```
<title>Examples</title>
<para>In this section we would like to present some examples for using the DVB
API.
</para>
<para>Maintainer note: This section is out of date. Please refer to the sample
programs packaged
with the driver distribution from <ulink url="http://linuxtv.org/hg/dvb-apps"
/>.
</para>

<section id="tuning">
<title>Tuning</title>
<para>We will start with a generic tuning subroutine that uses the frontend and
SEC, as well as
the demux devices. The example is given for QPSK tuners, but can easily be
adjusted for
QAM.
</para>
<programlisting>
 #include &#x003C;sys/ioctl.h&#x003E;
 #include &#x003C;stdio.h&#x003E;
 #include &#x003C;stdint.h&#x003E;
 #include &#x003C;sys/types.h&#x003E;
 #include &#x003C;sys/stat.h&#x003E;
 #include &#x003C;fcntl.h&#x003E;
 #include &#x003C;time.h&#x003E;
 #include &#x003C;unistd.h&#x003E;

 #include &#x003C;linux/dvb/dmx.h&#x003E;
 #include &#x003C;linux/dvb/frontend.h&#x003E;
 #include &#x003C;linux/dvb/sec.h&#x003E;
 #include &#x003C;sys/poll.h&#x003E;

 #define DMX "/dev/dvb/adapter0/demux1"
 #define FRONT "/dev/dvb/adapter0/frontend1"
 #define SEC "/dev/dvb/adapter0/sec1"

 /&#x22C6; routine for checking if we have a signal and other status
information&#x22C6;/
 int FEReadStatus(int fd, fe_status_t &#x22C6;stat)
 {
         int ans;

         if ( (ans = ioctl(fd,FE_READ_STATUS,stat) &#x003C; 0)){
                 perror("FE READ STATUS: ");
                 return -1;
         }

         if (&#x22C6;stat &amp; FE_HAS_POWER)
                 printf("FE HAS POWER\n");

         if (&#x22C6;stat &amp; FE_HAS_SIGNAL)
                 printf("FE HAS SIGNAL\n");

         if (&#x22C6;stat &amp; FE_SPECTRUM_INV)
                 printf("SPEKTRUM INV\n");
```

```
        return 0;
 }


 /&#x22C6; tune qpsk &#x22C6;/
 /&#x22C6; freq:             frequency of transponder
&#x22C6;/
 /&#x22C6; vpid, apid, tpid: PIDs of video, audio and teletext TS packets
&#x22C6;/
 /&#x22C6; diseqc:           DiSEqC address of the used LNB
&#x22C6;/
 /&#x22C6; pol:              Polarisation
&#x22C6;/
 /&#x22C6; srate:            Symbol Rate
&#x22C6;/
 /&#x22C6; fec.              FEC
&#x22C6;/
 /&#x22C6; lnb_lof1:         local frequency of lower LNB band
&#x22C6;/
 /&#x22C6; lnb_lof2:         local frequency of upper LNB band
&#x22C6;/
 /&#x22C6; lnb_slof:         switch frequency of LNB
&#x22C6;/

 int set_qpsk_channel(int freq, int vpid, int apid, int tpid,
                 int diseqc, int pol, int srate, int fec, int lnb_lof1,
                 int lnb_lof2, int lnb_slof)
 {
        struct secCommand scmd;
        struct secCmdSequence scmds;
        struct dmx_pes_filter_params pesFilterParams;
        FrontendParameters frp;
        struct pollfd pfd[1];
        FrontendEvent event;
        int demux1, demux2, demux3, front;

        frequency = (uint32_t) freq;
        symbolrate = (uint32_t) srate;

        if((front = open(FRONT,O_RDWR)) &#x003C; 0){
                perror("FRONTEND DEVICE: ");
                return -1;
        }

        if((sec = open(SEC,O_RDWR)) &#x003C; 0){
                perror("SEC DEVICE: ");
                return -1;
        }

        if (demux1 &#x003C; 0){
                if ((demux1=open(DMX, O_RDWR|O_NONBLOCK))
                    &#x003C; 0){
                        perror("DEMUX DEVICE: ");
                        return -1;
                }
```

```
}

if (demux2 &#x003C; 0){
        if ((demux2=open(DMX, O_RDWR|O_NONBLOCK))
            &#x003C; 0){
                perror("DEMUX DEVICE: ");
                return -1;
        }
}

if (demux3 &#x003C; 0){
        if ((demux3=open(DMX, O_RDWR|O_NONBLOCK))
            &#x003C; 0){
                perror("DEMUX DEVICE: ");
                return -1;
        }
}

if (freq &#x003C; lnb_slof) {
