specified parameters. The
result
 of this call will be successful if the parameters were valid and the tuning
could
 be initiated. The result of the tuning operation in itself, however, will
arrive
 asynchronously as an event (see documentation for <link
linkend="FE_GET_EVENT">FE_GET_EVENT</link> and
 FrontendEvent.) If a new <link linkend="FE_SET_FRONTEND">FE_SET_FRONTEND</link>
operation is initiated before
 the previous one was completed, the previous operation will be aborted in favor
 of the new one. This command requires read/write access to the device.</para>
</entry>
 </row></tbody></tgroup></informaltable>

<para>SYNOPSIS
</para>
<informaltable><tgroup cols="1"><tbody><row><entry
 align="char">
<para>int ioctl(int fd, int request = <link
linkend="FE_SET_FRONTEND">FE_SET_FRONTEND</link>,
 struct dvb_frontend_parameters &#x22C6;p);</para>
</entry>
 </row></tbody></tgroup></informaltable>
<para>PARAMETERS
</para>
<informaltable><tgroup cols="2"><tbody><row><entry
 align="char">
<para>int fd</para>
</entry><entry
```

```
 align="char">
<para>File descriptor returned by a previous call to open().</para>
</entry>
 </row><row><entry
 align="char">
<para>int request</para>
</entry><entry
 align="char">
<para>Equals <link linkend="FE_SET_FRONTEND">FE_SET_FRONTEND</link> for this
command.</para>
</entry>
 </row><row><entry
 align="char">
<para>struct
 dvb_frontend_parameters
 *p</para>
</entry><entry
 align="char">
<para>Points to parameters for tuning operation.</para>
</entry>
 </row></tbody></tgroup></informaltable>
<para>ERRORS
</para>
<informaltable><tgroup cols="2"><tbody><row><entry
 align="char">
<para>EBADF</para>
</entry><entry
 align="char">
<para>fd is not a valid open file descriptor.</para>
</entry>
 </row><row><entry
 align="char">
<para>EFAULT</para>
</entry><entry
 align="char">
<para>p points to invalid address.</para>
</entry>
 </row><row><entry
 align="char">
<para>EINVAL</para>
</entry><entry
 align="char">
<para>Maximum supported symbol rate reached.</para>
</entry>
</row></tbody></tgroup></informaltable>
</section>

<section id="FE_GET_FRONTEND">
<title>FE_GET_FRONTEND</title>
<para>DESCRIPTION
</para>
<informaltable><tgroup cols="1"><tbody><row><entry
 align="char">
<para>This ioctl call queries the currently effective frontend parameters. For
this
 command, read-only access to the device is sufficient.</para>
```

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

<para>SYNOPSIS
</para>
<informaltable><tgroup cols="1"><tbody><row><entry
 align="char">
<para>int ioctl(int fd, int request = <link
linkend="FE_GET_FRONTEND">FE_GET_FRONTEND</link>,
 struct dvb_frontend_parameters &#x22C6;p);</para>
</entry>
 </row></tbody></tgroup></informaltable>

<para>PARAMETERS
</para>
<informaltable><tgroup cols="2"><tbody><row><entry
 align="char">
<para>int fd</para>
</entry><entry
 align="char">
<para>File descriptor returned by a previous call to open().</para>
</entry>
 </row><row><entry
 align="char">
<para>int request</para>
</entry><entry
 align="char">
<para>Equals <link linkend="FE_SET_FRONTEND">FE_SET_FRONTEND</link> for this
command.</para>
</entry>
 </row><row><entry
 align="char">
<para>struct
 dvb_frontend_parameters
 *p</para>
</entry><entry
 align="char">
<para>Points to parameters for tuning operation.</para>
</entry>
 </row></tbody></tgroup></informaltable>

<para>ERRORS
</para>

<informaltable><tgroup cols="2"><tbody><row><entry
 align="char">
<para>EBADF</para>
</entry><entry
 align="char">
<para>fd is not a valid open file descriptor.</para>
</entry>
 </row><row><entry
 align="char">
<para>EFAULT</para>
</entry><entry
 align="char">
```

```
<para>p points to invalid address.</para>
