
Linux Media Documentation

The kernel development community

Dec 03, 2021

CONTENTS

1	Media subsystem admin and user guide	3
1.1	The media subsystem	3
1.1.1	Introduction	3
1.1.2	Building support for a media device	4
1.1.2.1	Configuring the Linux Kernel	4
1.1.2.2	Building and installing a new Kernel	8
1.1.2.3	Building just the new media drivers and core	9
1.1.3	Infrared remote control support in video4linux drivers	10
1.1.3.1	Basics	10
1.1.3.2	How it works	10
1.1.4	Digital TV	11
1.1.4.1	Using the Digital TV Framework	11
1.1.4.2	Digital TV Conditional Access Interface	22
1.1.4.3	FAQ	23
1.1.4.4	References	26
1.1.5	Cards List	26
1.1.5.1	USB drivers	27
1.1.5.2	PCI drivers	62
1.1.5.3	Platform drivers	83
1.1.5.4	Radio drivers	85
1.1.5.5	I ² C drivers	86
1.1.5.6	Firewire driver	98
1.1.5.7	Test drivers	98
1.1.6	Video4Linux (V4L) driver-specific documentation	99
1.1.6.1	The bttv driver	99
1.1.6.2	The cafe_ccic driver	132
1.1.6.3	The cpia2 driver	133
1.1.6.4	The cx88 driver	136
1.1.6.5	The VPBE V4L2 driver design	137
1.1.6.6	The Samsung S5P/Exynos4 FIMC driver	138
1.1.6.7	i.MX Video Capture Driver	140
1.1.6.8	i.MX7 Video Capture Driver	153
1.1.6.9	Intel Image Processing Unit 3 (IPU3) Imaging Unit (ImgU) driver	157
1.1.6.10	The ivtv driver	166
1.1.6.11	Vaio Picturebook Motion Eye Camera Driver	170
1.1.6.12	OMAP 3 Image Signal Processor (ISP) driver	171
1.1.6.13	OMAP4 ISS Driver	173
1.1.6.14	Philips webcams (pwc driver)	174
1.1.6.15	Qualcomm Camera Subsystem driver	178

1.1.6.16	Renesas R-Car Fine Display Processor (FDP1) Driver	182
1.1.6.17	Rockchip Image Signal Processor (rkisp1)	183
1.1.6.18	The saa7134 driver	187
1.1.6.19	The Silicon Labs Si470x FM Radio Receivers driver	188
1.1.6.20	The Silicon Labs Si4713 FM Radio Transmitter Driver	191
1.1.6.21	The SI476x Driver	195
1.1.6.22	The Virtual Media Controller Driver (vimc)	198
1.1.6.23	The Virtual Video Test Driver (vivid)	199
1.1.7	Digital TV driver-specific documentation	222
1.1.7.1	Avermedia DVB-T on BT878 Release Notes	222
1.1.7.2	How to get the bt8xx cards working	224
1.1.7.3	Firmware files for lmedm04 cards	227
1.1.7.4	Opera firmware	228
1.1.7.5	How to set up the Technisat/B2C2 Flexcop devices	229
1.1.7.6	TechnoTrend/Hauppauge DEC USB Driver	230
1.1.7.7	Zoran 364xx based USB webcam module	231
1.1.8	CEC driver-specific documentation	233
1.1.8.1	Pulse-Eight CEC Adapter driver	233

2 Media subsystem kernel internal API 235

2.1	Media Subsystem Profile	235
2.1.1	Overview	235
2.1.1.1	Media maintainers	236
2.1.2	Submit Checklist Addendum	236
2.1.2.1	Style Cleanup Patches	237
2.1.2.2	Coding Style Addendum	237
2.1.3	Key Cycle Dates	238
2.1.4	Review Cadence	238
2.2	Video4Linux devices	238
2.2.1	Introduction	238
2.2.2	Structure of a V4L driver	239
2.2.3	Structure of the V4L2 framework	239
2.2.4	Video device' s internal representation	239
2.2.4.1	ioctl's and locking	241
2.2.4.2	Video device registration	242
2.2.4.3	video device debugging	243
2.2.4.4	Video device cleanup	244
2.2.4.5	helper functions	244
2.2.4.6	video_device functions and data structures	245
2.2.5	V4L2 device instance	254
2.2.5.1	v4l2_device functions and data structures	256
2.2.6	V4L2 File handlers	264
2.2.6.1	V4L2 fh functions and data structures	266
2.2.7	V4L2 sub-devices	268
2.2.7.1	Subdev registration	270
2.2.7.2	Calling subdev operations	272
2.2.8	V4L2 sub-device userspace API	273
2.2.9	Read-only sub-device userspace API	274
2.2.10	I2C sub-device drivers	274
2.2.11	V4L2 sub-device functions and data structures	276
2.2.12	V4L2 events	294

2.2.12.1	Event subscription	295
2.2.12.2	Unsubscribing an event	296
2.2.12.3	Check if there's a pending event	296
2.2.12.4	How events work	296
2.2.13	V4L2 Controls	300
2.2.13.1	Introduction	300
2.2.13.2	Objects in the framework	301
2.2.13.3	Basic usage for V4L2 and sub-device drivers	301
2.2.13.4	Inheriting Sub-device Controls	304
2.2.13.5	Accessing Control Values	305
2.2.13.6	Menu Controls	306
2.2.13.7	Custom Controls	307
2.2.13.8	Active and Grabbed Controls	307
2.2.13.9	Control Clusters	307
2.2.13.10	Handling autogain/gain-type Controls with Auto Clusters	309
2.2.13.11	VIDIOC_LOG_STATUS Support	310
2.2.13.12	Different Handlers for Different Video Nodes	310
2.2.13.13	Binding Controls	311
2.2.13.14	Preventing Controls inheritance	311
2.2.13.15	V4L2_CTRL_TYPE_CTRL_CLASS Controls	312
2.2.13.16	Adding Notify Callbacks	312
2.2.13.17	V4L2_ctrl functions and data structures	312
2.2.14	Videobuf Framework	337
2.2.14.1	Introduction	337
2.2.14.2	Buffer types	337
2.2.14.3	Data structures, callbacks, and initialization	338
2.2.14.4	File operations	339
2.2.14.5	ioctl() operations	341
2.2.14.6	Buffer allocation	341
2.2.14.7	Filling the buffers	342
2.2.15	V4L2 videobuf2 functions and data structures	343
2.2.16	V4L2 DV Timings functions	368
2.2.17	V4L2 flash functions and data structures	372
2.2.18	V4L2 Media Controller functions and data structures	375
2.2.19	V4L2 Media Bus functions and data structures	378
2.2.20	V4L2 Memory to Memory functions and data structures	379
2.2.21	V4L2 async kAPI	392
2.2.22	V4L2 fwnode kAPI	397
2.2.23	V4L2 rect helper functions	406
2.2.24	Tuner functions and data structures	408
2.2.25	V4L2 common functions and data structures	412
2.2.26	Hauppauge TV EEPROM functions and data structures	428
2.3	Digital TV (DVB) devices	430
2.3.1	Digital TV Common functions	430
2.3.1.1	Math functions	430
2.3.1.2	DVB devices	431
2.3.1.3	Digital TV Ring buffer	438
2.3.1.4	Digital TV VB2 handler	443
2.3.2	Digital TV Frontend kABI	447
2.3.2.1	Digital TV Frontend	447
2.3.2.2	Digital TV Frontend statistics	450

2.3.2.3	Digital TV Frontend functions and types	455
2.3.3	Digital TV Demux kABI	468
2.3.3.1	Digital TV Demux	468
2.3.3.2	Demux Callback API	469
2.3.3.3	Digital TV Demux device registration functions and data structures	469
2.3.3.4	High-level Digital TV demux interface	472
2.3.3.5	Driver-internal low-level hardware specific driver demux interface	477
2.3.4	Digital TV Conditional Access kABI	485
2.3.5	Digital TV Network kABI	487
2.4	Remote Controller devices	488
2.4.1	Remote Controller core	488
2.4.1.1	Remote controller data structures and functions	489
2.5	Media Controller devices	495
2.5.1	Media Controller	495
2.5.1.1	Abstract media device model	495
2.5.1.2	Media device	496
2.5.1.3	Entities	496
2.5.1.4	Interfaces	496
2.5.1.5	Pads	496
2.5.1.6	Links	497
2.5.1.7	Graph traversal	497
2.5.1.8	Use count and power handling	498
2.5.1.9	Links setup	498
2.5.1.10	Pipelines and media streams	498
2.5.1.11	Link validation	499
2.5.1.12	Media Controller Device Allocator API	499
2.5.1.13	API Definitions	500
2.6	CEC Kernel Support	533
2.6.1	The CEC Protocol	533
2.6.2	CEC Adapter Interface	533
2.6.3	Implementing the Low-Level CEC Adapter	534
2.6.4	Implementing the interrupt handler	537
2.6.5	Optional: Implementing Error Injection Support	537
2.6.6	Implementing the High-Level CEC Adapter	538
2.6.7	CEC framework functions	538
2.6.8	CEC Pin framework	539
2.6.9	CEC Notifier framework	541
2.7	Pixel data transmitter and receiver drivers	543
2.7.1	Bus types	543
2.7.1.1	MIPI CSI-2	543
2.7.1.2	Parallel	543
2.7.2	Transmitter drivers	543
2.7.2.1	Media bus pixel code	543
2.7.2.2	Link frequency	543
2.7.2.3	.s_stream() callback	543
2.7.3	CSI-2 transmitter drivers	544
2.7.3.1	Pixel rate	544
2.7.3.2	LP-11 and LP-111 modes	544
2.7.3.3	Stopping the transmitter	545
2.8	Writing camera sensor drivers	545
2.8.1	CSI-2 and parallel (BT.601 and BT.656) busses	545

2.8.2	Handling clocks	545
2.8.2.1	ACPI	545
2.8.2.2	Devicetree	545
2.8.3	Frame size	546
2.8.3.1	Freely configurable camera sensor drivers	546
2.8.3.2	Register list based drivers	546
2.8.4	Frame interval configuration	546
2.8.4.1	Raw camera sensors	546
2.8.4.2	USB cameras etc. devices	547
2.8.5	Power management	547
2.8.5.1	Control framework	547
2.9	Media driver-specific documentation	547
2.9.1	Video4Linux (V4L) drivers	547
2.9.1.1	The bttv driver	547
2.9.1.2	The cpia2 driver	550
2.9.1.3	The cx2341x driver	551
2.9.1.4	The cx88 driver	621
2.9.1.5	The VPBE V4L2 driver design	623
2.9.1.6	The Samsung S5P/EXYNOS4 FIMC driver	624
2.9.1.7	The pvrusb2 driver	624
2.9.1.8	PXA-Camera Host Driver	628
2.9.1.9	The Radiotrack radio driver	631
2.9.1.10	The Rockchip Image Signal Processor Driver (rkisp1)	633
2.9.1.11	The saa7134 driver	634
2.9.1.12	Cropping and Scaling algorithm, used in the sh_mobile_ceu_camera driver	636
2.9.1.13	Tuner drivers	638
2.9.1.14	The Virtual Media Controller Driver (vimc)	640
2.9.1.15	The Zoran driver	642
2.9.1.16	MIPI CCS camera sensor driver	653
2.9.1.17	CCS PLL calculator	654
2.9.2	Digital TV drivers	658
2.9.2.1	Idea behind the dvb-usb-framework	658
2.9.2.2	Frontend drivers	664
2.9.2.3	vidtv: Virtual Digital TV driver	678
2.9.2.4	Contributors	710

3 Linux Media Infrastructure userspace API 713

3.1	Introduction	713
3.2	Part I - Video for Linux API	715
3.2.1	Common API Elements	715
3.2.1.1	Opening and Closing Devices	715
3.2.1.2	Querying Capabilities	718
3.2.1.3	Application Priority	719
3.2.1.4	Video Inputs and Outputs	719
3.2.1.5	Audio Inputs and Outputs	720
3.2.1.6	Tuners and Modulators	722
3.2.1.7	Video Standards	723
3.2.1.8	Digital Video (DV) Timings	726
3.2.1.9	User Controls	726
3.2.1.10	Extended Controls API	733

3.2.1.11	Camera Control Reference	736
3.2.1.12	Flash Control Reference	744
3.2.1.13	Image Source Control Reference	746
3.2.1.14	Image Process Control Reference	747
3.2.1.15	Codec Control Reference	748
3.2.1.16	Stateless Codec Control Reference	779
3.2.1.17	JPEG Control Reference	797
3.2.1.18	Digital Video Control Reference	798
3.2.1.19	RF Tuner Control Reference	800
3.2.1.20	FM Transmitter Control Reference	802
3.2.1.21	FM Receiver Control Reference	804
3.2.1.22	Detect Control Reference	805
3.2.1.23	Colorimetry Control Reference	806
3.2.1.24	Guidelines for Video4Linux pixel format 4CCs	806
3.2.1.25	Data Formats	807
3.2.1.26	Single- and multi-planar APIs	808
3.2.1.27	Cropping, compositing and scaling - the SELECTION API	809
3.2.1.28	Image Cropping, Insertion and Scaling - the CROP API	814
3.2.1.29	Streaming Parameters	819
3.2.2	Image Formats	820
3.2.2.1	Single-planar format structure	820
3.2.2.2	Multi-planar format structures	825
3.2.2.3	Standard Image Formats	826
3.2.2.4	Indexed Format	827
3.2.2.5	RGB Formats	827
3.2.2.6	Raw Bayer Formats	830
3.2.2.7	YUV Formats	837
3.2.2.8	HSV Formats	851
3.2.2.9	Depth Formats	851
3.2.2.10	Compressed Formats	853
3.2.2.11	SDR Formats	856
3.2.2.12	Touch Formats	860
3.2.2.13	Metadata Formats	861
3.2.2.14	Reserved Format Identifiers	945
3.2.2.15	Colorspaces	947
3.2.2.16	Defining Colorspaces in V4L2	949
3.2.2.17	Detailed Colorspace Descriptions	952
3.2.2.18	Detailed Transfer Function Descriptions	960
3.2.3	Input/Output	961
3.2.3.1	Read/Write	961
3.2.3.2	Streaming I/O (Memory Mapping)	962
3.2.3.3	Streaming I/O (User Pointers)	966
3.2.3.4	Streaming I/O (DMA buffer importing)	968
3.2.3.5	Asynchronous I/O	970
3.2.3.6	Buffers	970
3.2.3.7	Field Order	981
3.2.4	Interfaces	983
3.2.4.1	Video Capture Interface	983
3.2.4.2	Video Overlay Interface	986
3.2.4.3	Video Output Interface	990
3.2.4.4	Video Output Overlay Interface	992

3.2.4.5	Video Memory-To-Memory Interface	994
3.2.4.6	Raw VBI Data Interface	1027
3.2.4.7	Sliced VBI Data Interface	1032
3.2.4.8	Radio Interface	1039
3.2.4.9	RDS Interface	1040
3.2.4.10	Software Defined Radio Interface (SDR)	1042
3.2.4.11	Touch Devices	1043
3.2.4.12	Event Interface	1044
3.2.4.13	Sub-device Interface	1044
3.2.4.14	Metadata Interface	1063
3.2.5	Libv4l Userspace Library	1064
3.2.5.1	Introduction	1064
3.2.6	Changes	1066
3.2.6.1	Differences between V4L and V4L2	1066
3.2.6.2	Changes of the V4L2 API	1075
3.2.7	Function Reference	1094
3.2.7.1	V4L2 close()	1094
3.2.7.2	V4L2 ioctl()	1095
3.2.7.3	ioctl VIDIOC_CREATE_BUFS	1096
3.2.7.4	ioctl VIDIOC_CROPCAP	1099
3.2.7.5	ioctl VIDIOC_DBG_G_CHIP_INFO	1101
3.2.7.6	ioctl VIDIOC_DBG_G_REGISTER, VIDIOC_DBG_S_REGISTER	1103
3.2.7.7	ioctl VIDIOC_DECODER_CMD, VIDIOC_TRY_DECODER_CMD	1105
3.2.7.8	ioctl VIDIOC_DQEVENT	1108
3.2.7.9	ioctl VIDIOC_DV_TIMINGS_CAP, VIDIOC_SUBDEV_DV_TIMINGS_CAP	1113
3.2.7.10	ioctl VIDIOC_ENCODER_CMD, VIDIOC_TRY_ENCODER_CMD	1115
3.2.7.11	ioctl VIDIOC_ENUMAUDIO	1118
3.2.7.12	ioctl VIDIOC_ENUMAUDOUT	1119
3.2.7.13	ioctl VIDIOC_ENUM_DV_TIMINGS, VIDIOC_SUBDEV_ENUM_DV_TIMINGS	1120
3.2.7.14	ioctl VIDIOC_ENUM_FMT	1121
3.2.7.15	ioctl VIDIOC_ENUM_FRAMEIZES	1126
3.2.7.16	ioctl VIDIOC_ENUM_FRAMEINTERVALS	1129
3.2.7.17	ioctl VIDIOC_ENUM_FREQ_BANDS	1131
3.2.7.18	ioctl VIDIOC_ENUMINPUT	1134
3.2.7.19	ioctl VIDIOC_ENUMOUTPUT	1137
3.2.7.20	ioctl VIDIOC_ENUMSTD, VIDIOC_SUBDEV_ENUMSTD	1139
3.2.7.21	ioctl VIDIOC_EXPBUF	1144
3.2.7.22	ioctl VIDIOC_G_AUDIO, VIDIOC_S_AUDIO	1146
3.2.7.23	ioctl VIDIOC_G_AUDOUT, VIDIOC_S_AUDOUT	1148
3.2.7.24	ioctl VIDIOC_G_CROP, VIDIOC_S_CROP	1149
3.2.7.25	ioctl VIDIOC_G_CTRL, VIDIOC_S_CTRL	1151
3.2.7.26	ioctl VIDIOC_G_DV_TIMINGS, VIDIOC_S_DV_TIMINGS	1152
3.2.7.27	ioctl VIDIOC_G_EDID, VIDIOC_S_EDID, VIDIOC_SUBDEV_G_EDID, VIDIOC_SUBDEV_S_EDID	1157
3.2.7.28	ioctl VIDIOC_G_ENC_INDEX	1159
3.2.7.29	ioctl VIDIOC_G_EXT_CTRL, VIDIOC_S_EXT_CTRL, VIDIOC_TRY_EXT_CTRL	1161
3.2.7.30	ioctl VIDIOC_G_FBUF, VIDIOC_S_FBUF	1167
3.2.7.31	ioctl VIDIOC_G_FMT, VIDIOC_S_FMT, VIDIOC_TRY_FMT	1173
3.2.7.32	ioctl VIDIOC_G_FREQUENCY, VIDIOC_S_FREQUENCY	1176

3.2.7.33	ioctl VIDIOC_G_INPUT, VIDIOC_S_INPUT	1177
3.2.7.34	ioctl VIDIOC_G_JPEGCOMP, VIDIOC_S_JPEGCOMP	1178
3.2.7.35	ioctl VIDIOC_G_MODULATOR, VIDIOC_S_MODULATOR	1180
3.2.7.36	ioctl VIDIOC_G_OUTPUT, VIDIOC_S_OUTPUT	1183
3.2.7.37	ioctl VIDIOC_G_PARM, VIDIOC_S_PARM	1184
3.2.7.38	ioctl VIDIOC_G_PRIORITY, VIDIOC_S_PRIORITY	1188
3.2.7.39	ioctl VIDIOC_G_SELECTION, VIDIOC_S_SELECTION	1189
3.2.7.40	ioctl VIDIOC_G_SLICED_VBI_CAP	1192
3.2.7.41	ioctl VIDIOC_G_STD, VIDIOC_S_STD, VIDIOC_SUBDEV_G_STD, VIDIOC_SUBDEV_S_STD	1194
3.2.7.42	ioctl VIDIOC_G_TUNER, VIDIOC_S_TUNER	1195
3.2.7.43	ioctl VIDIOC_LOG_STATUS	1202
3.2.7.44	ioctl VIDIOC_OVERLAY	1203
3.2.7.45	ioctl VIDIOC_PREPARE_BUF	1204
3.2.7.46	ioctl VIDIOC_QBUF, VIDIOC_DQBUF	1205
3.2.7.47	ioctl VIDIOC_QUERYBUF	1207
3.2.7.48	ioctl VIDIOC_QUERYCAP	1208
3.2.7.49	ioctl VIDIOC_QUERYCTRL, VIDIOC_QUERY_EXT_CTRL and VID- IOC_QUERYMENU	1213
3.2.7.50	ioctl VIDIOC_QUERY_DV_TIMINGS	1221
3.2.7.51	ioctl VIDIOC_QUERYSTD, VIDIOC_SUBDEV_QUERYSTD	1222
3.2.7.52	ioctl VIDIOC_REQBUFS	1224
3.2.7.53	ioctl VIDIOC_S_HW_FREQ_SEEK	1226
3.2.7.54	ioctl VIDIOC_STREAMON, VIDIOC_STREAMOFF	1228
3.2.7.55	ioctl VIDIOC_SUBDEV_ENUM_FRAME_INTERVAL	1230
3.2.7.56	ioctl VIDIOC_SUBDEV_ENUM_FRAME_SIZE	1231
3.2.7.57	ioctl VIDIOC_SUBDEV_ENUM_MBUS_CODE	1233
3.2.7.58	ioctl VIDIOC_SUBDEV_G_CROP, VIDIOC_SUBDEV_S_CROP	1235
3.2.7.59	ioctl VIDIOC_SUBDEV_G_FMT, VIDIOC_SUBDEV_S_FMT	1236
3.2.7.60	ioctl VIDIOC_SUBDEV_G_FRAME_INTERVAL, VIDIOC_SUBDEV_S_FRAME_INTERVAL	1238
3.2.7.61	ioctl VIDIOC_SUBDEV_G_SELECTION, VIDIOC_SUBDEV_S_SELECTION	1240
3.2.7.62	ioctl VIDIOC_SUBDEV_QUERYCAP	1242
3.2.7.63	ioctl VIDIOC_SUBSCRIBE_EVENT, VIDIOC_UNSUBSCRIBE_EVENT	1243
3.2.7.64	V4L2 mmap()	1245
3.2.7.65	V4L2 munmap()	1247
3.2.7.66	V4L2 open()	1248
3.2.7.67	V4L2 poll()	1249
3.2.7.68	V4L2 read()	1251
3.2.7.69	V4L2 select()	1252
3.2.7.70	V4L2 write()	1254
3.2.8	Common definitions for V4L2 and V4L2 subdev interfaces	1255
3.2.8.1	Common selection definitions	1255
3.2.9	Video For Linux Two Header File	1256
3.2.9.1	videodev2.h	1256
3.2.10	Video Capture Example	1313
3.2.10.1	file: media/v4l/capture.c	1313
3.2.11	Video Grabber example using libv4l	1325
3.2.11.1	file: media/v4l/v4l2grab.c	1325
3.2.12	References	1328