        frp.Frequency = (freq - lnb_lof1);
        scmds.continuousTone = SEC_TONE_OFF;
} else {
        frp.Frequency = (freq - lnb_lof2);
        scmds.continuousTone = SEC_TONE_ON;
}
frp.Inversion = INVERSION_AUTO;
if (pol) scmds.voltage = SEC_VOLTAGE_18;
else scmds.voltage = SEC_VOLTAGE_13;

scmd.type=0;
scmd.u.diseqc.addr=0x10;
scmd.u.diseqc.cmd=0x38;
scmd.u.diseqc.numParams=1;
scmd.u.diseqc.params[0] = 0xF0 | ((diseqc &#x22C6; 4) &amp; 0x0F) |
        (scmds.continuousTone == SEC_TONE_ON ? 1 : 0) |
        (scmds.voltage==SEC_VOLTAGE_18 ? 2 : 0);

scmds.miniCommand=SEC_MINI_NONE;
scmds.numCommands=1;
scmds.commands=&amp;scmd;
if (ioctl(sec, SEC_SEND_SEQUENCE, &amp;scmds) &#x003C; 0){
        perror("SEC SEND: ");
        return -1;
}

if (ioctl(sec, SEC_SEND_SEQUENCE, &amp;scmds) &#x003C; 0){
        perror("SEC SEND: ");
        return -1;
}

frp.u.qpsk.SymbolRate = srate;
frp.u.qpsk.FEC_inner = fec;

if (ioctl(front, FE_SET_FRONTEND, &amp;frp) &#x003C; 0){
        perror("QPSK TUNE: ");
        return -1;
```

```
        }

        pfd[0].fd = front;
        pfd[0].events = POLLIN;

        if (poll(pfd,1,3000)){
                if (pfd[0].revents &amp; POLLIN){
                        printf("Getting QPSK event\n");
                        if ( ioctl(front, FE_GET_EVENT, &amp;event)

                             == -EOVERFLOW){
                                perror("qpsk get event");
                                return -1;
                        }
                        printf("Received ");
                        switch(event.type){
                        case FE_UNEXPECTED_EV:
                                printf("unexpected event\n");
                                return -1;
                        case FE_FAILURE_EV:
                                printf("failure event\n");
                                return -1;

                        case FE_COMPLETION_EV:
                                printf("completion event\n");
                        }
                }
        }


        pesFilterParams.pid     = vpid;
        pesFilterParams.input   = DMX_IN_FRONTEND;
        pesFilterParams.output  = DMX_OUT_DECODER;
        pesFilterParams.pes_type = DMX_PES_VIDEO;
        pesFilterParams.flags   = DMX_IMMEDIATE_START;
        if (ioctl(demux1, DMX_SET_PES_FILTER, &amp;pesFilterParams) &#x003C;
0){
                perror("set_vpid");
                return -1;
        }

        pesFilterParams.pid     = apid;
        pesFilterParams.input   = DMX_IN_FRONTEND;
        pesFilterParams.output  = DMX_OUT_DECODER;
        pesFilterParams.pes_type = DMX_PES_AUDIO;
        pesFilterParams.flags   = DMX_IMMEDIATE_START;
        if (ioctl(demux2, DMX_SET_PES_FILTER, &amp;pesFilterParams) &#x003C;
0){
                perror("set_apid");
                return -1;
        }

        pesFilterParams.pid     = tpid;
        pesFilterParams.input   = DMX_IN_FRONTEND;
        pesFilterParams.output  = DMX_OUT_DECODER;
        pesFilterParams.pes_type = DMX_PES_TELETEXT;
```

```
        pesFilterParams.flags   = DMX_IMMEDIATE_START;
        if (ioctl(demux3, DMX_SET_PES_FILTER, &amp;pesFilterParams) &#x003C;
0){
                perror("set_tpid");
                return -1;
        }

        return has_signal(fds);
 }

</programlisting>
<para>The program assumes that you are using a universal LNB and a standard
DiSEqC
switch with up to 4 addresses. Of course, you could build in some more checking
if
tuning was successful and maybe try to repeat the tuning process. Depending on
the
external hardware, i.e. LNB and DiSEqC switch, and weather conditions this may
be
necessary.
</para>
</section>

<section id="the_dvr_device">
<title>The DVR device</title>
<para>The following program code shows how to use the DVR device for recording.