</entry>
 </row><row><entry
 align="char">
<para>EINVAL</para>
</entry><entry
 align="char">
<para>Maximum supported symbol rate reached.</para>
</entry>
 </row></tbody></tgroup></informaltable>

</section>

<section id="FE_GET_EVENT">
<title>FE_GET_EVENT</title>
<para>DESCRIPTION
</para>
<informaltable><tgroup cols="1"><tbody><row><entry
 align="char">
<para>This ioctl call returns a frontend event if available. If an event is not
 available, the behavior depends on whether the device is in blocking or
 non-blocking mode. In the latter case, the call fails immediately with errno
 set to EWOULDBLOCK. In the former case, the call blocks until an event
 becomes available.</para>
</entry>
 </row><row><entry
 align="char">
<para>The standard Linux poll() and/or select() system calls can be used with
the
 device file descriptor to watch for new events. For select(), the file
descriptor
 should be included in the exceptfds argument, and for poll(), POLLPRI should
 be specified as the wake-up condition. Since the event queue allocated is
 rather small (room for 8 events), the queue must be serviced regularly to avoid
 overflow. If an overflow happens, the oldest event is discarded from the queue,
 and an error (EOVERFLOW) occurs the next time the queue is read. After
 reporting the error condition in this fashion, subsequent
 <link linkend="FE_GET_EVENT">FE_GET_EVENT</link>
 calls will return events from the queue as usual.</para>
</entry>
 </row><row><entry
 align="char">
<para>For the sake of implementation simplicity, this command requires
read/write
 access to the device.</para>
</entry>
 </row></tbody></tgroup></informaltable>

<para>SYNOPSIS
</para>
<informaltable><tgroup cols="1"><tbody><row><entry
 align="char">
<para>int ioctl(int fd, int request = QPSK_GET_EVENT,
 struct dvb_frontend_event &#x22C6;ev);</para>
</entry>
 </row></tbody></tgroup></informaltable>
```

```
<para>PARAMETERS
</para>
<informaltable><tgroup cols="2"><tbody><row><entry
 align="char">
<para>int fd</para>
</entry><entry
 align="char">
<para>File descriptor returned by a previous call to open().</para>
</entry>
 </row><row><entry
 align="char">
<para>int request</para>
</entry><entry
 align="char">
<para>Equals <link linkend="FE_GET_EVENT">FE_GET_EVENT</link> for this
command.</para>
</entry>
 </row><row><entry
 align="char">
<para>struct
 dvb_frontend_event
 *ev</para>
</entry><entry
 align="char">
<para>Points to the location where the event,</para>
</entry>
 </row><row><entry
 align="char">
</entry><entry
 align="char">
<para>if any, is to be stored.</para>
</entry>
 </row></tbody></tgroup></informaltable>

<para>ERRORS
</para>
<informaltable><tgroup cols="2"><tbody><row><entry
 align="char">
<para>EBADF</para>
</entry><entry
 align="char">
<para>fd is not a valid open file descriptor.</para>
</entry>
 </row><row><entry
 align="char">
<para>EFAULT</para>
</entry><entry
 align="char">
<para>ev points to invalid address.</para>
</entry>
 </row><row><entry
 align="char">
<para>EWOULDBLOCK</para>
</entry><entry
 align="char">
```

```
<para>There is no event pending, and the device is in
 non-blocking mode.</para>
</entry>
 </row><row><entry
 align="char">
<para>EOVERFLOW</para>
</entry><entry
 align="char">
</entry>
 </row><row><entry
 align="char">
</entry><entry
 align="char">
<para>Overflow in event queue - one or more events were lost.</para>
</entry>
</row></tbody></tgroup></informaltable>
</section>

<section id="FE_GET_INFO">
<title>FE_GET_INFO</title>
<para>DESCRIPTION
</para>
<informaltable><tgroup cols="1"><tbody><row><entry
 align="char">
<para>This ioctl call returns information about the front-end. This call only