3.2.12.1	CEA 608-E	1328
3.2.12.2	EN 300 294	1328
3.2.12.3	ETS 300 231	1328
3.2.12.4	ETS 300 706	1328
3.2.12.5	ISO 13818-1	1328
3.2.12.6	ISO 13818-2	1328
3.2.12.7	ITU BT.470	1329
3.2.12.8	ITU BT.601	1329
3.2.12.9	ITU BT.653	1329
3.2.12.10	ITU BT.709	1329
3.2.12.11	ITU BT.1119	1329
3.2.12.12	ITU-T Rec. H.264 Specification (04/2017 Edition)	1329
3.2.12.13	ITU H.265/HEVC	1329
3.2.12.14	JFIF	1330
3.2.12.15	ITU-T.81	1330
3.2.12.16	W3C JPEG JFIF	1330
3.2.12.17	MPTE 12M	1330
3.2.12.18	MPTE 170M	1330
3.2.12.19	MPTE 240M	1330
3.2.12.20	MPTE RP 431-2	1330
3.2.12.21	MPTE ST 2084	1331
3.2.12.22	RGB	1331
3.2.12.23	YCC	1331
3.2.12.24	YCC	1331
3.2.12.25	pRGB	1331
3.2.12.26	ITU BT.2020	1331
3.2.12.27	EBU Tech 3213	1332
3.2.12.28	EBU Tech 3321	1332
3.2.12.29	IEC 62106	1332
3.2.12.30	NRSC-4-B	1332
3.2.12.31	ISO 12232:2006	1332
3.2.12.32	CEA-861-E	1332
3.2.12.33	VESA DMT	1332
3.2.12.34	EDID	1333
3.2.12.35	EDCP	1333
3.2.12.36	EDMI	1333
3.2.12.37	EDMI2	1333
3.2.12.38	BP	1333
3.2.12.39	oynton	1333
3.2.12.40	colimg	1334
3.2.12.41	VP8	1334
3.2.12.42	VP9	1334
3.2.13	Revision and Copyright	1334
3.2.14	Revision History	1335
3.3	Part II - Digital TV API	1339
3.3.1	Introduction	1339
3.3.1.1	What you need to know	1339
3.3.1.2	History	1340
3.3.1.3	Overview	1340
3.3.1.4	Linux Digital TV Devices	1341
3.3.1.5	API include files	1342

3.3.2	Digital TV Frontend API	1342
3.3.2.1	Querying frontend information	1343
3.3.2.2	Querying frontend status and statistics	1343
3.3.2.3	Property types	1343
3.3.2.4	Frontend Function Calls	1380
3.3.3	Digital TV Demux Device	1391
3.3.3.1	Demux Data Types	1392
3.3.3.2	Demux Function Calls	1396
3.3.4	Digital TV CA Device	1415
3.3.4.1	CA Data Types	1415
3.3.4.2	CA Function Calls	1418
3.3.4.3	The High level CI API	1425
3.3.5	Digital TV Network API	1427
3.3.5.1	Digital TV net Function Calls	1428
3.3.6	Digital TV Deprecated APIs	1431
3.3.6.1	Digital TV Frontend legacy API (a. k. a. DVBv3)	1431
3.3.7	Examples	1441
3.3.8	Digital TV uAPI header files	1441
3.3.8.1	Digital TV uAPI headers	1441
3.3.9	Revision and Copyright	1472
3.3.10	Revision History	1473
3.4	Part III - Remote Controller API	1473
3.4.1	Introduction	1473
3.4.2	Remote Controller' s sysfs nodes	1474
3.4.2.1	/sys/class/rc/	1474
3.4.2.2	/sys/class/rc/rcN/	1474
3.4.2.3	/sys/class/rc/rcN/protocols	1474
3.4.2.4	/sys/class/rc/rcN/filter	1474
3.4.2.5	/sys/class/rc/rcN/filter_mask	1475
3.4.2.6	/sys/class/rc/rcN/wakeup_protocols	1475
3.4.2.7	/sys/class/rc/rcN/wakeup_filter	1475
3.4.2.8	/sys/class/rc/rcN/wakeup_filter_mask	1475
3.4.3	Remote Controller Protocols and Scancodes	1476
3.4.3.1	rc-5 (RC_PROTO_RC5)	1476
3.4.3.2	rc-5-sz (RC_PROTO_RC5_SZ)	1476
3.4.3.3	rc-5x-20 (RC_PROTO_RC5X_20)	1477
3.4.3.4	jvc (RC_PROTO_JVC)	1477
3.4.3.5	sony-12 (RC_PROTO_SONY12)	1477
3.4.3.6	sony-15 (RC_PROTO_SONY15)	1477
3.4.3.7	sony-20 (RC_PROTO_SONY20)	1478
3.4.3.8	nec (RC_PROTO_NEC)	1478
3.4.3.9	nec-x (RC_PROTO_NECX)	1478
3.4.3.10	nec-32 (RC_PROTO_NEC32)	1478
3.4.3.11	sanyo (RC_PROTO_SANYO)	1479
3.4.3.12	mcir2-kbd (RC_PROTO_MCIR2_KBD)	1479
3.4.3.13	mcir2-mse (RC_PROTO_MCIR2_MSE)	1479
3.4.3.14	rc-6-0 (RC_PROTO_RC6_0)	1479
3.4.3.15	rc-6-6a-20 (RC_PROTO_RC6_6A_20)	1479
3.4.3.16	rc-6-6a-24 (RC_PROTO_RC6_6A_24)	1479
3.4.3.17	rc-6-6a-32 (RC_PROTO_RC6_6A_32)	1479
3.4.3.18	rc-6-mce (RC_PROTO_RC6_MCE)	1480

3.4.3.19	sharp (RC_PROTO_SHARP)	1480
3.4.3.20	xmp (RC_PROTO_XMP)	1480
3.4.3.21	cec (RC_PROTO_CEC)	1480
3.4.3.22	imon (RC_PROTO_IMON)	1480
3.4.3.23	rc-mm-12 (RC_PROTO_RCMM12)	1480
3.4.3.24	rc-mm-24 (RC_PROTO_RCMM24)	1480
3.4.3.25	rc-mm-32 (RC_PROTO_RCMM32)	1481
3.4.3.26	xbox-dvd (RC_PROTO_XBOX_DVD)	1481
3.4.4	Remote controller tables	1481
3.4.5	Changing default Remote Controller mappings	1484
3.4.5.1	file: uapi/v4l/keytable.c	1484
3.4.6	LIRC Device Interface	1488
3.4.6.1	Introduction	1488
3.4.6.2	LIRC modes	1488
3.4.6.3	Data types used by LIRC_MODE_SCANCODE	1489
3.4.6.4	BPF based IR decoder	1491
3.4.6.5	LIRC Function Reference	1491
3.4.6.6	LIRC Header File	1506
3.4.7	Revision and Copyright	1511
3.4.8	Revision History	1511
3.5	Part IV - Media Controller API	1511
3.5.1	Introduction	1511
3.5.2	Media device model	1512
3.5.3	Types and flags used to represent the media graph elements	1512
3.5.4	Request API	1517
3.5.4.1	General Usage	1518
3.5.4.2	Request Allocation	1518
3.5.4.3	Request Preparation	1518
3.5.4.4	Request Submission	1518
3.5.4.5	Recycling and Destruction	1519
3.5.4.6	Example for a Codec Device	1519
3.5.4.7	Example for a Simple Capture Device	1521
3.5.5	Function Reference	1521
3.5.5.1	media open()	1521
3.5.5.2	media close()	1522
3.5.5.3	media ioctl()	1523
3.5.5.4	ioctl MEDIA_IOC_DEVICE_INFO	1524
3.5.5.5	ioctl MEDIA_IOC_G_TOPOLOGY	1526
3.5.5.6	ioctl MEDIA_IOC_ENUM_ENTITIES	1529
3.5.5.7	ioctl MEDIA_IOC_ENUM_LINKS	1531
3.5.5.8	ioctl MEDIA_IOC_SETUP_LINK	1532
3.5.5.9	ioctl MEDIA_IOC_REQUEST_ALLOC	1533
3.5.5.10	request close()	1534
3.5.5.11	request ioctl()	1535
3.5.5.12	request poll()	1536
3.5.5.13	ioctl MEDIA_REQUEST_IOC_QUEUE	1537
3.5.5.14	ioctl MEDIA_REQUEST_IOC_REINIT	1538
3.5.6	Media Controller Header File	1539
3.5.6.1	media.h	1539
3.5.7	Revision and Copyright	1548
3.5.8	Revision History	1548

3.6	Part V - Consumer Electronics Control API	1549
3.6.1	Introduction	1549
3.6.2	Function Reference	1549
3.6.2.1	cec open()	1549
3.6.2.2	cec close()	1550
3.6.2.3	cec ioctl()	1551
3.6.2.4	cec poll()	1552
3.6.2.5	ioctl CEC_ADAP_G_CAPS	1553
3.6.2.6	ioctls CEC_ADAP_G_LOG_ADDRS and CEC_ADAP_S_LOG_ADDRS	1556
3.6.2.7	ioctls CEC_ADAP_G_PHYS_ADDR and CEC_ADAP_S_PHYS_ADDR	1561
3.6.2.8	ioctl CEC_ADAP_G_CONNECTOR_INFO	1562
3.6.2.9	ioctl CEC_DQEVENT	1563
3.6.2.10	ioctls CEC_G_MODE and CEC_S_MODE	1568
3.6.2.11	ioctls CEC_RECEIVE and CEC_TRANSMIT	1573
3.6.3	CEC Pin Framework Error Injection	1579
3.6.3.1	Basic Syntax	1580
3.6.3.2	Clear Error Injections	1581
3.6.3.3	Receive Messages	1581
3.6.3.4	Transmit Messages	1582
3.6.3.5	Custom Pulses	1583
3.6.4	CEC Header File	1584
3.6.4.1	cec.h	1584
3.6.5	Revision and Copyright	1607
3.6.6	Revision History	1607
3.7	Generic Error Codes	1607
3.8	Glossary	1608
3.9	GNU Free Documentation License	1610
3.9.1	0. PREAMBLE	1610
3.9.2	1. APPLICABILITY AND DEFINITIONS	1611
3.9.3	2. VERBATIM COPYING	1612
3.9.4	3. COPYING IN QUANTITY	1612
3.9.5	4. MODIFICATIONS	1612
3.9.6	5. COMBINING DOCUMENTS	1614
3.9.7	6. COLLECTIONS OF DOCUMENTS	1614
3.9.8	7. AGGREGATION WITH INDEPENDENT WORKS	1614
3.9.9	8. TRANSLATION	1615
3.9.10	9. TERMINATION	1615
3.9.11	10. FUTURE REVISIONS OF THIS LICENSE	1615
3.9.12	Addendum	1615
3.10	Video4Linux (V4L) driver-specific documentation	1616
3.10.1	MIPI CCS camera sensor driver	1616
3.10.1.1	Pixel Array sub-device	1616
3.10.1.2	Binner	1616
3.10.1.3	Scaler	1617
3.10.1.4	Digital and analogue crop	1617
3.10.1.5	Private controls	1617
3.10.2	The cx2341x driver	1618
3.10.2.1	Non-compressed file format	1618
3.10.2.2	Format of embedded V4L2_MPEG_STREAM_VBI_FMT_IVTV VBI data	1620
3.10.3	Hantro video decoder driver	1621
3.10.4	i.MX Video Capture Driver	1621

3.10.4.1	Events	1621
3.10.4.2	Controls	1621
3.10.5	Maxim Integrated MAX2175 RF to bits tuner driver	1623
3.10.5.1	V4L2_CID_MAX2175_I2S_ENABLE	1623
3.10.5.2	V4L2_CID_MAX2175_HSL5	1623
3.10.5.3	V4L2_CID_MAX2175_RX_MODE (menu)	1623
3.10.6	Vaio Picturebook Motion Eye Camera Driver	1624
3.10.6.1	Private API	1624
3.10.7	OMAP 3 Image Signal Processor (ISP) driver	1625
3.10.7.1	Events	1625
3.10.7.2	Private IOCTLs	1625
3.10.7.3	CCDC and preview block IOCTLs	1626
3.10.7.4	Statistic blocks IOCTLs	1627
3.10.7.5	VIDIOC_OMAP3ISP_STAT_EN	1627
3.10.7.6	VIDIOC_OMAP3ISP_AEWB_CFG, VIDIOC_OMAP3ISP_HIST_CFG and VIDIOC_OMAP3ISP_AF_CFG	1627
3.10.7.7	VIDIOC_OMAP3ISP_STAT_REQ	1628
3.10.7.8	References	1628
3.10.8	The Linux USB Video Class (UVC) driver	1628
3.10.8.1	Extension Unit (XU) support	1628

Copyright © 1991-: LinuxTV Developers

MEDIA SUBSYSTEM ADMIN AND USER GUIDE

This section contains usage information about media subsystem and its supported drivers.

Please see:

[Documentation/userspace-api/media/index.rst](#)

- for the userspace APIs used on media devices.

[Documentation/driver-api/media/index.rst](#)

- for driver development information and Kernel APIs used by media devices;

1.1 The media subsystem

1.1.1 Introduction

The media subsystem consists on Linux support for several different types of devices:

- Audio and video grabbers;
- PC and Laptop Cameras;
- Complex cameras found on Embedded hardware;
- Analog and digital TV;
- HDMI Customer Electronics Control (CEC);
- Multi-touch input devices;
- Remote Controllers;
- Media encoders and decoders.

Due to the diversity of devices, the subsystem provides several different APIs:

- Remote Controller API;
- HDMI CEC API;
- Video4Linux API;
- Media controller API;
- Video4Linux Request API (experimental);
- Digital TV API (also known as DVB API).

1.1.2 Building support for a media device

The first step is to download the Kernel's source code, either via a distribution-specific source file or via the Kernel's main git tree¹.

Please notice, however, that, if:

- you're a braveheart and want to experiment with new stuff;
- if you want to report a bug;
- if you're developing new patches

you should use the main media development tree master branch:

https://git.linuxtv.org/media_tree.git/

In this case, you may find some useful information at the [LinuxTv wiki pages](#):

https://linuxtv.org/wiki/index.php/How_to_Obtain,_Build_and_Install_V4L-DVB_Device_Drivers

1.1.2.1 Configuring the Linux Kernel

You can access a menu of Kernel building options with:

```
$ make menuconfig
```

Then, select all desired options and exit it, saving the configuration.

The changed configuration will be at the `.config` file. It would look like:

```
...
# CONFIG_RC_CORE is not set
# CONFIG_CEC_CORE is not set
CONFIG_MEDIA_SUPPORT=m
CONFIG_MEDIA_SUPPORT_FILTER=y
...
```

The media subsystem is controlled by those menu configuration options:

```
Device Drivers --->
  <M> Remote Controller support --->
    [ ] HDMI CEC RC integration
    [ ] Enable CEC error injection support
    [*] HDMI CEC drivers --->
  <*> Multimedia support --->
```

The `Remote Controller support` option enables the core support for remote controllers².

The `HDMI CEC RC integration` option enables integration of HDMI CEC with Linux, allowing to receive data via HDMI CEC as if it were produced by a remote controller directly connected to the machine.

¹ The upstream Linux Kernel development tree is located at <https://git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux.git/>

² `Remote Controller support` should also be enabled if you want to use some TV card drivers that may depend on the remote controller core support.

The HDMI CEC drivers option allow selecting platform and USB drivers that receives and/or transmits CEC codes via HDMI interfaces³.

The last option (Multimedia support) enables support for cameras, audio/video grabbers and TV.

The media subsystem support can either be built together with the main Kernel or as a module. For most use cases, it is preferred to have it built as modules.

Note: Instead of using a menu, the Kernel provides a script with allows enabling configuration options directly. To enable media support and remote controller support using Kernel modules, you could use:

```
$ scripts/config -m RC_CORE
$ scripts/config -m MEDIA_SUPPORT
```

Media dependencies

It should be noticed that enabling the above from a clean config is usually not enough. The media subsystem depends on several other Linux core support in order to work.

For example, most media devices use a serial communication bus in order to talk with some peripherals. Such bus is called I²C (Inter-Integrated Circuit). In order to be able to build support for such hardware, the I²C bus support should be enabled, either via menu or with:

```
./scripts/config -m I2C
```

Another example: the remote controller core requires support for input devices, with can be enabled with:

```
./scripts/config -m INPUT
```

Other core functionality may also be needed (like PCI and/or USB support), depending on the specific driver(s) you would like to enable.

Enabling Remote Controller Support

The remote controller menu allows selecting drivers for specific devices. It's menu looks like this:

```
--- Remote Controller support
<M>  Compile Remote Controller keymap modules
[*]  LIRC user interface
[*]  Support for eBPF programs attached to lirc devices
[*]  Remote controller decoders --->
[*]  Remote Controller devices --->
```

³ Please notice that the DRM subsystem also have drivers for GPUs that use the media HDMI CEC support. Those GPU-specific drivers are selected via the Graphics support menu, under Device Drivers. When a GPU driver supports HDMI CEC, it will automatically enable the CEC core support at the media subsystem.

The `Compile Remote Controller keymap modules` option creates key maps for several popular remote controllers.

The `LIRC user interface` option adds enhanced functionality when using the `lirc` program, by enabling an API that allows userspace to receive raw data from remote controllers.

The `Support for eBPF programs attached to lirc devices` option allows the usage of special programs (called eBPF) that would allow applications to add extra remote controller decoding functionality to the Linux Kernel.

The `Remote controller decoders` option allows selecting the protocols that will be recognized by the Linux Kernel. Except if you want to disable some specific decoder, it is suggested to keep all sub-options enabled.

The `Remote Controller devices` allows you to select the drivers that would be needed to support your device.

The same configuration can also be set via the `script/config` script. So, for instance, in order to support the ITE remote controller driver (found on Intel NUCs and on some ASUS x86 desktops), you could do:

```
$ scripts/config -e INPUT
$ scripts/config -e ACPI
$ scripts/config -e MODULES
$ scripts/config -m RC_CORE
$ scripts/config -e RC_DEVICES
$ scripts/config -e RC_DECODERS
$ scripts/config -m IR_RC5_DECODER
$ scripts/config -m IR_ITE_CIR
```

Enabling HDMI CEC Support

The HDMI CEC support is set automatically when a driver requires it. So, all you need to do is to enable support either for a graphics card that needs it or by one of the existing HDMI drivers.

The HDMI-specific drivers are available at the `HDMI CEC drivers` menu⁴:

```
--- HDMI CEC drivers
< > ChromeOS EC CEC driver
< > Amlogic Meson A0 CEC driver
< > Amlogic Meson G12A A0 CEC driver
< > Generic GPIO-based CEC driver
< > Samsung S5P CEC driver
< > STMicroelectronics STiH4xx HDMI CEC driver
< > STMicroelectronics STM32 HDMI CEC driver
< > Tegra HDMI CEC driver
< > SECO Boards HDMI CEC driver
[ ] SECO Boards IR RC5 support
< > Pulse Eight HDMI CEC
< > RainShadow Tech HDMI CEC
```

⁴ The above contents is just an example. The actual options for HDMI devices depends on the system's architecture and may vary on new Kernels.

Enabling Media Support

The Media menu has a lot more options than the remote controller menu. Once selected, you should see the following options:

```

--- Media support
[ ] Filter media drivers
[*] Autoselect ancillary drivers
    Media device types --->
    Media core support --->
    Video4Linux options --->
    Media controller options --->
    Digital TV options --->
    HDMI CEC options --->
    Media drivers --->
    Media ancillary drivers --->

```

Except if you know exactly what you're doing, or if you want to build a driver for a SoC platform, it is strongly recommended to keep the Autoselect ancillary drivers option turned on, as it will auto-select the needed I²C ancillary drivers.

There are now two ways to select media device drivers, as described below.

Filter media drivers menu

This menu is meant to easy setup for PC and Laptop hardware. It works by letting the user to specify what kind of media drivers are desired, with those options:

```

[ ] Cameras and video grabbers
[ ] Analog TV
[ ] Digital TV
[ ] AM/FM radio receivers/transmitters
[ ] Software defined radio
[ ] Platform-specific devices
[ ] Test drivers

```

So, if you want to add support to a camera or video grabber only, select just the first option. Multiple options are allowed.

Once the options on this menu are selected, the building system will auto-select the needed core drivers in order to support the selected functionality.

Note: Most TV cards are hybrid: they support both Analog TV and Digital TV.

If you have an hybrid card, you may need to enable both Analog TV and Digital TV at the menu.

When using this option, the defaults for the media support core functionality are usually good enough to provide the basic functionality for the driver. Yet, you could manually enable some desired extra (optional) functionality using the settings under each of the following Media support sub-menus:

```
Media core support --->
Video4Linux options --->
Media controller options --->
Digital TV options --->
HDMI CEC options --->
```

Once you select the desired filters, the drivers that matches the filtering criteria will be available at the Media support->Media drivers sub-menu.

Media Core Support menu without filtering

If you disable the Filter media drivers menu, all drivers available for your system whose dependencies are met should be shown at the Media drivers menu.

Please notice, however, that you should first ensure that the Media Core Support menu has all the core functionality your drivers would need, as otherwise the corresponding device drivers won't be shown.

Example

In order to enable modular support for one of the boards listed on [this table](#), with modular media core modules, the .config file should contain those lines:

```
CONFIG_MODULES=y
CONFIG_USB=y
CONFIG_I2C=y
CONFIG_INPUT=y
CONFIG_RC_CORE=m
CONFIG_MEDIA_SUPPORT=m
CONFIG_MEDIA_SUPPORT_FILTER=y
CONFIG_MEDIA_ANALOG_TV_SUPPORT=y
CONFIG_MEDIA_DIGITAL_TV_SUPPORT=y
CONFIG_MEDIA_USB_SUPPORT=y
CONFIG_VIDEO_CX231XX=y
CONFIG_VIDEO_CX231XX_DVB=y
```

1.1.2.2 Building and installing a new Kernel

Once the .config file has everything needed, all it takes to build is to run the make command:

```
$ make
```

And then install the new Kernel and its modules:

```
$ sudo make modules_install
$ sudo make install
```


1.1.2.3 Building just the new media drivers and core

Running a new development Kernel from the development tree is usually risky, because it may have experimental changes that may have bugs. So, there are some ways to build just the new drivers, using alternative trees.