</para>
<programlisting>
 #include &#x003C;sys/ioctl.h&#x003E;
 #include &#x003C;stdio.h&#x003E;
 #include &#x003C;stdint.h&#x003E;
 #include &#x003C;sys/types.h&#x003E;
 #include &#x003C;sys/stat.h&#x003E;
 #include &#x003C;fcntl.h&#x003E;
 #include &#x003C;time.h&#x003E;
 #include &#x003C;unistd.h&#x003E;

 #include &#x003C;linux/dvb/dmx.h&#x003E;
 #include &#x003C;linux/dvb/video.h&#x003E;
 #include &#x003C;sys/poll.h&#x003E;
 #define DVR "/dev/dvb/adapter0/dvr1"
 #define AUDIO "/dev/dvb/adapter0/audio1"
 #define VIDEO "/dev/dvb/adapter0/video1"

 #define BUFFY (188&#x22C6;20)
 #define MAX_LENGTH (1024&#x22C6;1024&#x22C6;5) /&#x22C6; record 5MB &#x22C6;/


 /&#x22C6; switch the demuxes to recording, assuming the transponder is tuned
&#x22C6;/

 /&#x22C6; demux1, demux2: file descriptor of video and audio filters &#x22C6;/
 /&#x22C6; vpid, apid:     PIDs of video and audio channels           &#x22C6;/

 int switch_to_record(int demux1, int demux2, uint16_t vpid, uint16_t apid)
 {
```

```
        struct dmx_pes_filter_params pesFilterParams;

        if (demux1 &#x003C; 0){
                if ((demux1=open(DMX, O_RDWR|O_NONBLOCK))
                    &#x003C; 0){
                        perror("DEMUX DEVICE: ");
                        return -1;
                }
        }

        if (demux2 &#x003C; 0){
                if ((demux2=open(DMX, O_RDWR|O_NONBLOCK))
                    &#x003C; 0){
                        perror("DEMUX DEVICE: ");
                        return -1;
                }
        }

        pesFilterParams.pid = vpid;
        pesFilterParams.input = DMX_IN_FRONTEND;
        pesFilterParams.output = DMX_OUT_TS_TAP;
        pesFilterParams.pes_type = DMX_PES_VIDEO;
        pesFilterParams.flags = DMX_IMMEDIATE_START;
        if (ioctl(demux1, DMX_SET_PES_FILTER, &amp;pesFilterParams) &#x003C;
0){
                perror("DEMUX DEVICE");
                return -1;
        }
        pesFilterParams.pid = apid;
        pesFilterParams.input = DMX_IN_FRONTEND;
        pesFilterParams.output = DMX_OUT_TS_TAP;
        pesFilterParams.pes_type = DMX_PES_AUDIO;
        pesFilterParams.flags = DMX_IMMEDIATE_START;
        if (ioctl(demux2, DMX_SET_PES_FILTER, &amp;pesFilterParams) &#x003C;
0){
                perror("DEMUX DEVICE");
                return -1;
        }
        return 0;
 }

 /&#x22C6; start recording MAX_LENGTH , assuming the transponder is tuned
&#x22C6;/

 /&#x22C6; demux1, demux2: file descriptor of video and audio filters &#x22C6;/
 /&#x22C6; vpid, apid:     PIDs of video and audio channels           &#x22C6;/
 int record_dvr(int demux1, int demux2, uint16_t vpid, uint16_t apid)
 {
        int i;
        int len;
        int written;
        uint8_t buf[BUFFY];
        uint64_t length;
        struct pollfd pfd[1];
        int dvr, dvr_out;
```

```
        /&#x22C6; open dvr device &#x22C6;/
        if ((dvr = open(DVR, O_RDONLY|O_NONBLOCK)) &#x003C; 0){
                        perror("DVR DEVICE");
                        return -1;
        }

        /&#x22C6; switch video and audio demuxes to dvr &#x22C6;/
        printf ("Switching dvr on\n");
        i = switch_to_record(demux1, demux2, vpid, apid);
        printf("finished: ");

        printf("Recording %2.0f MB of test file in TS format\n",
               MAX_LENGTH/(1024.0&#x22C6;1024.0));
        length = 0;

        /&#x22C6; open output file &#x22C6;/
        if ((dvr_out = open(DVR_FILE,O_WRONLY|O_CREAT
                                |O_TRUNC, S_IRUSR|S_IWUSR
                                |S_IRGRP|S_IWGRP|S_IROTH|
                                S_IWOTH)) &#x003C; 0){
                perror("Can't open file for dvr test");
                return -1;
        }

        pfd[0].fd = dvr;
        pfd[0].events = POLLIN;

        /&#x22C6; poll for dvr data and write to file &#x22C6;/
        while (length &#x003C; MAX_LENGTH ) {
                if (poll(pfd,1,1)){
                        if (pfd[0].revents &amp; POLLIN){
                                len = read(dvr, buf, BUFFY);
                                if (len &#x003C; 0){
                                        perror("recording");
                                        return -1;
                                }
                                if (len &#x003E; 0){
                                        written = 0;
                                        while (written &#x003C; len)
                                                written +=
                                                        write (dvr_out,
                                                               buf, len);
                                        length += len;
                                        printf("written %2.0f MB\r",
                                               length/1024./1024.);
                                }
                        }
                }
        }
        return 0;
 }

</programlisting>

</section>
```