requires
 read-only access to the device.</para>
</entry>
 </row></tbody></tgroup></informaltable>
<para>SYNOPSIS
</para>

<informaltable><tgroup cols="1"><tbody><row><entry
 align="char">
<para> int ioctl(int fd, int request = <link
linkend="FE_GET_INFO">FE_GET_INFO</link>, struct
 dvb_frontend_info &#x22C6;info);</para>
</entry>
 </row></tbody></tgroup></informaltable>
<para>PARAMETERS
</para>

<informaltable><tgroup cols="2"><tbody><row><entry
 align="char">
<para>int fd</para>
</entry><entry
 align="char">
<para>File descriptor returned by a previous call to open().</para>
</entry>
 </row><row><entry
 align="char">
<para>int request</para>
</entry><entry
 align="char">
<para>Equals <link linkend="FE_GET_INFO">FE_GET_INFO</link> for this
command.</para>
```

```
</entry>
 </row><row><entry
 align="char">
<para>struct
 dvb_frontend_info
 *info</para>
</entry><entry
 align="char">
<para>Points to the location where the front-end information is
 to be stored.</para>
</entry>
 </row></tbody></tgroup></informaltable>
<para>ERRORS
</para>
<informaltable><tgroup cols="2"><tbody><row><entry
 align="char">
<para>EBADF</para>
</entry><entry
 align="char">
<para>fd is not a valid open file descriptor.</para>
</entry>
 </row><row><entry
 align="char">
<para>EFAULT</para>
</entry><entry
 align="char">
<para>info points to invalid address.</para>
</entry>
</row></tbody></tgroup></informaltable>
</section>

<section id="FE_DISEQC_RESET_OVERLOAD">
<title>FE_DISEQC_RESET_OVERLOAD</title>
<para>DESCRIPTION
</para>
<informaltable><tgroup cols="1"><tbody><row><entry
 align="char">
<para>If the bus has been automatically powered off due to power overload, this
ioctl
 call restores the power to the bus. The call requires read/write access to the
 device. This call has no effect if the device is manually powered off. Not all
 DVB adapters support this ioctl.</para>
</entry>
 </row></tbody></tgroup></informaltable>

<para>SYNOPSIS
</para>
<informaltable><tgroup cols="1"><tbody><row><entry
 align="char">
<para>int ioctl(int fd, int request =
 <link
linkend="FE_DISEQC_RESET_OVERLOAD">FE_DISEQC_RESET_OVERLOAD</link>);</para>
</entry>
 </row></tbody></tgroup></informaltable>
<para>PARAMETERS
</para>
```

```
<informaltable><tgroup cols="2"><tbody><row><entry
 align="char">
<para>int fd</para>
</entry><entry
 align="char">
<para>File descriptor returned by a previous call to open().</para>
</entry>
 </row><row><entry
 align="char">
<para>int request</para>
</entry><entry
 align="char">
<para>Equals <link
linkend="FE_DISEQC_RESET_OVERLOAD">FE_DISEQC_RESET_OVERLOAD</link> for this
 command.</para>
</entry>
 </row></tbody></tgroup></informaltable>

<para>ERRORS
</para>
<informaltable><tgroup cols="2"><tbody><row><entry
 align="char">
<para>EBADF</para>
</entry><entry
 align="char">
<para>fd is not a valid file descriptor.</para>
</entry>
 </row><row><entry
 align="char">
<para>EPERM</para>
</entry><entry
 align="char">
<para>Permission denied (needs read/write access).</para>
</entry>
 </row><row><entry
 align="char">
<para>EINTERNAL</para>
</entry><entry
 align="char">
<para>Internal error in the device driver.</para>
</entry>
</row></tbody></tgroup></informaltable>
</section>

<section id="FE_DISEQC_SEND_MASTER_CMD">
<title>FE_DISEQC_SEND_MASTER_CMD</title>
<para>DESCRIPTION
</para>
<informaltable><tgroup cols="1"><tbody><row><entry
 align="char">
<para>This ioctl call is used to send a a DiSEqC command.</para>
</entry>
 </row></tbody></tgroup></informaltable>
<para>SYNOPSIS
</para>
<informaltable><tgroup cols="1"><tbody><row><entry
```

```
 align="char">
<para>int ioctl(int fd, int request =