There is the [Linux Kernel backports project](#), which contains newer drivers meant to be compiled against stable Kernels.

The LinuxTV developers, who are responsible for maintaining the media subsystem also maintain a backport tree, with just the media drivers daily updated from the newest kernel. Such tree is available at:

https://git.linuxtv.org/media_build.git/

It should be noticed that, while it should be relatively safe to use the `media_build` tree for testing purposes, there are no warranties that it would work (or even build) on a random Kernel. This tree is maintained using a “best-efforts” principle, as time permits us to fix issues there.

If you notice anything wrong on it, feel free to submit patches at the Linux media subsystem’s mailing list: media@vger.kernel.org. Please add `[PATCH media-build]` at the e-mail’s subject if you submit a new patch for the `media-build`.

Before using it, you should run:

```
$ ./build
```

Note:

- 1) you may need to run it twice if the `media-build` tree gets updated;
- 2) you may need to do a `make distclean` if you had built it in the past for a different Kernel version than the one you’re currently using;
- 3) by default, it will use the same config options for media as the ones defined on the Kernel you’re running.

In order to select different drivers or different config options, use:

```
$ make menuconfig
```

Then, you can build and install the new drivers:

```
$ make && sudo make install
```

This will override the previous media drivers that your Kernel were using.

1.1.3 Infrared remote control support in video4linux drivers

Authors: Gerd Hoffmann, Mauro Carvalho Chehab

1.1.3.1 Basics

Most analog and digital TV boards support remote controllers. Several of them have a micro-processor that receives the IR carriers, convert into pulse/space sequences and then to scan codes, returning such codes to userspace (“scancode mode”). Other boards return just the pulse/space sequences (“raw mode”).

The support for remote controller in scancode mode is provided by the standard Linux input layer. The support for raw mode is provided via LIRC.

In order to check the support and test it, it is suggested to download the [v4l-utils](#). It provides two tools to handle remote controllers:

- `ir-keytable`: provides a way to query the remote controller, list the protocols it supports, enable in-kernel support for IR decoder or switch the protocol and to test the reception of scan codes;
- `ir-ctl`: provide tools to handle remote controllers that support raw mode via LIRC interface.

Usually, the remote controller module is auto-loaded when the TV card is detected. However, for a few devices, you need to manually load the `ir-kbd-i2c` module.

1.1.3.2 How it works

The modules register the remote as keyboard within the linux input layer, i.e. you’ll see the keys of the remote as normal key strokes (if `CONFIG_INPUT_KEYBOARD` is enabled).

Using the event devices (`CONFIG_INPUT_EVDEV`) it is possible for applications to access the remote via `/dev/input/event<n>` devices. The `udev/systemd` will automatically create the devices. If you install the [v4l-utils](#), it may also automatically load a different keytable than the default one. Please see [v4l-utils ir-keytable.1](#) man page for details.

The `ir-keytable` tool is nice for trouble shooting, i.e. to check whenever the input device is really present, which of the devices it is, check whenever pressing keys on the remote actually generates events and the like. You can also use any other input utility that changes the keymaps, like the `input kbd` utility.

Using with lircd

The latest versions of the `lircd` daemon supports reading events from the linux input layer (via event device). It also supports receiving IR codes in `lirc` mode.

Using without lircd

Xorg recognizes several IR keycodes that have its numerical value lower than 247. With the advent of Wayland, the input driver got updated too, and should now accept all keycodes. Yet, you may want to just reassign the keycodes to something that your favorite media application likes.

This can be done by setting `v4l-utils` to load your own keytable in runtime. Please read `ir-keytable.1` man page for details.

1.1.4 Digital TV

1.1.4.1 Using the Digital TV Framework

Introduction

One significant difference between Digital TV and Analogue TV that the unwary (like myself) should consider is that, although the component structure of DVB-T cards are substantially similar to Analogue TV cards, they function in substantially different ways.

The purpose of an Analogue TV is to receive and display an Analogue Television signal. An Analogue TV signal (otherwise known as composite video) is an analogue encoding of a sequence of image frames (25 frames per second in Europe) rasterised using an interlacing technique. Interlacing takes two fields to represent one frame. Therefore, an Analogue TV card for a PC has the following purpose:

- Tune the receiver to receive a broadcast signal
- demodulate the broadcast signal
- demultiplex the analogue video signal and analogue audio signal.

Note: some countries employ a digital audio signal embedded within the modulated composite analogue signal - using NICAM signaling.)

- digitize the analogue video signal and make the resulting datastream available to the data bus.

The digital datastream from an Analogue TV card is generated by circuitry on the card and is often presented uncompressed. For a PAL TV signal encoded at a resolution of 768x576 24-bit color pixels over 25 frames per second - a fair amount of data is generated and must be processed by the PC before it can be displayed on the video monitor screen. Some Analogue TV cards for PCs have onboard MPEG2 encoders which permit the raw digital data stream to be presented to the PC in an encoded and compressed form - similar to the form that is used in Digital TV.

The purpose of a simple budget digital TV card (DVB-T,C or S) is to simply:

- Tune the received to receive a broadcast signal. * Extract the encoded digital datastream from the broadcast signal.
- Make the encoded digital datastream (MPEG2) available to the data bus.

The significant difference between the two is that the tuner on the analogue TV card spits out an Analogue signal, whereas the tuner on the digital TV card spits out a compressed encoded digital datastream. As the signal is already digitised, it is trivial to pass this datastream to the PC databus with minimal additional processing and then extract the digital video and audio datastreams passing them to the appropriate software or hardware for decoding and viewing.

Getting the card going

The Device Driver API for DVB under Linux will the following device nodes via the devfs filesystem:

- `/dev/dvb/adapter0/demux0`
- `/dev/dvb/adapter0/dvr0`
- `/dev/dvb/adapter0/frontend0`

The `/dev/dvb/adapter0/dvr0` device node is used to read the MPEG2 Data Stream and the `/dev/dvb/adapter0/frontend0` device node is used to tune the frontend tuner module. The `/dev/dvb/adapter0/demux0` is used to control what programs will be received.

Depending on the card's feature set, the Device Driver API could also expose other device nodes:

- `/dev/dvb/adapter0/ca0`
- `/dev/dvb/adapter0/audio0`
- `/dev/dvb/adapter0/net0`
- `/dev/dvb/adapter0/osd0`
- `/dev/dvb/adapter0/video0`

The `/dev/dvb/adapter0/ca0` is used to decode encrypted channels. The other device nodes are found only on devices that use the av7110 driver, with is now obsoleted, together with the extra API whose such devices use.

Receiving a digital TV channel

This section attempts to explain how it works and how this affects the configuration of a Digital TV card.

On this example, we're considering tuning into DVB-T channels in Australia, at the Melbourne region.

The frequencies broadcast by Mount Dandenong transmitters are, currently:

Table 1. Transponder Frequencies Mount Dandenong, Vic, Aus.

Broadcaster	Frequency
Seven	177.500 Mhz
SBS	184.500 Mhz
Nine	191.625 Mhz
Ten	219.500 Mhz
ABC	226.500 Mhz
Channel 31	557.625 Mhz

The digital TV Scan utilities (like dvbv5-scan) have use a set of compiled-in defaults for various countries and regions. Those are currently provided as a separate package, called dtv-scan-tables. It's git tree is located at LinuxTV.org:

<https://git.linuxtv.org/dtv-scan-tables.git/>

If none of the tables there suit, you can specify a data file on the command line which contains the transponder frequencies. Here is a sample file for the above channel transponders, in the old "channel" format:

```
# Data file for DVB scan program
#
# C Frequency SymbolRate FEC QAM
# S Frequency Polarisation SymbolRate FEC
# T Frequency Bandwidth FEC FEC2 QAM Mode Guard Hier
T 177500000 7MHz AUTO AUTO QAM64 8k 1/16 NONE
T 184500000 7MHz AUTO AUTO QAM64 8k 1/8 NONE
T 191625000 7MHz AUTO AUTO QAM64 8k 1/16 NONE
T 219500000 7MHz AUTO AUTO QAM64 8k 1/16 NONE
T 226500000 7MHz AUTO AUTO QAM64 8k 1/16 NONE
T 557625000 7MHz AUTO AUTO QPSK 8k 1/16 NONE
```

Nowadays, we prefer to use a newer format, with is more verbose and easier to understand. With the new format, the "Seven" channel transponder's data is represented by:

```
[Seven]
DELIVERY_SYSTEM = DVBT
FREQUENCY = 177500000
BANDWIDTH_HZ = 7000000
CODE_RATE_HP = AUTO
CODE_RATE_LP = AUTO
MODULATION = QAM/64
TRANSMISSION_MODE = 8K
GUARD_INTERVAL = 1/16
HIERARCHY = NONE
INVERSION = AUTO
```

For an updated version of the complete table, please see:

<https://git.linuxtv.org/dtv-scan-tables.git/tree/dvb-t/au-Melbourne>

When the Digital TV scanning utility runs, it will output a file containing the information for all the audio and video programs that exists into each channel's transponders which the card's frontend can lock onto. (i.e. any whose signal is strong enough at your antenna).

Here's the output of the dvbv5 tools from a channel scan took from Melburne:

[ABC HDTV]

```
SERVICE_ID = 560
VIDEO_PID = 2307
AUDIO_PID = 0
DELIVERY_SYSTEM = DVBT
FREQUENCY = 226500000
INVERSION = OFF
BANDWIDTH_HZ = 7000000
CODE_RATE_HP = 3/4
CODE_RATE_LP = 3/4
MODULATION = QAM/64
TRANSMISSION_MODE = 8K
GUARD_INTERVAL = 1/16
HIERARCHY = NONE
```

[ABC TV Melbourne]

```
SERVICE_ID = 561
VIDEO_PID = 512
AUDIO_PID = 650
DELIVERY_SYSTEM = DVBT
FREQUENCY = 226500000
INVERSION = OFF
BANDWIDTH_HZ = 7000000
CODE_RATE_HP = 3/4
CODE_RATE_LP = 3/4
MODULATION = QAM/64
TRANSMISSION_MODE = 8K
GUARD_INTERVAL = 1/16
HIERARCHY = NONE
```

[ABC TV 2]

```
SERVICE_ID = 562
VIDEO_PID = 512
AUDIO_PID = 650
DELIVERY_SYSTEM = DVBT
FREQUENCY = 226500000
INVERSION = OFF
BANDWIDTH_HZ = 7000000
CODE_RATE_HP = 3/4
CODE_RATE_LP = 3/4
MODULATION = QAM/64
TRANSMISSION_MODE = 8K
GUARD_INTERVAL = 1/16
HIERARCHY = NONE
```

[ABC TV 3]

```
SERVICE_ID = 563
VIDEO_PID = 512
AUDIO_PID = 650
DELIVERY_SYSTEM = DVBT
FREQUENCY = 226500000
INVERSION = OFF
BANDWIDTH_HZ = 7000000
CODE_RATE_HP = 3/4
CODE_RATE_LP = 3/4
MODULATION = QAM/64
TRANSMISSION_MODE = 8K
```

```
GUARD_INTERVAL = 1/16
HIERARCHY = NONE
```

```
[ABC TV 4]
```

```
SERVICE_ID = 564
VIDEO_PID = 512
AUDIO_PID = 650
DELIVERY_SYSTEM = DVBT
FREQUENCY = 226500000
INVERSION = OFF
BANDWIDTH_HZ = 7000000
CODE_RATE_HP = 3/4
CODE_RATE_LP = 3/4
MODULATION = QAM/64
TRANSMISSION_MODE = 8K
GUARD_INTERVAL = 1/16
HIERARCHY = NONE
```

```
[ABC DiG Radio]
```

```
SERVICE_ID = 566
VIDEO_PID = 0
AUDIO_PID = 2311
DELIVERY_SYSTEM = DVBT
FREQUENCY = 226500000
INVERSION = OFF
BANDWIDTH_HZ = 7000000
CODE_RATE_HP = 3/4
CODE_RATE_LP = 3/4
MODULATION = QAM/64
TRANSMISSION_MODE = 8K
GUARD_INTERVAL = 1/16
HIERARCHY = NONE
```

```
[TEN Digital]
```

```
SERVICE_ID = 1585
VIDEO_PID = 512
AUDIO_PID = 650
DELIVERY_SYSTEM = DVBT
FREQUENCY = 219500000
INVERSION = OFF
BANDWIDTH_HZ = 7000000
CODE_RATE_HP = 3/4
CODE_RATE_LP = 1/2
MODULATION = QAM/64
TRANSMISSION_MODE = 8K
GUARD_INTERVAL = 1/16
HIERARCHY = NONE
```

```
[TEN Digital 1]
```

```
SERVICE_ID = 1586
VIDEO_PID = 512
AUDIO_PID = 650
DELIVERY_SYSTEM = DVBT
FREQUENCY = 219500000
INVERSION = OFF
BANDWIDTH_HZ = 7000000
CODE_RATE_HP = 3/4
```

```
CODE_RATE_LP = 1/2
MODULATION = QAM/64
TRANSMISSION_MODE = 8K
GUARD_INTERVAL = 1/16
HIERARCHY = NONE
```

[TEN Digital 2]

```
SERVICE_ID = 1587
VIDEO_PID = 512
AUDIO_PID = 650
DELIVERY_SYSTEM = DVBT
FREQUENCY = 219500000
INVERSION = OFF
BANDWIDTH_HZ = 7000000
CODE_RATE_HP = 3/4
CODE_RATE_LP = 1/2
MODULATION = QAM/64
TRANSMISSION_MODE = 8K
GUARD_INTERVAL = 1/16
HIERARCHY = NONE
```

[TEN Digital 3]

```
SERVICE_ID = 1588
VIDEO_PID = 512
AUDIO_PID = 650
DELIVERY_SYSTEM = DVBT
FREQUENCY = 219500000
INVERSION = OFF
BANDWIDTH_HZ = 7000000
CODE_RATE_HP = 3/4
CODE_RATE_LP = 1/2
MODULATION = QAM/64
TRANSMISSION_MODE = 8K
GUARD_INTERVAL = 1/16
HIERARCHY = NONE
```

[TEN Digital]

```
SERVICE_ID = 1589
VIDEO_PID = 512
AUDIO_PID = 650
DELIVERY_SYSTEM = DVBT
FREQUENCY = 219500000
INVERSION = OFF
BANDWIDTH_HZ = 7000000
CODE_RATE_HP = 3/4
CODE_RATE_LP = 1/2
MODULATION = QAM/64
TRANSMISSION_MODE = 8K
GUARD_INTERVAL = 1/16
HIERARCHY = NONE
```

[TEN Digital 4]

```
SERVICE_ID = 1590
VIDEO_PID = 512
AUDIO_PID = 650
DELIVERY_SYSTEM = DVBT
FREQUENCY = 219500000
```



```
INVERSION = OFF
BANDWIDTH_HZ = 7000000
CODE_RATE_HP = 3/4
CODE_RATE_LP = 1/2
MODULATION = QAM/64
TRANSMISSION_MODE = 8K
GUARD_INTERVAL = 1/16
HIERARCHY = NONE
```

[TEN Digital]

```
SERVICE_ID = 1591
VIDEO_PID = 512
AUDIO_PID = 650
DELIVERY_SYSTEM = DVBT
FREQUENCY = 219500000
INVERSION = OFF
BANDWIDTH_HZ = 7000000
CODE_RATE_HP = 3/4
CODE_RATE_LP = 1/2
MODULATION = QAM/64
TRANSMISSION_MODE = 8K
GUARD_INTERVAL = 1/16
HIERARCHY = NONE
```

[TEN HD]

```
SERVICE_ID = 1592
VIDEO_PID = 514
AUDIO_PID = 0
DELIVERY_SYSTEM = DVBT
FREQUENCY = 219500000
INVERSION = OFF
BANDWIDTH_HZ = 7000000
CODE_RATE_HP = 3/4
CODE_RATE_LP = 1/2
MODULATION = QAM/64
TRANSMISSION_MODE = 8K
GUARD_INTERVAL = 1/16
HIERARCHY = NONE
```

[TEN Digital]

```
SERVICE_ID = 1593
VIDEO_PID = 512
AUDIO_PID = 650
DELIVERY_SYSTEM = DVBT
FREQUENCY = 219500000
INVERSION = OFF
BANDWIDTH_HZ = 7000000
CODE_RATE_HP = 3/4
CODE_RATE_LP = 1/2
MODULATION = QAM/64
TRANSMISSION_MODE = 8K
GUARD_INTERVAL = 1/16
HIERARCHY = NONE
```

[Nine Digital]

```
SERVICE_ID = 1072
VIDEO_PID = 513
```

```
AUDIO_PID = 660
DELIVERY_SYSTEM = DVBT
FREQUENCY = 191625000
INVERSION = OFF
BANDWIDTH_HZ = 7000000
CODE_RATE_HP = 3/4
CODE_RATE_LP = 1/2
MODULATION = QAM/64
TRANSMISSION_MODE = 8K
GUARD_INTERVAL = 1/16
HIERARCHY = NONE
```

[Nine Digital HD]

```
SERVICE_ID = 1073
VIDEO_PID = 512
AUDIO_PID = 0
DELIVERY_SYSTEM = DVBT
FREQUENCY = 191625000
INVERSION = OFF
BANDWIDTH_HZ = 7000000
CODE_RATE_HP = 3/4
CODE_RATE_LP = 1/2
MODULATION = QAM/64
TRANSMISSION_MODE = 8K
GUARD_INTERVAL = 1/16
HIERARCHY = NONE
```

[Nine Guide]

```
SERVICE_ID = 1074
VIDEO_PID = 514
AUDIO_PID = 670
DELIVERY_SYSTEM = DVBT
FREQUENCY = 191625000
INVERSION = OFF
BANDWIDTH_HZ = 7000000
CODE_RATE_HP = 3/4
CODE_RATE_LP = 1/2
MODULATION = QAM/64
TRANSMISSION_MODE = 8K
GUARD_INTERVAL = 1/16
HIERARCHY = NONE
```

[7 Digital]

```
SERVICE_ID = 1328
VIDEO_PID = 769
AUDIO_PID = 770
DELIVERY_SYSTEM = DVBT
FREQUENCY = 177500000
INVERSION = OFF
BANDWIDTH_HZ = 7000000
CODE_RATE_HP = 2/3
CODE_RATE_LP = 2/3
MODULATION = QAM/64
TRANSMISSION_MODE = 8K
GUARD_INTERVAL = 1/8
HIERARCHY = NONE
```

[7 Digital 1]

```
SERVICE_ID = 1329
VIDEO_PID = 769
AUDIO_PID = 770
DELIVERY_SYSTEM = DVBT
FREQUENCY = 177500000
INVERSION = OFF
BANDWIDTH_HZ = 7000000
CODE_RATE_HP = 2/3
CODE_RATE_LP = 2/3
MODULATION = QAM/64
TRANSMISSION_MODE = 8K
GUARD_INTERVAL = 1/8
HIERARCHY = NONE
```

[7 Digital 2]

```
SERVICE_ID = 1330
VIDEO_PID = 769
AUDIO_PID = 770
DELIVERY_SYSTEM = DVBT
FREQUENCY = 177500000
INVERSION = OFF
BANDWIDTH_HZ = 7000000
CODE_RATE_HP = 2/3
CODE_RATE_LP = 2/3
MODULATION = QAM/64
TRANSMISSION_MODE = 8K
GUARD_INTERVAL = 1/8
HIERARCHY = NONE
```

[7 Digital 3]

```
SERVICE_ID = 1331
VIDEO_PID = 769
AUDIO_PID = 770
DELIVERY_SYSTEM = DVBT
FREQUENCY = 177500000
INVERSION = OFF
BANDWIDTH_HZ = 7000000
CODE_RATE_HP = 2/3
CODE_RATE_LP = 2/3
MODULATION = QAM/64
TRANSMISSION_MODE = 8K
GUARD_INTERVAL = 1/8
HIERARCHY = NONE
```

[7 HD Digital]

```
SERVICE_ID = 1332
VIDEO_PID = 833
AUDIO_PID = 834
DELIVERY_SYSTEM = DVBT
FREQUENCY = 177500000
INVERSION = OFF
BANDWIDTH_HZ = 7000000
CODE_RATE_HP = 2/3
CODE_RATE_LP = 2/3
MODULATION = QAM/64
TRANSMISSION_MODE = 8K
```

```
GUARD_INTERVAL = 1/8
HIERARCHY = NONE
```

[7 Program Guide]

```
SERVICE_ID = 1334
VIDEO_PID = 865
AUDIO_PID = 866
DELIVERY_SYSTEM = DVBT
FREQUENCY = 177500000
INVERSION = OFF
BANDWIDTH_HZ = 7000000
CODE_RATE_HP = 2/3
CODE_RATE_LP = 2/3
MODULATION = QAM/64
TRANSMISSION_MODE = 8K
GUARD_INTERVAL = 1/8
HIERARCHY = NONE
```

[SBS HD]

```
SERVICE_ID = 784
VIDEO_PID = 102
AUDIO_PID = 103
DELIVERY_SYSTEM = DVBT
FREQUENCY = 536500000
INVERSION = OFF
BANDWIDTH_HZ = 7000000
CODE_RATE_HP = 2/3
CODE_RATE_LP = 2/3
MODULATION = QAM/64
TRANSMISSION_MODE = 8K
GUARD_INTERVAL = 1/8
HIERARCHY = NONE
```

[SBS DIGITAL 1]

```
SERVICE_ID = 785
VIDEO_PID = 161
AUDIO_PID = 81
DELIVERY_SYSTEM = DVBT
FREQUENCY = 536500000
INVERSION = OFF
BANDWIDTH_HZ = 7000000
CODE_RATE_HP = 2/3
CODE_RATE_LP = 2/3
MODULATION = QAM/64
TRANSMISSION_MODE = 8K
GUARD_INTERVAL = 1/8
HIERARCHY = NONE
```

[SBS DIGITAL 2]

```
SERVICE_ID = 786
VIDEO_PID = 162
AUDIO_PID = 83
DELIVERY_SYSTEM = DVBT
FREQUENCY = 536500000
INVERSION = OFF
BANDWIDTH_HZ = 7000000
CODE_RATE_HP = 2/3
```

```
CODE_RATE_LP = 2/3
MODULATION = QAM/64
TRANSMISSION_MODE = 8K
GUARD_INTERVAL = 1/8
HIERARCHY = NONE
```

[SBS EPG]

```
SERVICE_ID = 787
VIDEO_PID = 163
AUDIO_PID = 85
DELIVERY_SYSTEM = DVBT
FREQUENCY = 536500000
INVERSION = OFF
BANDWIDTH_HZ = 7000000
CODE_RATE_HP = 2/3
CODE_RATE_LP = 2/3
MODULATION = QAM/64
TRANSMISSION_MODE = 8K
GUARD_INTERVAL = 1/8
HIERARCHY = NONE
```

[SBS RADIO 1]

```
SERVICE_ID = 798
VIDEO_PID = 0
AUDIO_PID = 201
DELIVERY_SYSTEM = DVBT
FREQUENCY = 536500000
INVERSION = OFF
BANDWIDTH_HZ = 7000000
CODE_RATE_HP = 2/3
CODE_RATE_LP = 2/3
MODULATION = QAM/64
TRANSMISSION_MODE = 8K
GUARD_INTERVAL = 1/8
HIERARCHY = NONE
```

[SBS RADIO 2]

```
SERVICE_ID = 799
VIDEO_PID = 0
AUDIO_PID = 202
DELIVERY_SYSTEM = DVBT
FREQUENCY = 536500000
INVERSION = OFF
BANDWIDTH_HZ = 7000000
CODE_RATE_HP = 2/3
CODE_RATE_LP = 2/3
MODULATION = QAM/64
TRANSMISSION_MODE = 8K
GUARD_INTERVAL = 1/8
HIERARCHY = NONE
```

1.1.4.2 Digital TV Conditional Access Interface

Note: This documentation is outdated.

This document describes the usage of the high level CI API as in accordance to the Linux DVB API. This is not a documentation for the, existing low level CI API.

Note: For the Twinhan/Twinhan clones, the `dst_ca` module handles the CI hardware handling. This module is loaded automatically if a CI (Common Interface, that holds the CAM (Conditional Access Module) is detected.

`ca_zap`

A userspace application, like `ca_zap` is required to handle encrypted MPEG-TS streams.

The `ca_zap` userland application is in charge of sending the descrambling related information to the Conditional Access Module (CAM).

This application requires the following to function properly as of now.

a) Tune to a valid channel, with `szap`.

eg: `$ szap -c channels.conf -r "TMC" -x`

b) a `channels.conf` containing a valid PMT PID

eg: `TMC:11996:h:0:27500:278:512:650:321`

here 278 is a valid PMT PID. the rest of the values are the same ones that `szap` uses.

c) after running a `szap`, you have to run `ca_zap`, for the descrambler to function,

eg: `$ ca_zap channels.conf "TMC"`

d) Hopefully enjoy your favourite subscribed channel as you do with a FTA card.

Note: Currently `ca_zap`, and `dst_test`, both are meant for demonstration purposes only, they can become full fledged applications if necessary.

Cards that fall in this category

At present the cards that fall in this category are the Twinhan and its clones, these cards are available as VVMER, Tomato, Hercules, Orange and so on.

CI modules that are supported

The CI module support is largely dependent upon the firmware on the cards. Some cards do support almost all of the available CI modules. There is nothing much that can be done in order to make additional CI modules working with these cards.

Modules that have been tested by this driver at present are

- (1) Irdeto 1 and 2 from SCM
- (2) Viaccess from SCM
- (3) Dragoncam

1.1.4.3 FAQ

Note:

1. With Digital TV, a single physical channel may have different contents inside it. The specs call each one as a *service*. This is what a TV user would call “channel”. So, in order to avoid confusion, we’re calling *transponders* as the physical channel on this FAQ, and *services* for the logical channel.
2. The LinuxTV community maintains some Wiki pages with contain a lot of information related to the media subsystem. If you don’t find an answer for your needs here, it is likely that you’ll be able to get something useful there. It is hosted at:

<https://www.linuxtv.org/wiki/>

Some very frequently asked questions about Linux Digital TV support

1. The signal seems to die a few seconds after tuning.

It’s not a bug, it’s a feature. Because the frontends have significant power requirements (and hence get very hot), they are powered down if they are unused (i.e. if the frontend device is closed). The `dvb-core` module parameter `dvb_shutdown_timeout` allow you to change the timeout (default 5 seconds). Setting the timeout to 0 disables the timeout feature.

2. How can I watch TV?

Together with the Linux Kernel, the Digital TV developers support some simple utilities which are mainly intended for testing and to demonstrate how the DVB API works. This is called DVB v5 tools and are grouped together with the `v4l-utils` git repository:

<https://git.linuxtv.org/v4l-utils.git/>

You can find more information at the LinuxTV wiki:

https://www.linuxtv.org/wiki/index.php/DVBv5_Tools

The first step is to get a list of services that are transmitted.

This is done by using several existing tools. You can use for example the `dvbv5-scan` tool. You can find more information about it at:

<https://www.linuxtv.org/wiki/index.php/Dvbv5-scan>

There are some other applications like `w_scan`¹ that do a blind scan, trying hard to find all possible channels, but those consumes a large amount of time to run.

Also, some applications like `kaffeine` have their own code to scan for services. So, you don't need to use an external application to obtain such list.

Most of such tools need a file containing a list of channel transponders available on your area. So, LinuxTV developers maintain tables of Digital TV channel transponders, receiving patches from the community to keep them updated.

This list is hosted at:

<https://git.linuxtv.org/dtv-scan-tables.git>

And packaged on several distributions.

`Kaffeine` has some blind scan support for some terrestrial standards. It also relies on DTV scan tables, although it contains a copy of it internally (and, if requested by the user, it will download newer versions of it).

If you are lucky you can just use one of the supplied channel transponders. If not, you may need to seek for such info at the Internet and create a new file. There are several sites with contains physical channel lists. For cable and satellite, usually knowing how to tune into a single channel is enough for the scanning tool to identify the other channels. On some places, this could also work for terrestrial transmissions.

Once you have a transponders list, you need to generate a services list with a tool like `dvbv5-scan`.

Almost all modern Digital TV cards don't have built-in hardware MPEG-decoders. So, it is up to the application to get a MPEG-TS stream provided by the board, split it into audio, video and other data and decode.

3. Which Digital TV applications exist?

Several media player applications are capable of tuning into digital TV channels, including `Kaffeine`, `Vlc`, `mplayer` and `MythTV`.

`Kaffeine` aims to be very user-friendly, and it is maintained by one of the Kernel driver developers.

A comprehensive list of those and other apps can be found at:

https://www.linuxtv.org/wiki/index.php/TV_Related_Software

Some of the most popular ones are linked below:

<https://kde.org/applications/multimedia/org.kde.kaffeine> KDE media player, focused on Digital TV support

https://www.linuxtv.org/vdrwiki/index.php/Main_Page Klaus Schmidinger's Video Disk Recorder

<https://linuxtv.org/downloads> and <https://git.linuxtv.org/> Digital TV and other media-related applications and Kernel drivers. The `v4l-utils` package there contains several swiss knife tools for using with Digital TV.

¹ https://www.linuxtv.org/wiki/index.php/W_scan

<http://sourceforge.net/projects/dvbtools/> Dave Chapman's dvbtools package, including dvbstream and dvbtune

<http://www.dbox2.info/> LinuxDVB on the dBox2

<http://www.tuxbox.org/> the TuxBox CVS many interesting DVB applications and the dBox2 DVB source

<http://www.nenie.org/misc/mpsys/> MPSYS: a MPEG2 system library and tools

<https://www.videolan.org/vlc/index.pt.html> Vlc

<http://mplayerhq.hu/> MPlayer

<http://xine.sourceforge.net/> and <http://xinehq.de/> Xine

<http://www.mythtv.org/> MythTV - analog TV and digital TV PVR

<http://dvbsnoop.sourceforge.net/> DVB sniffer program to monitor, analyze, debug, dump or view dvb/mpeg/dsm-cc/mhp stream information (TS, PES, SECTION)

4. Can't get a signal tuned correctly

That could be due to a lot of problems. On my personal experience, usually TV cards need stronger signals than TV sets, and are more sensitive to noise. So, perhaps you just need a better antenna or cabling. Yet, it could also be some hardware or driver issue.

For example, if you are using a Technotrend/Hauppauge DVB-C card *without* analog module, you might have to use module parameter `adac=-1` (`dvb-ttpci.o`).

Please see the FAQ page at linuxtv.org, as it could contain some valuable information:

https://www.linuxtv.org/wiki/index.php/FAQ_%26_Troubleshooting

If that doesn't work, check at the linux-media ML archives, to see if someone else had a similar problem with your hardware and/or digital TV service provider:

<https://lore.kernel.org/linux-media/>

If none of this works, you can try sending an e-mail to the linux-media ML and see if someone else could shed some light. The e-mail is `linux-media AT vger.kernel.org`.

5. The `dvb_net` device doesn't give me any packets at all

Run `tcpdump` on the `dvb0_0` interface. This sets the interface into promiscuous mode so it accepts any packets from the PID you have configured with the `dvbnet` utility. Check if there are any packets with the IP addr and MAC addr you have configured with `ifconfig` or with `ip addr`.

If `tcpdump` doesn't give you any output, check the statistics which `ifconfig` or `netstat -ni` outputs. (Note: If the MAC address is wrong, `dvb_net` won't get any input; thus you have to run `tcpdump` before checking the statistics.) If there are no packets at all then maybe the PID is wrong. If there are error packets, then either the PID is wrong or the stream does not conform to the MPE standard (EN 301 192, <http://www.etsi.org/>). You can use e.g. `dvbsnoop` for debugging.

6. The `dvb_net` device doesn't give me any multicast packets

Check your routes if they include the multicast address range. Additionally make sure that “source validation by reversed path lookup” is disabled:

```
$ "echo 0 > /proc/sys/net/ipv4/conf/dvb0/rp_filter"
```

7. What are all those modules that need to be loaded?

In order to make it more flexible and support different hardware combinations, the media subsystem is written on a modular way.

So, besides the Digital TV hardware module for the main chipset, it also needs to load a frontend driver, plus the Digital TV core. If the board also has remote controller, it will also need the remote controller core and the remote controller tables. The same happens if the board has support for analog TV: the core support for video4linux need to be loaded.

The actual module names are Linux-kernel version specific, as, from time to time, things change, in order to make the media support more flexible.

1.1.4.4 References

The main development site and GIT repository for Digital TV drivers is <https://linuxtv.org>.

The DVB mailing list linux-dvb is hosted at vger. Please see <http://vger.kernel.org/vger-lists.html#linux-media> for details.

There are also some other old lists hosted at: <https://linuxtv.org/lists.php>. If you're interested on that for historic reasons, please check the archive at <https://linuxtv.org/pipermail/linux-dvb/>.

The media subsystem Wiki is hosted at <https://linuxtv.org/wiki/>. There, you'll find lots of information, from both development and usage of media boards. Please check it before asking newbie questions on the mailing list or IRC channels.

The API documentation is documented at the Kernel tree. You can find it in both html and pdf formats, together with other useful documentation at:

- <https://linuxtv.org/docs.php>.

You may also find useful material at <https://linuxtv.org/downloads/>.

In order to get the needed firmware for some drivers to work, there's a script at the kernel tree, at `scripts/get_dvb_firmware`.

1.1.5 Cards List

The media subsystem provide support for lots of PCI and USB drivers, plus platform-specific drivers. It also contains several ancillary I²C drivers.

The platform-specific drivers are usually present on embedded systems, or are supported by the main board. Usually, setting them is done via OpenFirmware or ACPI.

The PCI and USB drivers, however, are independent of the system's board, and may be added/removed by the user.

You may also take a look at https://linuxtv.org/wiki/index.php/Hardware_Device_Information for more details about supported cards.

1.1.5.1 USB drivers

The USB boards are identified by an identification called USB ID.

The `lsusb` command allows identifying the USB IDs:

```
$ lsusb
...
Bus 001 Device 015: ID 046d:082d Logitech, Inc. HD Pro Webcam C920
Bus 001 Device 074: ID 2040:b131 Hauppauge
Bus 001 Device 075: ID 2013:024f PCTV Systems nanoStick T2 290e
...
```

Newer camera devices use a standard way to expose themselves as such, via USB Video Class. Those cameras are automatically supported by the `uvc-driver`.

Older cameras and TV USB devices uses USB Vendor Classes: each vendor defines its own way to access the device. This section contains card lists for such vendor-class devices.

While this is not as common as on PCI, sometimes the same USB ID is used by different products. So, several media drivers allow passing a `card=` parameter, in order to setup a card number that would match the correct settings for an specific product type.

The current supported USB cards (not including staging drivers) are listed below¹.

Driver	Name
<code>airspy</code>	AirSpy
<code>au0828</code>	Auvitek AU0828
<code>b2c2-flexcop-usb</code>	Technisat/B2C2 Air/Sky/Cable2PC USB
<code>cpia2</code>	CPiA2 Video For Linux
<code>cx231xx</code>	Conexant cx231xx USB video capture
<code>dvb-as102</code>	Abilis AS102 DVB receiver
<code>dvb-ttusb-budget</code>	Technotrend/Hauppauge Nova - USB devices
<code>dvb-usb-a800</code>	AVerMedia AverTV DVB-T USB 2.0 (A800)
<code>dvb-usb-af9005</code>	Afatech AF9005 DVB-T USB1.1
<code>dvb-usb-af9015</code>	Afatech AF9015 DVB-T USB2.0
<code>dvb-usb-af9035</code>	Afatech AF9035 DVB-T USB2.0
<code>dvb-usb-anysee</code>	Anysee DVB-T/C USB2.0
<code>dvb-usb-au6610</code>	Alcor Micro AU6610 USB2.0
<code>dvb-usb-az6007</code>	AzureWave 6007 and clones DVB-T/C USB2.0
<code>dvb-usb-az6027</code>	Azurewave DVB-S/S2 USB2.0 AZ6027
<code>dvb-usb-ce6230</code>	Intel CE6230 DVB-T USB2.0
<code>dvb-usb-cinergyT2</code>	Terratec CinergyT2/qanu USB 2.0 DVB-T
<code>dvb-usb-cxusb</code>	Conexant USB2.0 hybrid
<code>dvb-usb-dib0700</code>	DiBcom DiB0700
<code>dvb-usb-dibusb-common</code>	DiBcom DiB3000M-B
<code>dvb-usb-dibusb-mc</code>	DiBcom DiB3000M-C/P
<code>dvb-usb-digitv</code>	Nebula Electronics uDigiTV DVB-T USB2.0
<code>dvb-usb-dtt200u</code>	WideView WT-200U and WT-220U (pen) DVB-T
<code>dvb-usb-dtv5100</code>	AME DTV-5100 USB2.0 DVB-T

Continued on next page

¹ some of the drivers have sub-drivers, not shown at this table. In particular, `gspca` driver has lots of sub-drivers, for cameras not supported by the USB Video Class (UVC) driver, as shown at [gspca card list](#).

Table 1 - continued from previous page

Driver	Name
dvb-usb-dvbsky	DVBSky USB
dvb-usb-dw2102	DvbWorld & TeVii DVB-S/S2 USB2.0
dvb-usb-ec168	E3C EC168 DVB-T USB2.0
dvb-usb-gl861	Genesys Logic GL861 USB2.0
dvb-usb-gp8psk	GENPIX 8PSK->USB module
dvb-usb-lmedm04	LME DM04/QQBOX DVB-S USB2.0
dvb-usb-m920x	Uli m920x DVB-T USB2.0
dvb-usb-nova-t-usb2	Hauppauge WinTV-NOVA-T usb2 DVB-T USB2.0
dvb-usb-opera	Opera1 DVB-S USB2.0 receiver
dvb-usb-pctv452e	Pinnacle PCTV HDTV Pro USB device/TT Connect S2-3600
dvb-usb-rtl28xxu	Realtek RTL28xxU DVB USB
dvb-usb-technisat-usb2	Technisat DVB-S/S2 USB2.0
dvb-usb-ttusb2	Pinnacle 400e DVB-S USB2.0
dvb-usb-umt-010	HanfTek UMT-010 DVB-T USB2.0
dvb_usb_v2	Support for various USB DVB devices v2
dvb-usb-vp702x	TwinhanDTV StarBox and clones DVB-S USB2.0
dvb-usb-vp7045	TwinhanDTV Alpha/MagicBoxII, DNTV tinyUSB2, Beetle USB2.0
em28xx	Empia EM28xx USB devices
go7007	WIS GO7007 MPEG encoder
gspca	Drivers for several USB Cameras
hackrf	HackRF
hdpvr	Hauppauge HD PVR
msi2500	Mirics MSi2500
mxl111sf-tuner	MxL111SF DTV USB2.0
pvrusb2	Hauppauge WinTV-PVR USB2
pwc	USB Philips Cameras
s2250	Sensoray 2250/2251
s2255drv	USB Sensoray 2255 video capture device
smsusb	Siano SMS1xxx based MDTV receiver
stkwebcam	USB Syntek DC1125 Camera
tm6000-alsa	TV Master TM5600/6000/6010 audio
tm6000-dvb	DVB Support for tm6000 based TV cards
tm6000	TV Master TM5600/6000/6010 driver
ttusb_dec	Technotrend/Hauppauge USB DEC devices
usbtv	USBTV007 video capture
uvcvideo	USB Video Class (UVC)
zd1301	ZyDAS ZD1301
zr364xx	USB ZR364XX Camera

AU0828 cards list

Card number	Card name	USB IDs
0	Unknown board	
1	Hauppauge HVR950Q	2040:7200, 2040:7210, 2040:7217, 2040:721b, 2040:721e, 2040:721f, 2040:7280, 0fd9:0008, 2040:7260, 2040:7213, 2040:7270
2	Hauppauge HVR850	2040:7240
3	DViCO FusionHDTV USB	0fe9:d620
4	Hauppauge HVR950Q rev xxF8	2040:7201, 2040:7211, 2040:7281
5	Hauppauge Woodbury	05e1:0480, 2040:8200

cx231xx cards list

Card number	Card name	USB IDs
0	Unknown CX231xx video grabber	0572:5A3C
1	Conexant Hybrid TV - CARRAERA	0572:58A2
2	Conexant Hybrid TV - SHELBY	0572:58A1
3	Conexant Hybrid TV - RDE253S	0572:58A4
4	Conexant Hybrid TV - RDU253S	0572:58A5
5	Conexant VIDEO GRABBER	0572:58A6, 07ca:c039
6	Conexant Hybrid TV - rde 250	0572:589E
7	Conexant Hybrid TV - RDU 250	0572:58A0
8	Hauppauge EXETER	2040:b120, 2040:b140
9	Hauppauge USB Live 2	2040:c200
10	Pixelview PlayTV USB Hybrid	4000:4001
11	Pixelview Xcapture USB	1D19:6109, 4000:4001
12	Kworld UB430 USB Hybrid	1b80:e424
13	Iconbit Analog Stick U100 FM	1f4d:0237
14	Hauppauge WinTV USB2 FM (PAL)	2040:b110
15	Hauppauge WinTV USB2 FM (NTSC)	2040:b111
16	Elgato Video Capture V2	0fd9:0037
17	Geniatech OTG102	1f4d:0102
18	Kworld UB445 USB Hybrid	1b80:e421
19	Hauppauge WinTV 930C-HD (1113xx) / HVR-900H (111xxx) / PCTV QuatroStick 521e	2040:b130, 2040:b138, 2013:0259
20	Hauppauge WinTV 930C-HD (1114xx) / HVR-901H (1114xx) / PCTV QuatroStick 522e	2040:b131, 2040:b139, 2013:025e
21	Hauppauge WinTV-HVR-955Q (111401)	2040:b123, 2040:b124
22	Terratec Grabby	1f4d:0102
23	Evromedia USB Full Hybrid Full HD	1b80:d3b2
24	Astrometa T2hybrid	15f4:0135
25	The Imaging Source DFG/USB2pro	199e:8002
26	Hauppauge WinTV-HVR-935C	2040:b151
27	Hauppauge WinTV-HVR-975	2040:b150

EM28xx cards list

Card number	Card name	Empia Chip	USB IDs
0	Unknown EM2800 video grabber	em2800	eb1a:2800

Continued on next page

Table 2 - continued from previous page

Card number	Card name	Empia Chip	USB IDs
1	Unknown EM2750/28xx video grabber	em2820 or em2840	eb1a:2710, eb1a:2820, eb1a:2821, eb1a:2860, eb1a:2861, eb1a:2862, eb1a:2863, eb1a:2870, eb1a:2881, eb1a:2883, eb1a:2868, eb1a:2875
2	Terratec Cinergy 250 USB	em2820 or em2840	0ccd:0036
3	Pinnacle PCTV USB 2	em2820 or em2840	2304:0208
4	Hauppauge WinTV USB 2	em2820 or em2840	2040:4200, 2040:4201
5	MSI VOX USB 2.0	em2820 or em2840	
6	Terratec Cinergy 200 USB	em2800	
7	Leadtek Winfast USB II	em2800	0413:6023
8	Kworld USB2800	em2800	
9	Pinnacle Dazzle DVC 90/100/101/107 / Kaiser Baas Video to DVD maker / Kworld DVD Maker 2 / Plextor ConvertX PX-AV100U	em2820 or em2840	1b80:e302, 1b80:e304, 2304:0207, 2304:021a, 093b:a003
10	Hauppauge WinTV HVR 900	em2880	2040:6500
11	Terratec Hybrid XS	em2880	
12	Kworld PVR TV 2800 RF	em2820 or em2840	
13	Terratec Prodigy XS	em2880	
14	SIIG AVTuner-PVR / Pixelview Prolink PlayTV USB 2.0	em2820 or em2840	
15	V-Gear PocketTV	em2800	
16	Hauppauge WinTV HVR 950	em2883	2040:6513, 2040:6517, 2040:651b
17	Pinnacle PCTV HD Pro Stick	em2880	2304:0227