 <link linkend="FE_DISEQC_SEND_MASTER_CMD">FE_DISEQC_SEND_MASTER_CMD</link>,
struct
 dvb_diseqc_master_cmd &#x22C6;cmd);</para>
</entry>
 </row></tbody></tgroup></informaltable>

<para>PARAMETERS
</para>
<informaltable><tgroup cols="2"><tbody><row><entry
 align="char">
<para>int fd</para>
</entry><entry
 align="char">
<para>File descriptor returned by a previous call to open().</para>
</entry>
 </row><row><entry
 align="char">
<para>int request</para>
</entry><entry
 align="char">
<para>Equals <link
linkend="FE_DISEQC_SEND_MASTER_CMD">FE_DISEQC_SEND_MASTER_CMD</link> for this
 command.</para>
</entry>
 </row><row><entry
 align="char">
<para>struct
 dvb_diseqc_master_cmd
 *cmd</para>
</entry><entry
 align="char">
<para>Pointer to the command to be transmitted.</para>
</entry>
 </row></tbody></tgroup></informaltable>

<para>ERRORS
</para>
<informaltable><tgroup cols="2"><tbody><row><entry
 align="char">
<para>EBADF</para>
</entry><entry
 align="char">
<para>fd is not a valid file descriptor.</para>
</entry>
 </row><row><entry
 align="char">
<para>EFAULT</para>
</entry><entry
 align="char">
<para>Seq points to an invalid address.</para>
</entry>
 </row><row><entry
 align="char">
<para>EINVAL</para>
```

```
</entry><entry
 align="char">
<para>The data structure referred to by seq is invalid in some
 way.</para>
</entry>
 </row><row><entry
 align="char">
<para>EPERM</para>
</entry><entry
 align="char">
<para>Permission denied (needs read/write access).</para>
</entry>
 </row><row><entry
 align="char">
<para>EINTERNAL</para>
</entry><entry
 align="char">
<para>Internal error in the device driver.</para>
</entry>
</row></tbody></tgroup></informaltable>
</section>

<section id="FE_DISEQC_RECV_SLAVE_REPLY">
<title>FE_DISEQC_RECV_SLAVE_REPLY</title>
<para>DESCRIPTION
</para>
<informaltable><tgroup cols="1"><tbody><row><entry
 align="char">
<para>This ioctl call is used to receive reply to a DiSEqC 2.0 command.</para>
</entry>
 </row></tbody></tgroup></informaltable>

<para>SYNOPSIS
</para>
<informaltable><tgroup cols="1"><tbody><row><entry
 align="char">
<para>int ioctl(int fd, int request =
 <link linkend="FE_DISEQC_RECV_SLAVE_REPLY">FE_DISEQC_RECV_SLAVE_REPLY</link>,
struct
 dvb_diseqc_slave_reply &#x22C6;reply);</para>
</entry>
 </row></tbody></tgroup></informaltable>

<para>PARAMETERS
</para>
<informaltable><tgroup cols="2"><tbody><row><entry
 align="char">
<para>int fd</para>
</entry><entry
 align="char">
<para>File descriptor returned by a previous call to open().</para>
</entry>
 </row><row><entry
 align="char">
<para>int request</para>
</entry><entry
```

```
 align="char">
<para>Equals <link
linkend="FE_DISEQC_RECV_SLAVE_REPLY">FE_DISEQC_RECV_SLAVE_REPLY</link> for this
 command.</para>
</entry>
 </row><row><entry
 align="char">
<para>struct
 dvb_diseqc_slave_reply
 *reply</para>
</entry><entry
 align="char">
<para>Pointer to the command to be received.</para>
</entry>
 </row></tbody></tgroup></informaltable>
<para>ERRORS
</para>
<informaltable><tgroup cols="2"><tbody><row><entry
 align="char">
<para>EBADF</para>
</entry><entry
 align="char">
<para>fd is not a valid file descriptor.</para>
</entry>
 </row><row><entry
 align="char">
<para>EFAULT</para>
</entry><entry
 align="char">
<para>Seq points to an invalid address.</para>
</entry>
 </row><row><entry
 align="char">
<para>EINVAL</para>
</entry><entry
 align="char">
<para>The data structure referred to by seq is invalid in some
 way.</para>
</entry>
 </row><row><entry
 align="char">
<para>EPERM</para>
</entry><entry
 align="char">
<para>Permission denied (needs read/write access).</para>
</entry>
 </row><row><entry
 align="char">
<para>EINTERNAL</para>
</entry><entry