Continued on next page

Table 2 - continued from previous page

Card number	Card name	Empia Chip	USB IDs
18	Hauppauge WinTV HVR 900 (R2)	em2880	2040:6502
19	EM2860/SAA711X Reference Design	em2860	
20	AMD ATI TV Wonder HD 600	em2880	0438:b002
21	eMPIA Technology, Inc. GrabBeeX+ Video Encoder	em2800	eb1a:2801
22	EM2710/EM2750/EM2751 webcam grabber	em2750	eb1a:2750, eb1a:2751
23	Huaqi DLCW-130	em2750	
24	D-Link DUB-T210 TV Tuner	em2820 or em2840	2001:f112
25	Gadmei UTV310	em2820 or em2840	
26	Hercules Smart TV USB 2.0	em2820 or em2840	
27	Pinnacle PCTV USB 2 (Philips FM1216ME)	em2820 or em2840	
28	Leadtek Winfast USB II Deluxe	em2820 or em2840	
29	EM2860/TVP5150 Reference Design	em2860	eb1a:5051
30	Videology 20K14XUSB USB2.0	em2820 or em2840	
31	Usbgear VD204v9	em2821	
32	Supercomp USB 2.0 TV	em2821	
33	Elgato Video Capture	em2860	0fd9:0033
34	Terratec Cinergy A Hybrid XS	em2860	0ccd:004f
35	Typhoon DVD Maker	em2860	
36	NetGMBH Cam	em2860	
37	Gadmei UTV330	em2860	eb1a:50a6
38	Yakumo MovieMixer	em2861	
39	KWorld PVRTV 300U	em2861	eb1a:e300
40	Plextor ConvertX PX-TV100U	em2861	093b:a005
41	Kworld 350 U DVB-T	em2870	eb1a:e350
42	Kworld 355 U DVB-T	em2870	eb1a:e355, eb1a:e357, eb1a:e359
43	Terratec Cinergy T XS	em2870	
44	Terratec Cinergy T XS (MT2060)	em2870	0ccd:0043
45	Pinnacle PCTV DVB-T	em2870	
46	Compro, VideoMate U3	em2870	185b:2870
47	KWorld DVB-T 305U	em2880	eb1a:e305

Continued on next page

Table 2 - continued from previous page

Card number	Card name	Empia Chip	USB IDs
48	KWorld DVB-T 310U	em2880	
49	MSI DigiVox A/D	em2880	eb1a:e310
50	MSI DigiVox A/D II	em2880	eb1a:e320
51	Terratec Hybrid XS Secam	em2880	0ccd:004c
52	DNT DA2 Hybrid	em2881	
53	Pinnacle Hybrid Pro	em2881	
54	Kworld VS-DVB-T 323UR	em2882	eb1a:e323
55	Terratec Cinergy Hybrid T USB XS (em2882)	em2882	0ccd:005e, 0ccd:0042
56	Pinnacle Hybrid Pro (330e)	em2882	2304:0226
57	Kworld PlusTV HD Hybrid 330	em2883	eb1a:a316
58	Compro VideoMate ForYou/Stereo	em2820 or em2840	185b:2041
59	Pinnacle PCTV HD Mini	em2874	2304:023f
60	Hauppauge WinTV HVR 850	em2883	2040:651f
61	Pixelview PlayTV Box 4 USB 2.0	em2820 or em2840	
62	Gadmei TVR200	em2820 or em2840	
63	Kaiomy TVnPC U2	em2860	eb1a:e303
64	Easy Cap Capture DC-60	em2860	1b80:e309
65	IO-DATA GV-MVP/SZ	em2820 or em2840	04bb:0515
66	Empire dual TV	em2880	
67	Terratec Grabby	em2860	0ccd:0096, 0ccd:10AF
68	Terratec AV350	em2860	0ccd:0084
69	KWorld ATSC 315U HDTV TV Box	em2882	eb1a:a313
70	Evga inDtube	em2882	
71	Silvercrest Webcam 1.3mpix	em2820 or em2840	
72	Gadmei UTV330+	em2861	
73	Reddo DVB-C USB TV Box	em2870	
74	Actionmaster/LinXcel/Digitus VC211A	em2800	
75	Dikom DK300	em2882	
76	KWorld PlusTV 340U or UB435-Q (ATSC)	em2870	1b80:a340
77	EM2874 Leadership ISDBT	em2874	
78	PCTV nanoStick T2 290e	em28174	2013:024f

Continued on next page

Table 2 - continued from previous page

Card number	Card name	Empia Chip	USB IDs
79	Terratec Cinergy H5	em2884	eb1a:2885, 0ccd:10a2, 0ccd:10ad, 0ccd:10b6
80	PCTV DVB-S2 Stick (460e)	em28174	2013:024c
81	Hauppauge WinTV HVR 930C	em2884	2040:1605
82	Terratec Cinergy HTC Stick	em2884	0ccd:00b2
83	Honestech Vidbox NW03	em2860	eb1a:5006
84	MaxMedia UB425-TC	em2874	1b80:e425
85	PCTV QuatroStick (510e)	em2884	2304:0242
86	PCTV QuatroStick nano (520e)	em2884	2013:0251
87	Terratec Cinergy HTC USB XS	em2884	0ccd:008e, 0ccd:00ac
88	C3 Tech Digital Duo HDTV/SDTV USB	em2884	1b80:e755
89	Delock 61959	em2874	1b80:e1cc
90	KWorld USB ATSC TV Stick UB435-Q V2	em2874	1b80:e346
91	SpeedLink Vicious And Devine Laplace webcam	em2765	1ae7:9003, 1ae7:9004
92	PCTV DVB-S2 Stick (461e)	em28178	2013:0258
93	KWorld USB ATSC TV Stick UB435-Q V3	em2874	1b80:e34c
94	PCTV tripleStick (292e)	em28178	2013:025f, 2013:0264, 2040:0264, 2040:8264, 2040:8268
95	Leadtek VC100	em2861	0413:6f07
96	Terratec Cinergy T2 Stick HD	em28178	eb1a:8179
97	Elgato EyeTV Hybrid 2008 INT	em2884	0fd9:0018
98	PLEX PX-BCUD	em28178	3275:0085
99	Hauppauge WinTV-dualHD DVB	em28174	2040:0265, 2040:8265
100	Hauppauge WinTV-dualHD 01595 ATSC/QAM	em28174	2040:026d, 2040:826d
101	Terratec Cinergy H6 rev. 2	em2884	0ccd:10b2
102	:ZOLID HYBRID TV STICK	em2882	
103	Magix USB Videowandler-2	em2861	1b80:e349
104	PCTV DVB-S2 Stick (461e v2)	em28178	2013:0461, 2013:0259
105	MyGica iGrabber	em2860	1f4d:1abe

TM6000 cards list

Card number	Card name	USB IDs
0	Unknown tm6000 video grabber	
1	Generic tm5600 board	6000:0001
2	Generic tm6000 board	
3	Generic tm6010 board	6000:0002
4	10Moons UT 821	
5	10Moons UT 330	
6	ADSTECH Dual TV USB	06e1:f332
7	Freecom Hybrid Stick / Moka DVB-T Receiver Dual	14aa:0620
8	ADSTECH Mini Dual TV USB	06e1:b339
9	Hauppauge WinTV HVR-900H / WinTV USB2-Stick	2040:6600, 2040:6601, 2040:6610, 2040:6611
10	Beholder Wander DVB-T/TV/FM USB2.0	6000:dec0
11	Beholder Voyager TV/FM USB2.0	6000:dec1
12	Terratec Cinergy Hybrid XE / Cinergy Hybrid-Stick	0ccd:0086, 0ccd:00A5
13	Twinhan TU501(704D1)	13d3:3240, 13d3:3241, 13d3:3243, 13d3:3264
14	Beholder Wander Lite DVB-T/TV/FM USB2.0	6000:dec2
15	Beholder Voyager Lite TV/FM USB2.0	6000:dec3
16	Terratec Grabster AV 150/250 MX	0ccd:0079

Siano cards list

Card name	USB IDs
Hauppauge Catamount	2040:1700
Hauppauge Okemo-A	2040:1800
Hauppauge Okemo-B	2040:1801
Hauppauge WinTV MiniCard	2040:2000, 2040:200a, 2040:2010, 2040:2011, 2040:2019
Hauppauge WinTV MiniCard Rev 2	2040:2009
Hauppauge WinTV MiniStick	2040:5500, 2040:5510, 2040:5520, 2040:5530, 2040:5580, 2040:5590, 2040:b900, 2040:b910, 2040:b980, 2040:b990, 2040:c000, 2040:c010, 2040:c080, 2040:c090, 2040:c0a0, 2040:f5a0
Hauppauge microStick 77e	2013:0257
ONDA Data Card Digital Receiver	19D2:0078
Siano Denver (ATSC-M/H) Digital Receiver	187f:0800
Siano Denver (TDMB) Digital Receiver	187f:0700
Siano Ming Digital Receiver	187f:0310
Siano Nice Digital Receiver	187f:0202, 187f:0202
Siano Nova A Digital Receiver	187f:0200
Siano Nova B Digital Receiver	187f:0201
Siano Pele Digital Receiver	187f:0500
Siano Rio Digital Receiver	187f:0600, 3275:0080
Siano Stellar Digital Receiver	187f:0100
Siano Stellar Digital Receiver ROM	187f:0010
Siano Vega Digital Receiver	187f:0300
Siano Venice Digital Receiver	187f:0301, 187f:0301, 187f:0302
ZTE Data Card Digital Receiver	19D2:0086

The gspca cards list

The modules for the gspca webcam drivers are:

- `gspca_main`: main driver
- `gspca_driver`: subdriver module with *driver* as follows

<i>driver</i>	vend:prod	Device
<code>spca501</code>	0000:0000	MystFromOri Unknown Camera
<code>spca508</code>	0130:0130	Clone Digital Webcam 11043
<code>se401</code>	03e8:0004	Endpoints/AoxSE401
<code>zc3xx</code>	03f0:1b07	HP Premium Starter Cam
<code>m5602</code>	0402:5602	ALi Video Camera Controller
<code>spca501</code>	040a:0002	Kodak DVC-325
<code>spca500</code>	040a:0300	Kodak EZ200
<code>zc3xx</code>	041e:041e	Creative WebCam Live!
<code>ov519</code>	041e:4003	Video Blaster WebCam Go Plus
<code>stv0680</code>	041e:4007	Go Mini
<code>spca500</code>	041e:400a	Creative PC-CAM 300
<code>sunplus</code>	041e:400b	Creative PC-CAM 600
<code>sunplus</code>	041e:4012	PC-Cam350
<code>sunplus</code>	041e:4013	Creative Pccam750
<code>zc3xx</code>	041e:4017	Creative Webcam Mobile PD1090
<code>spca508</code>	041e:4018	Creative Webcam Vista (PD1100)
<code>spca561</code>	041e:401a	Creative Webcam Vista (PD1100)
<code>zc3xx</code>	041e:401c	Creative NX
<code>spca505</code>	041e:401d	Creative Webcam NX ULTRA
<code>zc3xx</code>	041e:401e	Creative Nx Pro
<code>zc3xx</code>	041e:401f	Creative Webcam Notebook PD1171
<code>zc3xx</code>	041e:4022	Webcam NX Pro
<code>pac207</code>	041e:4028	Creative Webcam Vista Plus
<code>zc3xx</code>	041e:4029	Creative WebCam Vista Pro
<code>zc3xx</code>	041e:4034	Creative Instant P0620
<code>zc3xx</code>	041e:4035	Creative Instant P0620D
<code>zc3xx</code>	041e:4036	Creative Live !
<code>sq930x</code>	041e:4038	Creative Joy-IT
<code>zc3xx</code>	041e:403a	Creative Nx Pro 2
<code>spca561</code>	041e:403b	Creative Webcam Vista (VF0010)
<code>sq930x</code>	041e:403c	Creative Live! Ultra
<code>sq930x</code>	041e:403d	Creative Live! Ultra for Notebooks
<code>sq930x</code>	041e:4041	Creative Live! Motion
<code>zc3xx</code>	041e:4051	Creative Live!Cam Notebook Pro (VF0250)
<code>ov519</code>	041e:4052	Creative Live! VISTA IM
<code>zc3xx</code>	041e:4053	Creative Live!Cam Video IM
<code>vc032x</code>	041e:405b	Creative Live! Cam Notebook Ultra (VC0130)
<code>ov519</code>	041e:405f	Creative Live! VISTA VF0330
<code>ov519</code>	041e:4060	Creative Live! VISTA VF0350
<code>ov519</code>	041e:4061	Creative Live! VISTA VF0400

Continued on next page

Table 3 - continued from previous page

<i>driver</i>	<i>vend:prod</i>	<i>Device</i>
ov519	041e:4064	Creative Live! VISTA VF0420
ov519	041e:4067	Creative Live! Cam Video IM (VF0350)
ov519	041e:4068	Creative Live! VISTA VF0470
sn9c2028	0458:7003	GeniusVideocam Live v2
spca561	0458:7004	Genius VideoCAM Express V2
sn9c2028	0458:7005	Genius Smart 300, version 2
sunplus	0458:7006	Genius Dsc 1.3 Smart
zc3xx	0458:7007	Genius VideoCam V2
zc3xx	0458:700c	Genius VideoCam V3
zc3xx	0458:700f	Genius VideoCam Web V2
sonixj	0458:7025	Genius Eye 311Q
sn9c20x	0458:7029	Genius Look 320s
sonixj	0458:702e	Genius Slim 310 NB
sn9c20x	0458:7045	Genius Look 1320 V2
sn9c20x	0458:704a	Genius Slim 1320
sn9c20x	0458:704c	Genius i-Look 1321
sn9c20x	045e:00f4	LifeCam VX-6000 (SN9C20x + OV9650)
sonixj	045e:00f5	MicroSoft VX3000
sonixj	045e:00f7	MicroSoft VX1000
ov519	045e:028c	Micro\$oft xbox cam
kinect	045e:02ae	Xbox NUI Camera
kinect	045e:02bf	Kinect for Windows NUI Camera
spca561	0461:0815	Micro Innovations IC200 Webcam
sunplus	0461:0821	Fujifilm MV-1
zc3xx	0461:0a00	MicroInnovation WebCam320
stv06xx	046D:08F0	QuickCamMessenger
stv06xx	046D:08F5	QuickCamCommunicate
stv06xx	046D:08F6	QuickCamMessenger (new)
stv06xx	046d:0840	QuickCamExpress
stv06xx	046d:0850	LEGOcam / QuickCam Web
stv06xx	046d:0870	DexxaWebCam USB
spca500	046d:0890	Logitech QuickCam traveler
vc032x	046d:0892	Logitech Orbicam
vc032x	046d:0896	Logitech Orbicam
vc032x	046d:0897	Logitech QuickCam for Dell notebooks
zc3xx	046d:089d	Logitech QuickCam E2500
zc3xx	046d:08a0	Logitech QC IM
zc3xx	046d:08a1	Logitech QC IM 0x08A1 +sound
zc3xx	046d:08a2	Labtec Webcam Pro
zc3xx	046d:08a3	Logitech QC Chat
zc3xx	046d:08a6	Logitech QCim
zc3xx	046d:08a7	Logitech QuickCam Image
zc3xx	046d:08a9	Logitech Notebook Deluxe
zc3xx	046d:08aa	Labtec Webcam Notebook
zc3xx	046d:08ac	Logitech QuickCam Cool
zc3xx	046d:08ad	Logitech QCCommunicate STX
zc3xx	046d:08ae	Logitech QuickCam for Notebooks

Continued on next page

Table 3 - continued from previous page

<i>driver</i>	<i>vend:prod</i>	<i>Device</i>
zc3xx	046d:08af	Logitech QuickCam Cool
zc3xx	046d:08b9	Logitech QuickCam Express
zc3xx	046d:08d7	Logitech QCam STX
zc3xx	046d:08d8	Logitech Notebook Deluxe
zc3xx	046d:08d9	Logitech QuickCam IM/Connect
zc3xx	046d:08da	Logitech QuickCam Messenger
zc3xx	046d:08dd	Logitech QuickCam for Notebooks
spca500	046d:0900	Logitech Inc. ClickSmart 310
spca500	046d:0901	Logitech Inc. ClickSmart 510
sunplus	046d:0905	Logitech ClickSmart 820
tv8532	046d:0920	Logitech QuickCam Express
tv8532	046d:0921	Labtec Webcam
spca561	046d:0928	Logitech QC Express Etch2
spca561	046d:0929	Labtec Webcam Elch2
spca561	046d:092a	Logitech QC for Notebook
spca561	046d:092b	Labtec Webcam Plus
spca561	046d:092c	Logitech QC chat Elch2
spca561	046d:092d	Logitech QC Elch2
spca561	046d:092e	Logitech QC Elch2
spca561	046d:092f	Logitech QuickCam Express Plus
sunplus	046d:0960	Logitech ClickSmart 420
nw80x	046d:d001	Logitech QuickCam Pro (dark focus ring)
se401	0471:030b	PhilipsPCVC665K
sunplus	0471:0322	Philips DMVC1300K
zc3xx	0471:0325	Philips SPC 200 NC
zc3xx	0471:0326	Philips SPC 300 NC
sonixj	0471:0327	Philips SPC 600 NC
sonixj	0471:0328	Philips SPC 700 NC
zc3xx	0471:032d	Philips SPC 210 NC
zc3xx	0471:032e	Philips SPC 315 NC
sonixj	0471:0330	Philips SPC 710 NC
se401	047d:5001	Kensington67014
se401	047d:5002	Kensington6701(5/7)
se401	047d:5003	Kensington67016
spca501	0497:c001	Smile International
sunplus	04a5:3003	Benq DC 1300
sunplus	04a5:3008	Benq DC 1500
sunplus	04a5:300a	Benq DC 3410
spca500	04a5:300c	Benq DC 1016
benq	04a5:3035	Benq DC E300
vicam	04c1:009d	HomeConnect Webcam [vicam]
konica	04c8:0720	IntelYC 76
finepix	04cb:0104	Fujifilm FinePix 4800
finepix	04cb:0109	Fujifilm FinePix A202
finepix	04cb:010b	Fujifilm FinePix A203
finepix	04cb:010f	Fujifilm FinePix A204
finepix	04cb:0111	Fujifilm FinePix A205

Continued on next page

Table 3 – continued from previous page

<i>driver</i>	<i>vend:prod</i>	<i>Device</i>
finepix	04cb:0113	Fujifilm FinePix A210
finepix	04cb:0115	Fujifilm FinePix A303
finepix	04cb:0117	Fujifilm FinePix A310
finepix	04cb:0119	Fujifilm FinePix F401
finepix	04cb:011b	Fujifilm FinePix F402
finepix	04cb:011d	Fujifilm FinePix F410
finepix	04cb:0121	Fujifilm FinePix F601
finepix	04cb:0123	Fujifilm FinePix F700
finepix	04cb:0125	Fujifilm FinePix M603
finepix	04cb:0127	Fujifilm FinePix S300
finepix	04cb:0129	Fujifilm FinePix S304
finepix	04cb:012b	Fujifilm FinePix S500
finepix	04cb:012d	Fujifilm FinePix S602
finepix	04cb:012f	Fujifilm FinePix S700
finepix	04cb:0131	Fujifilm FinePix unknown model
finepix	04cb:013b	Fujifilm FinePix unknown model
finepix	04cb:013d	Fujifilm FinePix unknown model
finepix	04cb:013f	Fujifilm FinePix F420
sunplus	04f1:1001	JVC GC A50
spca561	04fc:0561	Flexcam 100
spca1528	04fc:1528	Sunplus MD80 clone
sunplus	04fc:500c	Sunplus CA500C
sunplus	04fc:504a	Aiptek Mini PenCam 1.3
sunplus	04fc:504b	Maxell MaxPocket LE 1.3
sunplus	04fc:5330	Digitrex 2110
sunplus	04fc:5360	Sunplus Generic
spca500	04fc:7333	PalmPixDC85
sunplus	04fc:ffff	Pure DigitalDakota
nw80x	0502:d001	DVC V6
spca501	0506:00df	3Com HomeConnect Lite
sunplus	052b:1507	Megapixel 5 Pretec DC-1007
sunplus	052b:1513	Megapix V4
sunplus	052b:1803	MegaImage VI
nw80x	052b:d001	EZCam Pro p35u
tv8532	0545:808b	Veo Stingray
tv8532	0545:8333	Veo Stingray
sunplus	0546:3155	Polaroid PDC3070
sunplus	0546:3191	Polaroid Ion 80
sunplus	0546:3273	Polaroid PDC2030
touptek	0547:6801	TTUCMOS08000KPB, AS MU800
dtcs033	0547:7303	Anchor Chips, Inc
ov519	054c:0154	Sonny toy4
ov519	054c:0155	Sonny toy5
cpia1	0553:0002	CPIA CPiA (version1) based cameras
stv0680	0553:0202	STV0680 Camera
zc3xx	055f:c005	Mustek Wcam300A
spca500	055f:c200	Mustek Gsmart 300

Continued on next page

Table 3 - continued from previous page

<i>driver</i>	<i>vend:prod</i>	<i>Device</i>
sunplus	055f:c211	Kowa Bs888e Microcamera
spca500	055f:c220	Gsmart Mini
sunplus	055f:c230	Mustek Digicam 330K
sunplus	055f:c232	Mustek MDC3500
sunplus	055f:c360	Mustek DV4000 Mpeg4
sunplus	055f:c420	Mustek gSmart Mini 2
sunplus	055f:c430	Mustek Gsmart LCD 2
sunplus	055f:c440	Mustek DV 3000
sunplus	055f:c520	Mustek gSmart Mini 3
sunplus	055f:c530	Mustek Gsmart LCD 3
sunplus	055f:c540	Gsmart D30
sunplus	055f:c630	Mustek MDC4000
sunplus	055f:c650	Mustek MDC5500Z
nw80x	055f:d001	Mustek Wcam 300 mini
zc3xx	055f:d003	Mustek WCam300A
zc3xx	055f:d004	Mustek WCam300 AN
conex	0572:0041	Creative Notebook cx11646
ov519	05a9:0511	Video Blaster WebCam 3/WebCam Plus, D-Link USB Digital Video Camera
ov519	05a9:0518	Creative WebCam
ov519	05a9:0519	OV519 Microphone
ov519	05a9:0530	OmniVision
ov534_9	05a9:1550	OmniVision VEHO Filmscanner
ov519	05a9:2800	OmniVision SuperCAM
ov519	05a9:4519	Webcam Classic
ov534_9	05a9:8065	OmniVision test kit ov538+ov9712
ov519	05a9:8519	OmniVision
ov519	05a9:a511	D-Link USB Digital Video Camera
ov519	05a9:a518	D-Link DSB-C310 Webcam
sunplus	05da:1018	Digital Dream Enigma 1.3
stk014	05e1:0893	Syntek DV4000
gl860	05e3:0503	Genesys Logic PC Camera
gl860	05e3:f191	Genesys Logic PC Camera
vicam	0602:1001	ViCam Webcam
spca561	060b:a001	Maxell Compact Pc PM3
zc3xx	0698:2003	CTX M730V built in
topro	06a2:0003	TP6800 PC Camera, CmoX CX0342 webcam
topro	06a2:6810	Creative Qmax
nw80x	06a5:0000	Typhoon Webcam 100 USB
nw80x	06a5:d001	Divio based webcams
nw80x	06a5:d800	Divio Chicony TwinkleCam, Trust SpaceCam
spca500	06bd:0404	Agfa CL20
spca500	06be:0800	Optimedia
nw80x	06be:d001	EZCam Pro p35u
sunplus	06d6:0031	Trust 610 LCD PowerC@m Zoom
sunplus	06d6:0041	Aashima Technology B.V.
spca506	06e1:a190	ADS Instant VCD
ov534	06f8:3002	Hercules Blog Webcam

Continued on next page

Table 3 - continued from previous page

<i>driver</i>	<i>vend:prod</i>	<i>Device</i>
ov534_9	06f8:3003	Hercules Dualpix HD Weblog
sonixj	06f8:3004	Hercules Classic Silver
sonixj	06f8:3008	Hercules Deluxe Optical Glass
pac7302	06f8:3009	Hercules Classic Link
pac7302	06f8:301b	Hercules Link
nw80x	0728:d001	AVerMedia Camguard
spca508	0733:0110	ViewQuest VQ110
spca501	0733:0401	Intel Create and Share
spca501	0733:0402	ViewQuest M318B
spca505	0733:0430	Intel PC Camera Pro
sunplus	0733:1311	Digital Dream Epsilon 1.3
sunplus	0733:1314	Mercury 2.1MEG Deluxe Classic Cam
sunplus	0733:2211	Jenoptik jdc 21 LCD
sunplus	0733:2221	Mercury Digital Pro 3.1p
sunplus	0733:3261	Concord 3045 spca536a
sunplus	0733:3281	Cyberpix S550V
spca506	0734:043b	3DeMon USB Capture aka
cpia1	0813:0001	QX3 camera
ov519	0813:0002	Dual Mode USB Camera Plus
spca500	084d:0003	D-Link DSC-350
spca500	08ca:0103	Aiptek PocketDV
sunplus	08ca:0104	Aiptek PocketDVII 1.3
sunplus	08ca:0106	Aiptek Pocket DV3100+
mr97310a	08ca:0110	Trust Spyc@m 100
mr97310a	08ca:0111	Aiptek PenCam VGA+
sunplus	08ca:2008	Aiptek Mini PenCam 2 M
sunplus	08ca:2010	Aiptek PocketCam 3M
sunplus	08ca:2016	Aiptek PocketCam 2 Mega
sunplus	08ca:2018	Aiptek Pencam SD 2M
sunplus	08ca:2020	Aiptek Slim 3000F
sunplus	08ca:2022	Aiptek Slim 3200
sunplus	08ca:2024	Aiptek DV3500 Mpeg4
sunplus	08ca:2028	Aiptek PocketCam4M
sunplus	08ca:2040	Aiptek PocketDV4100M
sunplus	08ca:2042	Aiptek PocketDV5100
sunplus	08ca:2050	Medion MD 41437
sunplus	08ca:2060	Aiptek PocketDV5300
tv8532	0923:010f	ICM532 cams
mr97310a	093a:010e	All known CIF cams with this ID
mr97310a	093a:010f	All known VGA cams with this ID
mars	093a:050f	Mars-Semi Pc-Camera
pac207	093a:2460	Qttec Webcam 100
pac207	093a:2461	HP Webcam
pac207	093a:2463	Philips SPC 220 NC
pac207	093a:2464	Labtec Webcam 1200
pac207	093a:2468	Webcam WB-1400T
pac207	093a:2470	Genius GF112

Continued on next page

Table 3 - continued from previous page

<i>driver</i>	<i>vend:prod</i>	<i>Device</i>
pac207	093a:2471	Genius VideoCam ge111
pac207	093a:2472	Genius VideoCam ge110
pac207	093a:2474	Genius iLook 111
pac207	093a:2476	Genius e-Messenger 112
pac7311	093a:2600	PAC7311 Typhoon
pac7311	093a:2601	Philips SPC 610 NC
pac7311	093a:2603	Philips SPC 500 NC
pac7311	093a:2608	Trust WB-3300p
pac7311	093a:260e	Gigaware VGA PC Camera, Trust WB-3350p, SIGMA cam 2350
pac7311	093a:260f	SnakeCam
pac7302	093a:2620	Apollo AC-905
pac7302	093a:2621	PAC731x
pac7302	093a:2622	Genius Eye 312
pac7302	093a:2623	Pixart Imaging, Inc.
pac7302	093a:2624	PAC7302
pac7302	093a:2625	Genius iSlim 310
pac7302	093a:2626	Labtec 2200
pac7302	093a:2627	Genius FaceCam 300
pac7302	093a:2628	Genius iLook 300
pac7302	093a:2629	Genius iSlim 300
pac7302	093a:262a	Webcam 300k
pac7302	093a:262c	Philips SPC 230 NC
jl2005bcd	0979:0227	Various brands, 19 known cameras supported
jeilinj	0979:0270	Sakar 57379
jeilinj	0979:0280	Sportscam DV15, Sakar 57379
zc3xx	0ac8:0301	Web Camera
zc3xx	0ac8:0302	Z-star Vimicro zc0302
vc032x	0ac8:0321	Vimicro generic vc0321
vc032x	0ac8:0323	Vimicro Vc0323
vc032x	0ac8:0328	A4Tech PK-130MG
zc3xx	0ac8:301b	Z-Star zc301b
zc3xx	0ac8:303b	Vimicro 0x303b
zc3xx	0ac8:305b	Z-star Vimicro zc0305b
zc3xx	0ac8:307b	PC Camera (ZS0211)
vc032x	0ac8:c001	Sony embedded vimicro
vc032x	0ac8:c002	Sony embedded vimicro
vc032x	0ac8:c301	Samsung Q1 Ultra Premium
spca508	0af9:0010	Hama USB Sightcam 100
spca508	0af9:0011	Hama USB Sightcam 100
ov519	0b62:0059	iBOT2 Webcam
sonixb	0c45:6001	Genius VideoCAM NB
sonixb	0c45:6005	Microdia Sweex Mini Webcam
sonixb	0c45:6007	Sonix sn9c101 + Tas5110D
sonixb	0c45:6009	spcaCam@120
sonixb	0c45:600d	spcaCam@120
sonixb	0c45:6011	Microdia PC Camera (SN9C102)
sonixb	0c45:6019	Generic Sonix OV7630

Continued on next page

Table 3 - continued from previous page

<i>driver</i>	<i>vend:prod</i>	<i>Device</i>
sonixb	0c45:6024	Generic Sonix Tas5130c
sonixb	0c45:6025	Xcam Shanga
sonixb	0c45:6027	GeniusEye 310
sonixb	0c45:6028	Sonix Btc Pc380
sonixb	0c45:6029	spcaCam@150
sonixb	0c45:602a	Meade ETX-105EC Camera
sonixb	0c45:602c	Generic Sonix OV7630
sonixb	0c45:602d	LIC-200 LG
sonixb	0c45:602e	Genius VideoCam Messenger
sonixj	0c45:6040	Speed NVC 350K
sonixj	0c45:607c	Sonix sn9c102p Hv7131R
sonixb	0c45:6083	VideoCAM Look
sonixb	0c45:608c	VideoCAM Look
sonixb	0c45:608f	PC Camera (SN9C103 + OV7630)
sonixb	0c45:60a8	VideoCAM Look
sonixb	0c45:60aa	VideoCAM Look
sonixb	0c45:60af	VideoCAM Look
sonixb	0c45:60b0	Genius VideoCam Look
sonixj	0c45:60c0	Sangha Sn535
sonixj	0c45:60ce	USB-PC-Camera-168 (TALK-5067)
sonixj	0c45:60ec	SN9C105+MO4000
sonixj	0c45:60fb	Surfer NoName
sonixj	0c45:60fc	LG-LIC300
sonixj	0c45:60fe	Microdia Audio
sonixj	0c45:6100	PC Camera (SN9C128)
sonixj	0c45:6102	PC Camera (SN9C128)
sonixj	0c45:610a	PC Camera (SN9C128)
sonixj	0c45:610b	PC Camera (SN9C128)
sonixj	0c45:610c	PC Camera (SN9C128)
sonixj	0c45:610e	PC Camera (SN9C128)
sonixj	0c45:6128	Microdia/Sonix SNP325
sonixj	0c45:612a	Avant Camera
sonixj	0c45:612b	Speed-Link REFLECT2
sonixj	0c45:612c	Typhoon Rasy Cam 1.3MPix
sonixj	0c45:612e	PC Camera (SN9C110)
sonixj	0c45:6130	Sonix Pccam
sonixj	0c45:6138	Sn9c120 Mo4000
sonixj	0c45:613a	Microdia Sonix PC Camera
sonixj	0c45:613b	Surfer SN-206
sonixj	0c45:613c	Sonix Pccam168
sonixj	0c45:613e	PC Camera (SN9C120)
sonixj	0c45:6142	Hama PC-Webcam AC-150
sonixj	0c45:6143	Sonix Pccam168
sonixj	0c45:6148	Digitus DA-70811/ZSMC USB PC Camera ZS211/Microdia
sonixj	0c45:614a	Frontech E-Ccam (JIL-2225)
sn9c20x	0c45:6240	PC Camera (SN9C201 + MT9M001)
sn9c20x	0c45:6242	PC Camera (SN9C201 + MT9M111)

Continued on next page

Table 3 - continued from previous page

<i>driver</i>	<i>vend:prod</i>	<i>Device</i>
sn9c20x	0c45:6248	PC Camera (SN9C201 + OV9655)
sn9c20x	0c45:624c	PC Camera (SN9C201 + MT9M112)
sn9c20x	0c45:624e	PC Camera (SN9C201 + SOI968)
sn9c20x	0c45:624f	PC Camera (SN9C201 + OV9650)
sn9c20x	0c45:6251	PC Camera (SN9C201 + OV9650)
sn9c20x	0c45:6253	PC Camera (SN9C201 + OV9650)
sn9c20x	0c45:6260	PC Camera (SN9C201 + OV7670)
sn9c20x	0c45:6270	PC Camera (SN9C201 + MT9V011/MT9V111/MT9V112)
sn9c20x	0c45:627b	PC Camera (SN9C201 + OV7660)
sn9c20x	0c45:627c	PC Camera (SN9C201 + HV7131R)
sn9c20x	0c45:627f	PC Camera (SN9C201 + OV9650)
sn9c20x	0c45:6280	PC Camera (SN9C202 + MT9M001)
sn9c20x	0c45:6282	PC Camera (SN9C202 + MT9M111)
sn9c20x	0c45:6288	PC Camera (SN9C202 + OV9655)
sn9c20x	0c45:628c	PC Camera (SN9C201 + MT9M112)
sn9c20x	0c45:628e	PC Camera (SN9C202 + SOI968)
sn9c20x	0c45:628f	PC Camera (SN9C202 + OV9650)
sn9c20x	0c45:62a0	PC Camera (SN9C202 + OV7670)
sn9c20x	0c45:62b0	PC Camera (SN9C202 + MT9V011/MT9V111/MT9V112)
sn9c20x	0c45:62b3	PC Camera (SN9C202 + OV9655)
sn9c20x	0c45:62bb	PC Camera (SN9C202 + OV7660)
sn9c20x	0c45:62bc	PC Camera (SN9C202 + HV7131R)
sn9c2028	0c45:8001	Wild Planet Digital Spy Camera
sn9c2028	0c45:8003	Sakar #11199, #6637x, #67480 keychain cams
sn9c2028	0c45:8008	Mini-Shotz ms-350
sn9c2028	0c45:800a	Vivitar Vivicam 3350B
sunplus	0d64:0303	Sunplus FashionCam DXG
ov519	0e96:c001	TRUST 380 USB2 SPACEC@M
etoms	102c:6151	Qcam Sangha CIF
etoms	102c:6251	Qcam xxxxxx VGA
ov519	1046:9967	W9967CF/W9968CF WebCam IC, Video Blaster WebCam Go
zc3xx	10fd:0128	Typhoon Webshot II USB 300k 0x0128
spca561	10fd:7e50	FlyCam Usb 100
zc3xx	10fd:804d	Typhoon Webshot II Webcam [zc0301]
zc3xx	10fd:8050	Typhoon Webshot II USB 300k
ov534	1415:2000	Sony HD Eye for PS3 (SLEH 00201)
pac207	145f:013a	Trust WB-1300N
pac7302	145f:013c	Trust
sn9c20x	145f:013d	Trust WB-3600R
vc032x	15b8:6001	HP 2.0 Megapixel
vc032x	15b8:6002	HP 2.0 Megapixel rz406aa
stk1135	174f:6a31	ASUSlaptop, MT9M112 sensor
spca501	1776:501c	Arowana 300K CMOS Camera
t613	17a1:0128	TASCORP JPEG Webcam, NGS Cyclops
vc032x	17ef:4802	Lenovo Vc0323+MI1310_SOC
pac7302	1ae7:2001	SpeedLinkSnappy Mic SL-6825-SBK
pac207	2001:f115	D-Link DSB-C120

Continued on next page

Table 3 – continued from previous page

<i>driver</i>	<i>vend:prod</i>	<i>Device</i>
sq905c	2770:9050	Disney pix micro (CIF)
sq905c	2770:9051	Lego Bionicle
sq905c	2770:9052	Disney pix micro 2 (VGA)
sq905c	2770:905c	All 11 known cameras with this ID
sq905	2770:9120	All 24 known cameras with this ID
sq905c	2770:913d	All 4 known cameras with this ID
sq930x	2770:930b	Sweex Motion Tracking / I-Tec iCam Tracer
sq930x	2770:930c	Trust WB-3500T / NSG Robbie 2.0
spca500	2899:012c	Topstro Industrial
ov519	8020:ef04	ov519
spca508	8086:0110	Intel Easy PC Camera
spca500	8086:0630	Intel Pocket PC Camera
spca506	99fa:8988	Grandtec V.cap
sn9c20x	a168:0610	Dino-Lite Digital Microscope (SN9C201 + HV7131R)
sn9c20x	a168:0611	Dino-Lite Digital Microscope (SN9C201 + HV7131R)
sn9c20x	a168:0613	Dino-Lite Digital Microscope (SN9C201 + HV7131R)
sn9c20x	a168:0614	Dino-Lite Digital Microscope (SN9C201 + MT9M111)
sn9c20x	a168:0615	Dino-Lite Digital Microscope (SN9C201 + MT9M111)
sn9c20x	a168:0617	Dino-Lite Digital Microscope (SN9C201 + MT9M111)
sn9c20x	a168:0618	Dino-Lite Digital Microscope (SN9C201 + HV7131R)
spca561	abcd:cdee	Petcam

dvb-usb-dib0700 cards list

Card name	USB IDs
ASUS My Cinema U3000 Mini DVBT Tuner	0b05:171f
ASUS My Cinema U3100 Mini DVBT Tuner	0b05:173f
AVerMedia AVerTV DVB-T Express	07ca:b568
AVerMedia AVerTV DVB-T Volar	07ca:a807, 07ca:b808
Artec T14BR DVB-T	05d8:810f
Asus My Cinema-U3000Hybrid	0b05:1736
Compro Videomate U500	185b:1e78, 185b:1e80
DiBcom NIM7090 reference design	10b8:1bb2
DiBcom NIM8096MD reference design	10b8:1fa8
DiBcom NIM9090MD reference design	10b8:2384
DiBcom STK7070P reference design	10b8:1ebc
DiBcom STK7070PD reference design	10b8:1ebe
DiBcom STK7700D reference design	10b8:1ef0
DiBcom STK7700P reference design	10b8:1e14, 10b8:1e78

Continued on next page

Table 4 - continued from previous page

Card name	USB IDs
DiBcom STK7770P reference design	10b8:1e80
DiBcom STK807xP reference design	10b8:1f90
DiBcom STK807xPVR reference design	10b8:1f98
DiBcom STK8096-PVR reference design	2013:1faa, 10b8:1faa
DiBcom STK8096GP reference design	10b8:1fa0
DiBcom STK9090M reference design	10b8:2383
DiBcom TFE7090PVR reference design	10b8:1bb4
DiBcom TFE7790P reference design	10b8:1e6e
DiBcom TFE8096P reference design	10b8:1f9C
Elgato EyeTV DTT	0fd9:0021
Elgato EyeTV DTT rev. 2	0fd9:003f
Elgato EyeTV Diversity	0fd9:0011
Elgato EyeTV Dtt Dlx PD378S	0fd9:0020
EvolutePC TVWay+	1e59:0002
Gigabyte U7000	1044:7001
Gigabyte U8000-RH	1044:7002
Hama DVB-T Hybrid USB Stick	147f:2758
Hauppauge ATSC MiniCard (B200)	2040:b200
Hauppauge ATSC MiniCard (B210)	2040:b210
Hauppauge Nova-T 500 Dual DVB-T	2040:9941, 2040:9950
Hauppauge Nova-T MyTV.t	2040:7080
Hauppauge Nova-T Stick	2040:7050, 2040:7060, 2040:7070
Hauppauge Nova-TD Stick (52009)	2040:5200
Hauppauge Nova-TD Stick/Elgato Eye-TV Diversity	2040:9580
Hauppauge Nova-TD-500 (84xxx)	2040:8400
Leadtek WinFast DTV Dongle H	0413:60f6
Leadtek Winfast DTV Dongle (STK7700P based)	0413:6f00, 0413:6f01
Medion CTX1921 DVB-T USB	1660:1921
Microsoft Xbox One Digital TV Tuner	045e:02d5
PCTV 2002e	2013:025c
PCTV 2002e SE	2013:025d
Pinnacle Expresscard 320cx	2304:022e
Pinnacle PCTV 2000e	2304:022c
Pinnacle PCTV 282e	2013:0248, 2304:0248
Pinnacle PCTV 340e HD Pro USB Stick	2304:023d
Pinnacle PCTV 72e	2304:0236
Pinnacle PCTV 73A	2304:0243
Pinnacle PCTV 73e	2304:0237

Continued on next page

Table 4 - continued from previous page

Card name	USB IDs
Pinnacle PCTV 73e SE	2013:0245, 2304:0245
Pinnacle PCTV DVB-T Flash Stick	2304:0228
Pinnacle PCTV Dual DVB-T Diversity Stick	2304:0229
Pinnacle PCTV HD Pro USB Stick	2304:023a
Pinnacle PCTV HD USB Stick	2304:023b
Pinnacle PCTV Hybrid Stick Solo	2304:023e
Prolink Pixelview SBTVD	1554:5010
Sony PlayTV	1415:0003
TechniSat AirStar TeleStick 2	14f7:0004
Terratec Cinergy DT USB XS Diversity/ T5	0ccd:0081, 0ccd:10a1
Terratec Cinergy DT XS Diversity	0ccd:005a
Terratec Cinergy HT Express	0ccd:0060
Terratec Cinergy HT USB XE	0ccd:0058
Terratec Cinergy T Express	0ccd:0062
Terratec Cinergy T USB XXS (HD)/ T3	0ccd:0078, 0ccd:10a0, 0ccd:00ab
Uniwill STK7700P based (Hama and others)	1584:6003
YUAN High-Tech DiBcom STK7700D	1164:1e8c
YUAN High-Tech MC770	1164:0871
YUAN High-Tech STK7700D	1164:1efc
YUAN High-Tech STK7700PH	1164:1f08
Yuan EC372S	1164:1edc
Yuan PD378S	1164:2edc

dvb-usb-dibusb-mb cards list

Card name	USB IDs
AVerMedia AverTV DVBT USB1.1	14aa:0001, 14aa:0002
Artec T1 USB1.1 TVBOX with AN2135	05d8:8105, 05d8:8106
Artec T1 USB1.1 TVBOX with AN2235	05d8:8107, 05d8:8108
Artec T1 USB1.1 TVBOX with AN2235 (faulty USB IDs)	0547:2235
Artec T1 USB2.0	05d8:8109, 05d8:810a
Compro Videomate DVB-U2000 - DVB-T USB1.1 (please confirm to linux-dvb)	185b:d000, 145f:010c, 185b:d001
DiBcom USB1.1 DVB-T reference design (MOD3000)	10b8:0bb8, 10b8:0bb9
Grandtec USB1.1 DVB-T	5032:0fa0, 5032:0bb8, 5032:0fa1, 5032:0bb9
KWorld V-Stream XPERT DTV - DVB-T USB1.1	eb1a:17de, eb1a:17df
KWorld Xpert DVB-T USB2.0	eb2a:17de
KWorld/ADSTech Instant DVB-T USB2.0	06e1:a333, 06e1:a334
TwinhanDTV USB-Ter USB1.1 / Magic Box I / HAMA USB1.1 DVB-T device	13d3:3201, 1822:3201, 13d3:3202, 1822:3202
Unknown USB1.1 DVB-T device ???? please report the name to the author	1025:005e, 1025:005f
VideoWalker DVB-T USB	0458:701e, 0458:701f

dvb-usb-dibusb-mc cards list

Card name	USB IDs
Artec T1 USB2.0 TVBOX (please check the warm ID)	05d8:8109, 05d8:810a
Artec T14 - USB2.0 DVB-T	05d8:810b, 05d8:810c
DiBcom USB2.0 DVB-T reference design (MOD3000P)	10b8:0bc6, 10b8:0bc7
GRAND - USB2.0 DVB-T adapter	5032:0bc6, 5032:0bc7
Humax/Coex DVB-T USB Stick 2.0 High Speed	10b9:5000, 10b9:5001
LITE-ON USB2.0 DVB-T Tuner	04ca:f000, 04ca:f001
Leadtek - USB2.0 Winfast DTV dongle	0413:6025, 0413:6026
MSI Digivox Mini SL	eb1a:e360, eb1a:e361

dvb-usb-a800 cards list

Card name	USB IDs
AVerMedia AverTV DVB-T USB 2.0 (A800)	07ca:a800, 07ca:a801

dvb-usb-af9005 cards list

Card name	USB IDs
Afatech DVB-T USB1.1 stick	15a4:9020
Ansonic DVB-T USB1.1 stick	10b9:6000
TerraTec Cinergy T USB XE	0ccd:0055

dvb-usb-az6027 cards list

Card name	USB IDs
AZUREWAVE DVB-S/S2 USB2.0 (AZ6027)	13d3:3275
Elgato EyeTV Sat	0fd9:002a, 0fd9:0025, 0fd9:0036
TERRATEC S7	0ccd:10a4
TERRATEC S7 MKII	0ccd:10ac
Technisat SkyStar USB 2 HD CI	14f7:0001, 14f7:0002

dvb-usb-cinergyT2 cards list

Card name	USB IDs
TerraTec/qanu USB2.0 Highspeed DVB-T Receiver	0ccd:0x0038

dvb-usb-cxusb cards list

Card name	USB IDs
AVerMedia AVerTVHD Volar (A868R)	
Conexant DMB-TH Stick	
DViCO FusionHDTV DVB-T Dual Digital 2	
DViCO FusionHDTV DVB-T Dual Digital 4	
DViCO FusionHDTV DVB-T Dual Digital 4 (rev 2)	
DViCO FusionHDTV DVB-T Dual USB	
DViCO FusionHDTV DVB-T NANO2	
DViCO FusionHDTV DVB-T USB (LGZ201)	
DViCO FusionHDTV DVB-T USB (TH7579)	
DViCO FusionHDTV5 USB Gold	
DigitalNow DVB-T Dual USB	
Medion MD95700 (MDUSBTV-HYBRID)	
Mygica D689 DMB-TH	