 align="char">
<para>Internal error in the device driver.</para>
</entry>
 </row></tbody></tgroup></informaltable>
</section>
```

```
<section id="FE_DISEQC_SEND_BURST">
<title>FE_DISEQC_SEND_BURST</title>
<para>DESCRIPTION
</para>
<informaltable><tgroup cols="1"><tbody><row><entry
 align="char">
<para>This ioctl call is used to send a 22KHz tone burst.</para>
</entry>
 </row></tbody></tgroup></informaltable>

<para>SYNOPSIS
</para>
<informaltable><tgroup cols="1"><tbody><row><entry
 align="char">
<para>int ioctl(int fd, int request =
 <link linkend="FE_DISEQC_SEND_BURST">FE_DISEQC_SEND_BURST</link>,
fe_sec_mini_cmd_t burst);</para>
</entry>
 </row></tbody></tgroup></informaltable>

<para>PARAMETERS
</para>
<informaltable><tgroup cols="2"><tbody><row><entry
 align="char">
<para>int fd</para>
</entry><entry
 align="char">
<para>File descriptor returned by a previous call to open().</para>
</entry>
 </row><row><entry
 align="char">
<para>int request</para>
</entry><entry
 align="char">
<para>Equals <link linkend="FE_DISEQC_SEND_BURST">FE_DISEQC_SEND_BURST</link>
for this command.</para>
</entry>
 </row><row><entry
 align="char">
<para>fe_sec_mini_cmd_t
 burst</para>
</entry><entry
 align="char">
<para>burst A or B.</para>
</entry>
 </row></tbody></tgroup></informaltable>

<para>ERRORS
</para>
<informaltable><tgroup cols="2"><tbody><row><entry
 align="char">
<para>EBADF</para>
</entry><entry
 align="char">
<para>fd is not a valid file descriptor.</para>
</entry>
```

```
 </row><row><entry
 align="char">
<para>EFAULT</para>
</entry><entry
 align="char">
<para>Seq points to an invalid address.</para>
</entry>
 </row><row><entry
 align="char">
<para>EINVAL</para>
</entry><entry
 align="char">
<para>The data structure referred to by seq is invalid in some
 way.</para>
</entry>
 </row><row><entry
 align="char">
<para>EPERM</para>
</entry><entry
 align="char">
<para>Permission denied (needs read/write access).</para>
</entry>
 </row><row><entry
 align="char">
<para>EINTERNAL</para>
</entry><entry
 align="char">
<para>Internal error in the device driver.</para>
</entry>
</row></tbody></tgroup></informaltable>
</section>

<section id="FE_SET_TONE">
<title>FE_SET_TONE</title>
<para>DESCRIPTION
</para>
<informaltable><tgroup cols="1"><tbody><row><entry
 align="char">
<para>This call is used to set the generation of the continuous 22kHz tone. This
call
 requires read/write permissions.</para>
</entry>
 </row></tbody></tgroup></informaltable>
<para>SYNOPSIS
</para>
<informaltable><tgroup cols="1"><tbody><row><entry
 align="char">
<para>int ioctl(int fd, int request = <link
linkend="FE_SET_TONE">FE_SET_TONE</link>,
 fe_sec_tone_mode_t tone);</para>
</entry>
 </row></tbody></tgroup></informaltable>
<para>PARAMETERS
</para>
<informaltable><tgroup cols="2"><tbody><row><entry
 align="char">
```

```
<para>int fd</para>
</entry><entry
 align="char">
<para>File descriptor returned by a previous call to open().</para>
</entry>
 </row><row><entry
 align="char">
<para>int request</para>
</entry><entry
 align="char">
<para>Equals <link linkend="FE_SET_TONE">FE_SET_TONE</link> for this
command.</para>
</entry>
 </row><row><entry
 align="char">
<para>fe_sec_tone_mode_t
 tone</para>
</entry><entry
 align="char">
<para>The requested tone generation mode (on/off).</para>
</entry>
 </row></tbody></tgroup></informaltable>
<para>ERRORS
</para>
<informaltable><tgroup cols="2"><tbody><row><entry
 align="char">
<para>ENODEV</para>
</entry><entry
 align="char">
<para>Device driver not loaded/available.</para>
</entry>
 </row><row><entry
 align="char">
<para>EBUSY</para>
</entry><entry
 align="char">
<para>Device or resource busy.</para>