dvb-usb-digitv cards list

Card name	USB IDs
Nebula Electronics uDigiTV DVB-T USB2.0)	0547:0201

dvb-usb-dtt200u cards list

Card name	USB IDs
WideView WT-220U PenType Receiver (Miglia)	18f3:0220
WideView WT-220U PenType Receiver (Typhoon/Freecom)	14aa:0222, 14aa:0220, 14aa:0221, 14aa:0225, 14aa:0226
WideView WT-220U PenType Receiver (based on ZL353)	14aa:022a, 14aa:022b
WideView/Yuan/Yakumo/Hama/Typhoon DVB-T USB2.0 (WT-200U)	14aa:0201, 14aa:0301

dvb-usb-dtv5100 cards list

Card name	USB IDs
AME DTV-5100 USB2.0 DVB-T	0x06be:0xa232

dvb-usb-dw2102 cards list

Card name	USB IDs
DVBWorld DVB-C 3101 USB2.0	04b4:3101
DVBWorld DVB-S 2101 USB2.0	04b4:0x2101
DVBWorld DVB-S 2102 USB2.0	04b4:2102
DVBWorld DW2104 USB2.0	04b4:2104
GOTVIEW Satellite HD	0x1FE1:5456
Geniatech T220 DVB-T/T2 USB2.0	0x1f4d:0xD220
SU3000HD DVB-S USB2.0	0x1f4d:0x3000
TeVii S482 (tuner 1)	0x9022:0xd483
TeVii S482 (tuner 2)	0x9022:0xd484
TeVii S630 USB	0x9022:d630
TeVii S650 USB2.0	0x9022:d650
TeVii S662	0x9022:d662
TechnoTrend TT-connect S2-4600	0b48:3011
TerraTec Cinergy S USB	0ccd:0064
Terratec Cinergy S2 PCIe Dual Port 1	153b:1181
Terratec Cinergy S2 PCIe Dual Port 2	153b:1182
Terratec Cinergy S2 USB BOX	0ccd:0x0105
Terratec Cinergy S2 USB HD	0ccd:00a8
Terratec Cinergy S2 USB HD Rev.2	0ccd:00b0
Terratec Cinergy S2 USB HD Rev.3	0ccd:0102
X3M TV SPC1400HD PCI	0x1f4d:0x3100

dvb-usb-gp8psk cards list

Card name	USB IDs
Genpixon 8PSK-to-USB2 Rev.1 DVB-S receiver	09c0:0200, 09c0:0201
Genpixon 8PSK-to-USB2 Rev.2 DVB-S receiver	09c0:0202
Genpixon SkyWalker-1 DVB-S receiver	09c0:0203
Genpixon SkyWalker-2 DVB-S receiver	09c0:0206

dvb-usb-m920x cards list

Card name	USB IDs
DTV-DVB UDTT7049	13d3:3219
Dposh DVB-T USB2.0	1498:9206, 1498:a090
LifeView TV Walker Twin DVB-T USB2.0	10fd:0514, 10fd:0513
MSI DIGI VOX mini II DVB-T USB2.0	10fd:1513
MSI Mega Sky 580 DVB-T USB2.0	0db0:5580
Pinnacle PCTV 310e	13d3:3211

dvb-usb-nova-t-usb2 cards list

Card name	USB IDs
Hauppauge WinTV-NOVA-T usb2	2040:9300, 2040:9301

dvb-usb-opera1 cards list

Card name	USB IDs
Opera1 DVB-S USB2.0	04b4:2830, 695c:3829

dvb-usb-pctv452e cards list

Card name	USB IDs
PCTV HDTV USB	2304:021f
Technotrend TT Connect S2-3600	0b48:3007
Technotrend TT Connect S2-3650-CI	0b48:300a

dvb-usb-technisat-usb2 cards list

Card name	USB IDs
Technisat SkyStar USB HD (DVB-S/S2)	14f7:0500

dvb-usb-ttusb2 cards list

Card name	USB IDs
Pinnacle 400e DVB-S USB2.0	2304:020f
Pinnacle 450e DVB-S USB2.0	2304:0222
Technotrend TT-connect CT-3650	0b48:300d
Technotrend TT-connect S-2400	0b48:3006
Technotrend TT-connect S-2400 (8kB EEPROM)	0b48:3009

dvb-usb-umt-010 cards list

Card name	USB IDs
Hanftek UMT-010 DVB-T USB2.0	15f4:0001, 15f4:0015

dvb-usb-vp702x cards list

Card name	USB IDs
TwinhanDTV StarBox DVB-S USB2.0 (VP7021)	13d3:3207

dvb-usb-vp7045 cards list

Card name	USB IDs
DigitalNow TinyUSB 2 DVB-t Receiver	13d3:3223, 13d3:3224
Twinhan USB2.0 DVB-T receiver (TwinhanDTV Alpha/MagicBox II)	13d3:3205, 13d3:3206

dvb-usb-af9015 cards list

Card name	USB IDs
AVerMedia A309	07ca:a309
AVerMedia AVerTV DVB-T Volar X	07ca:a815
Afatech AF9015 reference design	15a4:9015, 15a4:9016
AverMedia AVerTV Red HD+ (A850T)	07ca:850b
AverMedia AVerTV Volar Black HD (A850)	07ca:850a
AverMedia AVerTV Volar GPS 805 (A805)	07ca:a805

Continued on next page

Table 5 - continued from previous page

Card name	USB IDs
AverMedia AVerTV Volar M (A815Mac)	07ca:815a
Conceptronic USB2.0 DVB-T CTVDIGRCU V3.0	1b80:e397
DigitalNow TinyTwin	13d3:3226
DigitalNow TinyTwin v2	1b80:e402
DigitalNow TinyTwin v3	1f4d:9016
Fujitsu-Siemens Slim Mobile USB DVB-T	07ca:8150
Genius TVGo DVB-T03	0458:4012
KWorld Digital MC-810	1b80:c810
KWorld PlusTV DVB-T PCI Pro Card (DVB-T PC160-T)	1b80:c161
KWorld PlusTV Dual DVB-T PCI (DVB-T PC160-2T)	1b80:c160
KWorld PlusTV Dual DVB-T Stick (DVB-T 399U)	1b80:e399, 1b80:e400
KWorld USB DVB-T Stick Mobile (UB383-T)	1b80:e383
KWorld USB DVB-T TV Stick II (VS-DVB-T 395U)	1b80:e396, 1b80:e39b, 1b80:e395, 1b80:e39a
Leadtek WinFast DTV Dongle Gold	0413:6029
Leadtek WinFast DTV2000DS	0413:6a04
MSI DIGIVOX Duo	1462:8801
MSI Digi VOX mini III	1462:8807
Pinnacle PCTV 71e	2304:022b
Sveon STV20 Tuner USB DVB-T HDTV	1b80:e39d
Sveon STV22 Dual USB DVB-T Tuner HDTV	1b80:e401
Telestar Starstick 2	10b9:8000
TerraTec Cinergy T Stick Dual RC	0ccd:0099
TerraTec Cinergy T Stick RC	0ccd:0097
TerraTec Cinergy T USB XE	0ccd:0069
TrekStor DVB-T USB Stick	15a4:901b
TwinHan AzureWave AD-TU700(704J)	13d3:3237
Xtensions XD-380	1ae7:0381

dVB-USB-AF9035 cards list

Card name	USB IDs
AVerMedia AVerTV Volar HD/PRO (A835)	07ca:a835, 07ca:b835
AVerMedia HD Volar (A867)	07ca:1867, 07ca:a867, 07ca:0337
AVerMedia TD310 DVB-T2	07ca:1871
AVerMedia Twinstar (A825)	07ca:0825
Afatech AF9035 reference design	15a4:9035, 15a4:1000, 15a4:1001, 15a4:1002, 15a4:1003
Asus U3100Mini Plus	0b05:1779
Avermedia A835B(1835)	07ca:1835
Avermedia A835B(2835)	07ca:2835
Avermedia A835B(3835)	07ca:3835
Avermedia A835B(4835)	07ca:4835
Avermedia AVerTV Volar HD 2 (TD110)	07ca:a110
Avermedia H335	07ca:0335
Digital Dual TV Receiver CTVDIG-DUAL_V2	1b80:e410
EVOLVEO XtraTV stick	1f4d:a115
Hauppauge WinTV-MiniStick 2	2040:f900
ITE 9135 Generic	048d:9135
ITE 9135(9005) Generic	048d:9005
ITE 9135(9006) Generic	048d:9006
ITE 9303 Generic	048d:9306
Kworld UB499-2T T09	1b80:e409
Leadtek WinFast DTV Dongle Dual	0413:6a05
Logilink VG0022A	1d19:0100
PCTV AndroidDTV (78e)	2013:025a
PCTV microStick (79e)	2013:0262
Sveon STV22 Dual DVB-T HDTV	1b80:e411
TerraTec Cinergy T Stick	0ccd:0093
TerraTec Cinergy T Stick (rev. 2)	0ccd:00aa
TerraTec Cinergy T Stick Dual RC (rev. 2)	0ccd:0099
TerraTec Cinergy TC2 Stick	0ccd:10b2
TerraTec T1	0ccd:10ae

dvb-usb-anysee cards list

Card name	USB IDs
Anysee	04b4:861f, 1c73:861f

dvb-usb-au6610 cards list

Card name	USB IDs
Sigmathek DVB-110	058f:6610

dvb-usb-az6007 cards list

Card name	USB IDs
Azurewave 6007	13d3:0ccd
Technisat CableStar Combo HD CI	14f7:0003
Terratec H7	0ccd:10b4, 0ccd:10a3

dvb-usb-ce6230 cards list

Card name	USB IDs
AVerMedia A310 USB 2.0 DVB-T tuner	07ca:a310
Intel CE9500 reference design	8086:9500

dvb-usb-dvbsky cards list

Card name	USB IDs
DVBSky S960/S860	0572:6831
DVBSky S960CI	0572:960c
DVBSky T330	0572:0320
DVBSky T680CI	0572:680c
MyGica Mini DVB-(T/T2/C) USB Stick T230	0572:c688
MyGica Mini DVB-(T/T2/C) USB Stick T230C	0572:c689
MyGica Mini DVB-(T/T2/C) USB Stick T230C Lite	0572:c699
MyGica Mini DVB-(T/T2/C) USB Stick T230C v2	0572:c68a
TechnoTrend TT-connect CT2-4650 CI	0b48:3012
TechnoTrend TT-connect CT2-4650 CI v1.1	0b48:3015
TechnoTrend TT-connect S2-4650 CI	0b48:3017
TechnoTrend TVStick CT2-4400	0b48:3014
Terratec Cinergy S2 Rev.4	0ccd:0105
Terratec H7 Rev.4	0ccd:10a5

dvb-usb-ec168 cards list

Card name	USB IDs
E3C EC168 reference design	18b4:1689, 18b4:fffa, 18b4:fffb, 18b4:1001, 18b4:1002

dvb-usb-gl861 cards list

Card name	USB IDs
774 Friio White ISDB-T USB2.0	7a69:0001
A-LINK DTU DVB-T USB2.0	05e3:f170
MSI Mega Sky 55801 DVB-T USB2.0	0db0:5581

dvb-usb-lmedm04 cards list

Card name	USB IDs
DM04_LME2510C_DVB-S	3344:1120
DM04_LME2510C_DVB-S RS2000	3344:22f0
DM04_LME2510_DVB-S	3344:1122

dvb-usb-mxl111sf cards list

Card name	USB IDs
HCW 117xxx	2040:b702
HCW 126xxx	2040:c602, 2040:c60a
Hauppauge 117xxx ATSC+	2040:b700, 2040:b703, 2040:b753, 2040:b763, 2040:b757, 2040:b767
Hauppauge 117xxx DVBT	2040:b704, 2040:b764
Hauppauge 126xxx	2040:c612, 2040:c61a
Hauppauge 126xxx ATSC	2040:c601, 2040:c609, 2040:b701
Hauppauge 126xxx ATSC+	2040:c600, 2040:c603, 2040:c60b, 2040:c653, 2040:c65b
Hauppauge 126xxx DVBT	2040:c604, 2040:c60c
Hauppauge 138xxx DVBT	2040:d854, 2040:d864, 2040:d8d4, 2040:d8e4
Hauppauge Mercury	2040:d853, 2040:d863, 2040:d8d3, 2040:d8e3, 2040:d8ff
Hauppauge WinTV-Aero-M	2040:c613, 2040:c61b

dvb-usb-rtl28xxu cards list

Card name	USB IDs
ASUS My Cinema-U3100Mini Plus V2	1b80:d3a8
Astrometa DVB-T2	15f4:0131
Compro VideoMate U620F	185b:0620
Compro VideoMate U650F	185b:0650
Crypto ReDi PC 50 A	1f4d:a803
Dexatek DK DVB-T Dongle	1d19:1101
Dexatek DK mini DVB-T Dongle	1d19:1102
DigitalNow Quad DVB-T Receiver	0413:6680
Freecom USB2.0 DVB-T	14aa:0160, 14aa:0161
G-Tek Electronics Group Lifeview LV5TDLX DVB-T	1f4d:b803
GIGABYTE U7300	1b80:d393
Genius TVGo DVB-T03	0458:707f
GoTView MasterHD 3	5654:ca42
Leadtek WinFast DTV Dongle mini	0413:6a03

Continued on next page

Table 7 - continued from previous page

Card name	USB IDs
Leadtek WinFast DTV2000DS Plus	0413:6f12
Leadtek Winfast DTV Dongle Mini D	0413:6f0f
MSI DIGIVOX Micro HD	1d19:1104
MaxMedia HU394-T	1b80:d394
PROlectrix DV107669	1f4d:d803
Peak DVB-T USB	1b80:d395
Realtek RTL2831U reference design	0bda:2831
Realtek RTL2832U reference design	0bda:2832, 0bda:2838
Sveon STV20	1b80:d39d
Sveon STV21	1b80:d3b0
Sveon STV27	1b80:d3af
TURBO-X Pure TV Tuner DTT-2000	1b80:d3a4
TerraTec Cinergy T Stick Black	0ccd:00a9
TerraTec Cinergy T Stick RC (Rev. 3)	0ccd:00d3
TerraTec Cinergy T Stick+	0ccd:00d7
TerraTec NOXON DAB Stick	0ccd:00b3
TerraTec NOXON DAB Stick (rev 2)	0ccd:00e0
TerraTec NOXON DAB Stick (rev 3)	0ccd:00b4
Trekstor DVB-T Stick Terres 2.0	1f4d:C803

dvb-usb-zd1301 cards list

Card name	USB IDs
ZyDAS ZD1301 reference design	0ace:13a1

Other USB cards list

Driver	Card name	USB IDs
airspy	Airspy	1d50:60a1
dvb-as102	Abilis Systems DVB-Titan	1BA6:0001
dvb-as102	PCTV Systems picoStick (74e)	2013:0246
dvb-as102	Elgato EyeTV DTT Deluxe	0fd9:002c
dvb-as102	nBox DVB-T Dongle	0b89:0007
dvb-as102	Sky IT Digital Key (green led)	2137:0001
b2c2-flexcop-usb	Technisat/B2C2 FlexCop II/IIb/III Digital TV	0af7:0101
cpia2	Vision's CPiA2 cameras such as the Digital Blue QX5	0553:0100, 0553:014
go7007	WIS GO7007 MPEG encoder	1943:a250, 093b:a00
hackrf	HackRF Software Decoder Radio	1d50:6089
hdpvr	Hauppauge HD PVR	2040:4900, 2040:490
msi2500	Mirics MSi3101 SDR Dongle	1df7:2500, 2040:d30

Driver	Card name	USB IDs
pvrusb2	Hauppauge WinTV-PVR USB2	2040:2900, 2040:2950
pwc	Creative Webcam 5	041E:400C
pwc	Creative Webcam Pro Ex	041E:4011
pwc	Logitech QuickCam 3000 Pro	046D:08B0
pwc	Logitech QuickCam Notebook Pro	046D:08B1
pwc	Logitech QuickCam 4000 Pro	046D:08B2
pwc	Logitech QuickCam Zoom (old model)	046D:08B3
pwc	Logitech QuickCam Zoom (new model)	046D:08B4
pwc	Logitech QuickCam Orbit/Sphere	046D:08B5
pwc	Logitech/Cisco VT Camera	046D:08B6
pwc	Logitech ViewPort AV 100	046D:08B7
pwc	Logitech QuickCam	046D:08B8
pwc	Philips PCA645VC	0471:0302
pwc	Philips PCA646VC	0471:0303
pwc	Askey VC010 type 2	0471:0304
pwc	Philips PCVC675K (Vesta)	0471:0307
pwc	Philips PCVC680K (Vesta Pro)	0471:0308
pwc	Philips PCVC690K (Vesta Pro Scan)	0471:030C
pwc	Philips PCVC730K (ToUCam Fun), PCVC830 (ToUCam II)	0471:0310
pwc	Philips PCVC740K (ToUCam Pro), PCVC840 (ToUCam II)	0471:0311
pwc	Philips PCVC750K (ToUCam Pro Scan)	0471:0312
pwc	Philips PCVC720K/40 (ToUCam XS)	0471:0313
pwc	Philips SPC 900NC	0471:0329
pwc	Philips SPC 880NC	0471:032C
pwc	Sotec Afina Eye	04CC:8116
pwc	Samsung MPC-C10	055D:9000
pwc	Samsung MPC-C30	055D:9001
pwc	Samsung SNC-35E (Ver3.0)	055D:9002
pwc	Askey VC010 type 1	069A:0001
pwc	AME Co. Afina Eye	06BE:8116
pwc	Visionite VCS-UC300	0d81:1900
pwc	Visionite VCS-UM100	0d81:1910
s2255drv	Sensoray 2255	1943:2255, 1943:2256
stk1160	STK1160 USB video capture dongle	05e1:0408
stkwebcam	Syntek DC1125	174f:a311, 05e1:0500
dvb-ttusb-budget	Technotrend/Hauppauge Nova-USB devices	0b48:1003, 0b48:1004
dvb-ttusb_dec	Technotrend/Hauppauge MPEG decoder DEC3000-s	0b48:1006
dvb-ttusb_dec	Technotrend/Hauppauge MPEG decoder	0b48:1007
dvb-ttusb_dec	Technotrend/Hauppauge MPEG decoder DEC2000-t	0b48:1008
dvb-ttusb_dec	Technotrend/Hauppauge MPEG decoder DEC2540-t	0b48:1009
usbtv	Fushicai USBTV007 Audio-Video Grabber	1b71:3002, 1f71:3300
zr364xx	USB ZR364XX Camera	08ca:0109, 041e:4020

1.1.5.2 PCI drivers

The PCI boards are identified by an identification called PCI ID. The PCI ID is actually composed by two parts:

- Vendor ID and device ID;
- Subsystem ID and Subsystem device ID;

The `lspci -nn` command allows identifying the vendor/device PCI IDs:

```
$ lspci -nn
...
00:0a.0 Multimedia controller [0480]: Philips Semiconductors SAA7131/SAA7133/SAA7135
↳Video Broadcast Decoder [1131:7133] (rev d1)
00:0b.0 Multimedia controller [0480]: Brooktree Corporation Bt878 Audio Capture
↳[109e:0878] (rev 11)
01:00.0 Multimedia video controller [0400]: Conexant Systems, Inc. CX23887/8 PCIe
↳Broadcast Audio and Video Decoder with 3D Comb [14f1:8880] (rev 0f)
02:01.0 Multimedia video controller [0400]: Internext Compression Inc iTVC15
↳(CX23415) Video Decoder [4444:0803] (rev 01)
02:02.0 Multimedia video controller [0400]: Conexant Systems, Inc. CX23418 Single-
↳Chip MPEG-2 Encoder with Integrated Analog Video/Broadcast Audio Decoder [14f1:5b7a]
02:03.0 Multimedia video controller [0400]: Brooktree Corporation Bt878 Video Capture
↳[109e:036e] (rev 11)
...
```

The subsystem IDs can be obtained using `lspci -vn`

```
$ lspci -vn
...
00:0a.0 0480: 1131:7133 (rev d1)
Subsystem: 1461:f01d
Flags: bus master, medium devsel, latency 32, IRQ 209
Memory at e2002000 (32-bit, non-prefetchable) [size=2K]
Capabilities: [40] Power Management version 2
...
```

At the above example, the first card uses the `saa7134` driver, and has a vendor/device PCI ID equal to `1131:7133` and a PCI subsystem ID equal to `1461:f01d` (see [Saa7134 card list](#)).

Unfortunately, sometimes the same PCI subsystem ID is used by different products. So, several media drivers allow passing a `card=` parameter, in order to setup a card number that would match the correct settings for an specific board.

The current supported PCI/PCIe cards (not including staging drivers) are listed below¹.