</entry>
 </row><row><entry
 align="char">
<para>EINVAL</para>
</entry><entry
 align="char">
<para>Invalid argument.</para>
</entry>
 </row><row><entry
 align="char">
<para>EPERM</para>
</entry><entry
 align="char">
<para>File not opened with read permissions.</para>
</entry>
 </row><row><entry
 align="char">
<para>EINTERNAL</para>
</entry><entry
```

```
 align="char">
<para>Internal error in the device driver.</para>
</entry>
</row></tbody></tgroup></informaltable>
</section>

<section id="FE_SET_VOLTAGE">
<title>FE_SET_VOLTAGE</title>
<para>DESCRIPTION
</para>
<informaltable><tgroup cols="1"><tbody><row><entry
 align="char">
<para>This call is used to set the bus voltage. This call requires read/write
 permissions.</para>
</entry>
 </row></tbody></tgroup></informaltable>
<para>SYNOPSIS
</para>
<informaltable><tgroup cols="1"><tbody><row><entry
 align="char">
<para>int ioctl(int fd, int request = <link
linkend="FE_SET_VOLTAGE">FE_SET_VOLTAGE</link>,
 fe_sec_voltage_t voltage);</para>
</entry>
 </row></tbody></tgroup></informaltable>

<para>PARAMETERS
</para>
<informaltable><tgroup cols="2"><tbody><row><entry
 align="char">
<para>int fd</para>
</entry><entry
 align="char">
<para>File descriptor returned by a previous call to open().</para>
</entry>
 </row><row><entry
 align="char">
<para>int request</para>
</entry><entry
 align="char">
<para>Equals <link linkend="FE_SET_VOLTAGE">FE_SET_VOLTAGE</link> for this
command.</para>
</entry>
 </row><row><entry
 align="char">
<para>fe_sec_voltage_t
 voltage</para>
</entry><entry
 align="char">
<para>The requested bus voltage.</para>
</entry>
 </row></tbody></tgroup></informaltable>

<para>ERRORS
</para>
<informaltable><tgroup cols="2"><tbody><row><entry
```

```
 align="char">
<para>ENODEV</para>
</entry><entry
 align="char">
<para>Device driver not loaded/available.</para>
</entry>
 </row><row><entry
 align="char">
<para>EBUSY</para>
</entry><entry
 align="char">
<para>Device or resource busy.</para>
</entry>
 </row><row><entry
 align="char">
<para>EINVAL</para>
</entry><entry
 align="char">
<para>Invalid argument.</para>
</entry>
 </row><row><entry
 align="char">
<para>EPERM</para>
</entry><entry
 align="char">
<para>File not opened with read permissions.</para>
</entry>
 </row><row><entry
 align="char">
<para>EINTERNAL</para>
</entry><entry
 align="char">
<para>Internal error in the device driver.</para>
</entry>
 </row></tbody></tgroup></informaltable>
</section>

<section id="FE_ENABLE_HIGH_LNB_VOLTAGE">
<title>FE_ENABLE_HIGH_LNB_VOLTAGE</title>
<para>DESCRIPTION
</para>
<informaltable><tgroup cols="1"><tbody><row><entry
 align="char">
<para>If high != 0 enables slightly higher voltages instead of 13/18V (to
compensate
 for long cables). This call requires read/write permissions. Not all DVB
 adapters support this ioctl.</para>
</entry>
 </row></tbody></tgroup></informaltable>

<para>SYNOPSIS
</para>
<informaltable><tgroup cols="1"><tbody><row><entry
 align="char">
<para>int ioctl(int fd, int request =
 <link linkend="FE_ENABLE_HIGH_LNB_VOLTAGE">FE_ENABLE_HIGH_LNB_VOLTAGE</link>,
int high);</para>
</entry>
 </row></tbody></tgroup></informaltable>

<para>PARAMETERS
</para>
<informaltable><tgroup cols="2"><tbody><row><entry
 align="char">
<para>int fd</para>
</entry><entry
 align="char">
<para>File descriptor returned by a previous call to open().</para>
</entry>
 </row><row><entry
 align="char">
<para>int request</para>
</entry><entry
 align="char">
<para>Equals <link linkend="FE_SET_VOLTAGE">FE_SET_VOLTAGE</link> for this
command.</para>
</entry>
 </row><row><entry
 align="char">
<para>int high</para>
</entry><entry
 align="char">
<para>The requested bus voltage.</para>
</entry>
 </row></tbody></tgroup></informaltable>

<para>ERRORS
</para>
<informaltable><tgroup cols="2"><tbody><row><entry
 align="char">
<para>ENODEV</para>
</entry><entry
 align="char">
<para>Device driver not loaded/available.</para>
</entry>
 </row><row><entry
 align="char">
<para>EBUSY</para>
</entry><entry
 align="char">
<para>Device or resource busy.</para>
</entry>
 </row><row><entry
 align="char">
<para>EINVAL</para>
</entry><entry
 align="char">
<para>Invalid argument.</para>
</entry>
 </row><row><entry
 align="char">
<para>EPERM</para>
```

```
</entry><entry
 align="char">
<para>File not opened with read permissions.</para>
</entry>
 </row><row><entry
 align="char">
<para>EINTERNAL</para>
</entry><entry
 align="char">
<para>Internal error in the device driver.</para>
</entry>
 </row></tbody></tgroup></informaltable>
</section>

<section id="FE_SET_FRONTEND_TUNE_MODE">
<title>FE_SET_FRONTEND_TUNE_MODE</title>
<para>DESCRIPTION</para>
<informaltable><tgroup cols="1"><tbody><row>
<entry align="char">
<para>Allow setting tuner mode flags to the frontend.</para>
</entry>
</row></tbody></tgroup></informaltable>

<para>SYNOPSIS</para>
<informaltable><tgroup cols="1"><tbody><row>
<entry align="char">
<para>int ioctl(int fd, int request =
<link linkend="FE_SET_FRONTEND_TUNE_MODE">FE_SET_FRONTEND_TUNE_MODE</link>,
unsigned int flags);</para>
</entry>
</row></tbody></tgroup></informaltable>

<para>PARAMETERS</para>
<informaltable><tgroup cols="2"><tbody><row>
<entry align="char">
        <para>unsigned int flags</para>
</entry>
<entry align="char">
<para>
FE_TUNE_MODE_ONESHOT When set, this flag will disable any zigzagging or other
"normal" tuning behaviour. Additionally, there will be no automatic monitoring
of the lock status, and hence no frontend events will be generated. If a
frontend device is closed, this flag will be automatically turned off when the
device is reopened read-write.
</para>
</entry>
 </row></tbody></tgroup></informaltable>

<para>ERRORS</para>
<informaltable><tgroup cols="2"><tbody><row>
<entry align="char"><para>EINVAL</para></entry>
<entry align="char"><para>Invalid argument.</para></entry>
 </row></tbody></tgroup></informaltable>
</section>

<section id="FE_DISHNETWORK_SEND_LEGACY_CMD">
```

```
        <title>FE_DISHNETWORK_SEND_LEGACY_CMD</title>
<para>DESCRIPTION</para>
<informaltable><tgroup cols="1"><tbody><row>
<entry align="char">
<para>WARNING: This is a very obscure legacy command, used only at stv0299
driver. Should not be used on newer drivers.</para>
<para>It provides a non-standard method for selecting Diseqc voltage on the
frontend, for Dish Network legacy switches.</para>
<para>As support for this ioctl were added in 2004, this means that such dishes
were already legacy in 2004.</para>
</entry>
</row></tbody></tgroup></informaltable>

<para>SYNOPSIS</para>
<informaltable><tgroup cols="1"><tbody><row>
<entry align="char">
<para>int ioctl(int fd, int request =
        <link
linkend="FE_DISHNETWORK_SEND_LEGACY_CMD">FE_DISHNETWORK_SEND_LEGACY_CMD</link>,
unsigned long cmd);</para>
</entry>
</row></tbody></tgroup></informaltable>

<para>PARAMETERS</para>
<informaltable><tgroup cols="2"><tbody><row>
<entry align="char">
        <para>unsigned long cmd</para>
</entry>
<entry align="char">
<para>
sends the specified raw cmd to the dish via DISEqC.
</para>
</entry>
 </row></tbody></tgroup></informaltable>

<para>ERRORS</para>
<informaltable><tgroup cols="1"><tbody><row>
<entry align="char">
        <para>There are no errors in use for this call</para>
</entry>
</row></tbody></tgroup></informaltable>
</section>

</section>

&sub-dvbproperty;
```