Driver	Name
<code>altera-ci</code>	Altera FPGA based CI module
<code>b2c2-flexcop-pci</code>	Technisat/B2C2 Air/Sky/Cable2PC PCI
<code>bt878</code>	DVB/ATSC Support for bt878 based TV cards
<code>bttv</code>	BT8x8 Video For Linux
<code>cobalt</code>	Cisco Cobalt
<code>cx18</code>	Conexant cx23418 MPEG encoder

Continued on next page

¹ some of the drivers have sub-drivers, not shown at this table

Table 9 - continued from previous page

Driver	Name
cx23885	Conexant cx23885 (2388x successor)
cx25821	Conexant cx25821
cx88xx	Conexant 2388x (bt878 successor)
ddbbridge	Digital Devices bridge
dm1105	SDMC DM1105 based PCI cards
dt3155	DT3155 frame grabber
dvb-ttpci	AV7110 cards
earth-pt1	PT1 cards
earth-pt3	Earthsoft PT3 cards
hexium_gemini	Hexium Gemini frame grabber
hexium_orion	Hexium HV-PCI6 and Orion frame grabber
hopper	HOPPER based cards
ipu3-cio2	Intel ipu3-cio2 driver
ivtv	Conexant cx23416/cx23415 MPEG encoder/decoder
ivtvfb	Conexant cx23415 framebuffer
mantis	MANTIS based cards
meye	Sony Vaio Picturebook Motion Eye
mxb	Siemens-Nixdorf 'Multimedia eXtension Board'
netup-unidvb	NetUP Universal DVB card
ngene	Micronas nGene
pluto2	Pluto2 cards
saa7134	Philips SAA7134
saa7164	NXP SAA7164
smipcie	SMI PCIe DVBSky cards
solo6x10	Bluecherry / Softlogic 6x10 capture cards (MPEG-4/H.264)
sta2x11_vip	STA2X11 VIP Video For Linux
tw5864	Techwell TW5864 video/audio grabber and encoder
tw686x	Intersil/Techwell TW686x
tw68	Techwell tw68x Video For Linux
zoran	Zoran-36057/36067 JPEG codec

Some of those drivers support multiple devices, as shown at the card lists below:

BTTV cards list

Card number	Card name	PCI subsystem IDs
0	* UNKNOWN/GENERIC *	
1	MIRO PCTV	
2	Hauppauge (bt848)	
3	STB, Gateway P/N 6000699 (bt848)	
4	Intel Create and Share PCI/ Smart Video Recorder III	
5	Diamond DTV2000	
6	AVerMedia TVPhone	

Continued on next page

Table 10 – continued from previous page

Card number	Card name	PCI subsystem IDs
7	MATRIX-Vision MV-Delta	
8	Lifeview FlyVideo II (Bt848) LR26 / MAXI TV Video PCI2 LR26	
9	IMS/IXmicro TurboTV	
10	Hauppauge (bt878)	0070:13eb, 0070:3900, 2636:10b4
11	MIRO PCTV pro	
12	ADS Technologies Channel Surfer TV (bt848)	
13	AVerMedia TVCapture 98	1461:0002, 1461:0004, 1461:0300
14	Aimslab Video Highway Xtreme (VHX)	
15	Zoltrix TV-Max	a1a0:a0fc
16	Prolink Pixelview PlayTV (bt878)	
17	Leadtek WinView 601	
18	AVEC Intercapture	
19	Lifeview FlyVideo II EZ /FlyKit LR38 Bt848 (capture only)	
20	CEI Raffles Card	
21	Lifeview FlyVideo 98/ Lucky Star Image World ConferenceTV LR50	
22	Askey CPH050/ Phoebe Tv Master + FM	14ff:3002
23	Modular Technology MM201/MM202/MM205/MM210/MM215 PCTV, bt878	1457:0101
24	Askey CPH05X/06X (bt878) [many vendors]	144f:3002, 144f:3005, 144f:5000, 14ff:3000
25	Terratec TerraTV+ Version 1.0 (Bt848)/ Terra TValue Version 1.0/ Vobis TV-Boostar	
26	Hauppauge WinCam newer (bt878)	
27	Lifeview FlyVideo 98/ MAXI TV Video PCI2 LR50	
28	Terratec TerraTV+ Version 1.1 (bt878)	153b:1127, 1852:1852
29	Imagination PXC200	1295:200a
30	Lifeview FlyVideo 98 LR50	1f7f:1850
31	Formac iProTV, Formac ProTV I (bt848)	
32	Intel Create and Share PCI/ Smart Video Recorder III	
33	Terratec TerraTValue Version Bt878	153b:1117, 153b:1118, 153b:1119, 153b:111a, 153b:1134, 153b:5018

Continued on next page

Table 10 – continued from previous page

Card number	Card name	PCI subsystem IDs
34	Leadtek WinFast 2000/ WinFast 2000 XP	107d:6606, 107d:6609, 6606:217d, f6ff:fff6
35	Liferview FlyVideo 98 LR50 / Chronos Video Shuttle II	1851:1850, 1851:a050
36	Liferview FlyVideo 98FM LR50 / Typhoon TView TV/FM Tuner	1852:1852
37	Prolink PixelView PlayTV pro	
38	Askey CPH06X TView99	144f:3000, 144f:a005, a04f:a0fc
39	Pinnacle PCTV Studio/Rave	11bd:0012, bd11:1200, bd11:ff00, 11bd:ff12
40	STB TV PCI FM, Gateway P/N 6000704 (bt878), 3Dfx VoodooTV 100	10b4:2636, 10b4:2645, 121a:3060
41	AVerMedia TVPhone 98	1461:0001, 1461:0003
42	ProVideo PV951	aa0c:146c
43	Little OnAir TV	
44	Sigma TVII-FM	
45	MATRIX-Vision MV-Delta 2	
46	Zoltrix Genie TV/FM	15b0:4000, 15b0:400a, 15b0:400d, 15b0:4010, 15b0:4016
47	Terratec TV/Radio+	153b:1123
48	Askey CPH03x/ Dynalink Magic TView	
49	IODATA GV-BCTV3/PCI	10fc:4020
50	Prolink PV-BT878P+4E / PixelView PlayTV PAK / Lenco MXTV-9578 CP	
51	Eagle Wireless Capricorn2 (bt878A)	
52	Pinnacle PCTV Studio Pro	
53	Typhoon TView RDS + FM Stereo / KNC1 TV Station RDS	
54	Liferview FlyVideo 2000 /FlyVideo A2/ Lifetec LT 9415 TV [LR90]	
55	Askey CPH031/ BESTBUY Easy TV	
56	Liferview FlyVideo 98FM LR50	a051:41a0
57	GrandTec 'Grand Video Capture' (Bt848)	4344:4142
58	Askey CPH060/ Phoebe TV Master Only (No FM)	
59	Askey CPH03x TV Capturer	
60	Modular Technology MM100PCTV	
61	AG Electronics GMV1	15cb:0101
62	Askey CPH061/ BESTBUY Easy TV (bt878)	

Continued on next page

Table 10 – continued from previous page

Card number	Card name	PCI subsystem IDs
63	ATI TV-Wonder	1002:0001
64	ATI TV-Wonder VE	1002:0003
65	Lifeview FlyVideo 2000S LR90	
66	Terratec TValueRadio	153b:1135, 153b:ff3b
67	IODATA GV-BCTV4/PCI	10fc:4050
68	3Dfx VoodooTV FM (Euro)	10b4:2637
69	Active Imaging AIMMS	
70	Prolink Pixelview PV-BT878P+ (Rev.4C,8E)	
71	Lifeview FlyVideo 98EZ (capture only) LR51	1851:1851
72	Prolink Pixelview PV-BT878P+9B (PlayTV Pro rev.9B FM+NICAM)	1554:4011
73	Sensoray 311/611	6000:0311, 6000:0611
74	RemoteVision MX (RV605)	
75	Powercolor MTV878/ MTV878R/ MTV878F	
76	Canopus WinDVR PCI (COMPAQ Presario 3524JP, 5112JP)	0e11:0079
77	GrandTec Multi Capture Card (Bt878)	
78	Jetway TV/Capture JW-TV878-FBK, Kworld KW-TV878RF	0a01:17de
79	DSP Design TCVIDEO	
80	Hauppauge WinTV PVR	0070:4500
81	IODATA GV-BCTV5/PCI	10fc:4070, 10fc:d018
82	Osprey 100/150 (878)	0070:ff00
83	Osprey 100/150 (848)	
84	Osprey 101 (848)	
85	Osprey 101/151	
86	Osprey 101/151 w/ svid	
87	Osprey 200/201/250/251	
88	Osprey 200/250	0070:ff01
89	Osprey 210/220/230	
90	Osprey 500	0070:ff02
91	Osprey 540	0070:ff04
92	Osprey 2000	0070:ff03
93	IDS Eagle	
94	Pinnacle PCTV Sat	11bd:001c
95	Formac ProTV II (bt878)	
96	MachTV	
97	Euresys Picolo	

Continued on next page

Table 10 – continued from previous page

Card number	Card name	PCI subsystem IDs
98	ProVideo PV150	aa00:1460, aa01:1461, aa02:1462, aa03:1463, aa04:1464, aa05:1465, aa06:1466, aa07:1467
99	AD-TVK503	
100	Hercules Smart TV Stereo	
101	Pace TV & Radio Card	
102	IVC-200	0000:a155, 0001:a155, 0002:a155, 0003:a155, 0100:a155, 0101:a155, 0102:a155, 0103:a155, 0800:a155, 0801:a155, 0802:a155, 0803:a155
103	Grand X-Guard / Trust 814PCI	0304:0102
104	Nebula Electronics DigiTV	0071:0101
105	ProVideo PV143	aa00:1430, aa00:1431, aa00:1432, aa00:1433, aa03:1433
106	PHYTEC VD-009-X1 VD-011 MiniDIN (bt878)	
107	PHYTEC VD-009-X1 VD-011 Combi (bt878)	
108	PHYTEC VD-009 MiniDIN (bt878)	
109	PHYTEC VD-009 Combi (bt878)	
110	IVC-100	ff00:a132
111	IVC-120G	ff00:a182, ff01:a182, ff02:a182, ff03:a182, ff04:a182, ff05:a182, ff06:a182, ff07:a182, ff08:a182, ff09:a182, ff0a:a182, ff0b:a182, ff0c:a182, ff0d:a182, ff0e:a182, ff0f:a182
112	pcHDTV HD-2000 TV	7063:2000

Continued on next page

Table 10 – continued from previous page

Card number	Card name	PCI subsystem IDs
113	Twinhan DST + clones	11bd:0026, 1822:0001, 270f:fc00, 1822:0026
114	Winfast VC100	107d:6607
115	Teppro TEV-560/InterVision IV-560	
116	SIMUS GVC1100	aa6a:82b2
117	NGS NGSTV+	
118	LMLBT4	
119	Tekram M205 PRO	
120	Conceptronic CONTVFMi	
121	Euresys Picolo Tetra	1805:0105, 1805:0106, 1805:0107, 1805:0108
122	Spirit TV Tuner	
123	AVerMedia AVerTV DVB-T 771	1461:0771
124	AverMedia AverTV DVB-T 761	1461:0761
125	MATRIX Vision Sigma-SQ	
126	MATRIX Vision Sigma-SLC	
127	APAC Viewcomp 878(AMAX)	
128	DViCO FusionHDTV DVB-T Lite	18ac:db10, 18ac:db11
129	V-Gear MyVCD	
130	Super TV Tuner	
131	Tibet Systems 'Progress DVR' CS16	
132	Kodicom 4400R (master)	
133	Kodicom 4400R (slave)	
134	Adlink RTV24	
135	DViCO FusionHDTV 5 Lite	18ac:d500
136	Acorp Y878F	9511:1540
137	Conceptronic CTVFMi v2	036e:109e
138	Prolink Pixelview PV-BT878P+ (Rev.2E)	
139	Prolink PixelView PlayTV MPEG2 PV-M4900	
140	Osprey 440	0070:ff07
141	Asound Skyeye PCTV	
142	Sabrent TV-FM (bttv version)	
143	Hauppauge ImpactVCB (bt878)	0070:13eb
144	MagicTV	
145	SSAI Security Video Interface	4149:5353
146	SSAI Ultrasound Video Interface	414a:5353
147	VoodooTV 200 (USA)	121a:3000
148	DViCO FusionHDTV 2	dbc0:d200
149	Typhoon TV-Tuner PCI (50684)	
150	Geovision GV-600	008a:763c
151	Kozumi KTV-01C	

Continued on next page

Table 10 – continued from previous page

Card number	Card name	PCI subsystem IDs
152	Encore ENL TV-FM-2	1000:1801
153	PHYTEC VD-012 (bt878)	
154	PHYTEC VD-012-X1 (bt878)	
155	PHYTEC VD-012-X2 (bt878)	
156	IVCE-8784	0000:f050, 0001:f050, 0002:f050, 0003:f050
157	Geovision GV-800(S) (master)	800a:763d
158	Geovision GV-800(S) (slave)	800b:763d, 800c:763d, 800d:763d
159	ProVideo PV183	1830:1540, 1831:1540, 1832:1540, 1833:1540, 1834:1540, 1835:1540, 1836:1540, 1837:1540
160	Tongwei Video Technology TD-3116	f200:3116
161	Aposonic W-DVR	0279:0228
162	Adlink MPG24	
163	Bt848 Capture 14MHz	
164	CyberVision CV06 (SV)	
165	Kworld V-Stream Xpert TV PVR878	
166	PCI-8604PW	

CX18 cards list

Those cards are supported by cx18 driver:

- Hauppauge HVR-1600 (ESMT memory)
- Hauppauge HVR-1600 (Samsung memory)
- Compro VideoMate H900
- Yuan MPC718 MiniPCI DVB-T/Analog
- Conexant Raptor PAL/SECAM
- Toshiba Qosmio DVB-T/Analog
- Leadtek WinFast PVR2100
- Leadtek WinFast DVR3100
- GoTView PCI DVD3 Hybrid
- Hauppauge HVR-1600 (s5h1411/tda18271)

cx23885 cards list

Card number	Card name	PCI subsystem IDs
0	UNKNOWN/GENERIC	0070:3400
1	Hauppauge WinTV-HVR1800lp	0070:7600
2	Hauppauge WinTV-HVR1800	0070:7800, 0070:7801, 0070:7809
3	Hauppauge WinTV-HVR1250	0070:7911
4	DViCO FusionHDTV5 Express	18ac:d500
5	Hauppauge WinTV-HVR1500Q	0070:7790, 0070:7797
6	Hauppauge WinTV-HVR1500	0070:7710, 0070:7717
7	Hauppauge WinTV-HVR1200	0070:71d1, 0070:71d3
8	Hauppauge WinTV-HVR1700	0070:8101
9	Hauppauge WinTV-HVR1400	0070:8010
10	DViCO FusionHDTV7 Dual Express	18ac:d618
11	DViCO FusionHDTV DVB-T Dual Express	18ac:db78
12	Leadtek Winfast PxDVR3200 H	107d:6681
13	Compro VideoMate E650F	185b:e800
14	TurboSight TBS 6920	6920:8888
15	TeVii S470	d470:9022
16	DVBWorld DVB-S2 2005	0001:2005
17	NetUP Dual DVB-S2 CI	1b55:2a2c
18	Hauppauge WinTV-HVR1270	0070:2211
19	Hauppauge WinTV-HVR1275	0070:2215, 0070:221d, 0070:22f2
20	Hauppauge WinTV-HVR1255	0070:2251, 0070:22f1
21	Hauppauge WinTV-HVR1210	0070:2291, 0070:2295, 0070:2299, 0070:229d, 0070:22f0, 0070:22f3, 0070:22f4, 0070:22f5
22	Mygica X8506 DMB-TH	14f1:8651
23	Magic-Pro ProHDTV Extreme 2	14f1:8657
24	Hauppauge WinTV-HVR1850	0070:8541
25	Compro VideoMate E800	1858:e800
26	Hauppauge WinTV-HVR1290	0070:8551
27	Mygica X8558 PRO DMB-TH	14f1:8578
28	LEADTEK WinFast PxDTV1200	107d:6f22

Continued on next page

Table 11 – continued from previous page

Card number	Card name	PCI subsystem IDs
29	GoTView X5 3D Hybrid	5654:2390
30	NetUP Dual DVB-T/C-CI RF	1b55:e2e4
31	Leadtek Winfast PxDVR3200 H XC4000	107d:6f39
32	MPX-885	
33	Mygica X8502/X8507 ISDB-T	14f1:8502
34	TerraTec Cinergy T PCIe Dual	153b:117e
35	TeVii S471	d471:9022
36	Hauppauge WinTV-HVR1255	0070:2259
37	Prof Revolution DVB-S2 8000	8000:3034
38	Hauppauge WinTV-HVR4400/HVR5500	0070:c108, 0070:c138, 0070:c1f8
39	AVerTV Hybrid Express Slim HC81R	1461:d939
40	TurboSight TBS 6981	6981:8888
41	TurboSight TBS 6980	6980:8888
42	Leadtek Winfast PXPVR2200	107d:6f21
43	Hauppauge ImpactVCB-e	0070:7133, 0070:7137
44	DViCO FusionHDTV DVB-T Dual Express2	18ac:db98
45	DVBsky T9580	4254:9580
46	DVBsky T980C	4254:980c
47	DVBsky S950C	4254:950c
48	Technotrend TT-budget CT2-4500 CI	13c2:3013
49	DVBsky S950	4254:0950
50	DVBsky S952	4254:0952
51	DVBsky T982	4254:0982
52	Hauppauge WinTV-HVR5525	0070:f038
53	Hauppauge WinTV Starburst	0070:c12a
54	ViewCast 260e	1576:0260
55	ViewCast 460e	1576:0460
56	Hauppauge WinTV-QuadHD-DVB	0070:6a28, 0070:6b28
57	Hauppauge WinTV-QuadHD-ATSC	0070:6a18, 0070:6b18
58	Hauppauge WinTV-HVR-1265(161111)	0070:2a18
59	Hauppauge WinTV-Starburst2	0070:f02a
60	Hauppauge WinTV-QuadHD-DVB(885)	
61	Hauppauge WinTV-QuadHD-ATSC(885)	
62	AVerMedia CE310B	1461:3100

CX88 cards list

Card number	Card name	PCI subsystem IDs
0	UNKNOWN/GENERIC	
1	Hauppauge WinTV 34xxx models	0070:3400, 0070:3401
2	GDI Black Gold	14c7:0106, 14c7:0107
3	PixelView	1554:4811
4	ATI TV Wonder Pro	1002:00f8, 1002:00f9
5	Leadtek Winfast 2000XP Expert	107d:6611, 107d:6613
6	AverTV Studio 303 (M126)	1461:000b
7	MSI TV-@nywhere Master	1462:8606
8	Leadtek Winfast DV2000	107d:6620, 107d:6621
9	Leadtek PVR 2000	107d:663b, 107d:663c, 107d:6632, 107d:6630, 107d:6638, 107d:6631, 107d:6637, 107d:663d
10	IODATA GV-VCP3/PCI	10fc:d003
11	Prolink PlayTV PVR	
12	ASUS PVR-416	1043:4823, 1461:c111
13	MSI TV-@nywhere	
14	KWorld/VStream XPert DVB-T	17de:08a6
15	DViCO FusionHDTV DVB-T1	18ac:db00
16	KWorld LTV883RF	
17	DViCO FusionHDTV 3 Gold-Q	18ac:d810, 18ac:d800
18	Hauppauge Nova-T DVB-T	0070:9002, 0070:9001, 0070:9000
19	Conexant DVB-T reference design	14f1:0187
20	Provideo PV259	1540:2580
21	DViCO FusionHDTV DVB-T Plus	18ac:db10, 18ac:db11
22	pcHDTV HD3000 HDTV	7063:3000
23	digitalnow DNTV Live! DVB-T	17de:a8a6
24	Hauppauge WinTV 28xxx (Roslyn) models	0070:2801
25	Digital-Logic MICROSPACE Entertainment Center (MEC)	14f1:0342
26	IODATA GV/BCTV7E	10fc:d035

Continued on next page

Table 12 – continued from previous page

Card number	Card name	PCI subsystem IDs
27	PixelView PlayTV Ultra Pro (Stereo)	
28	DViCO FusionHDTV 3 Gold-T	18ac:d820
29	ADS Tech Instant TV DVB-T PCI	1421:0334
30	TerraTec Cinergy 1400 DVB-T	153b:1166
31	DViCO FusionHDTV 5 Gold	18ac:d500
32	AverMedia UltraTV Media Center PCI 550	1461:8011
33	Kworld V-Stream Xpert DVD	
34	ATI HDTV Wonder	1002:a101
35	WinFast DTV1000-T	107d:665f
36	AVerTV 303 (M126)	1461:000a
37	Hauppauge Nova-S-Plus DVB-S	0070:9201, 0070:9202
38	Hauppauge Nova-SE2 DVB-S	0070:9200
39	KWorld DVB-S 100	17de:08b2, 1421:0341
40	Hauppauge WinTV-HVR1100 DVB-T/Hybrid	0070:9400, 0070:9402
41	Hauppauge WinTV-HVR1100 DVB-T/Hybrid (Low Profile)	0070:9800, 0070:9802
42	digitalnow DNTV Live! DVB-T Pro	1822:0025, 1822:0019
43	KWorld/VStream XPert DVB-T with cx22702	17de:08a1, 12ab:2300
44	DViCO FusionHDTV DVB-T Dual Digital	18ac:db50, 18ac:db54
45	KWorld HardwareMpegTV XPert	17de:0840, 1421:0305
46	DViCO FusionHDTV DVB-T Hybrid	18ac:db40, 18ac:db44
47	pcHDTV HD5500 HDTV	7063:5500
48	Kworld MCE 200 Deluxe	17de:0841
49	PixelView PlayTV P7000	1554:4813
50	NPG Tech Real TV FM Top 10	14f1:0842
51	WinFast DTV2000 H	107d:665e
52	Geniatech DVB-S	14f1:0084
53	Hauppauge WinTV-HVR3000 TriMode Analog/DVB-S/DVB-T	0070:1404, 0070:1400, 0070:1401, 0070:1402
54	Norwood Micro TV Tuner	
55	Shenzhen Tungsten Ages Tech TE-DTV-250 / Swann OEM	c180:c980
56	Hauppauge WinTV-HVR1300 DVB-T/Hybrid MPEG Encoder	0070:9600, 0070:9601, 0070:9602
57	ADS Tech Instant Video PCI	1421:0390

Continued on next page

Table 12 – continued from previous page

Card number	Card name	PCI subsystem IDs
58	Pinnacle PCTV HD 800i	11bd:0051
59	DViCO FusionHDTV 5 PCI nano	18ac:d530
60	Pinnacle Hybrid PCTV	12ab:1788
61	Leadtek TV2000 XP Global	107d:6f18, 107d:6618, 107d:6619
62	PowerColor RA330	14f1:ea3d
63	Geniatech X8000-MT DVBT	14f1:8852
64	DViCO FusionHDTV DVB-T PRO	18ac:db30
65	DViCO FusionHDTV 7 Gold	18ac:d610
66	Prolink Pixelview MPEG 8000GT	1554:4935
67	Kworld PlusTV HD PCI 120 (ATSC 120)	17de:08c1
68	Hauppauge WinTV-HVR4000 DVB-S/S2/T/Hybrid	0070:6900, 0070:6904, 0070:6902
69	Hauppauge WinTV-HVR4000(Lite) DVB-S/S2	0070:6905, 0070:6906
70	TeVii S460 DVB-S/S2	d460:9022
71	Omicom SS4 DVB-S/S2 PCI	A044:2011
72	TBS 8920 DVB-S/S2	8920:8888
73	TeVii S420 DVB-S	d420:9022
74	Prolink Pixelview Global Extreme	1554:4976
75	PROF 7300 DVB-S/S2	B033:3033
76	SATTRADE ST4200 DVB-S/S2	b200:4200
77	TBS 8910 DVB-S	8910:8888
78	Prof 6200 DVB-S	b022:3022
79	Terratec Cinergy HT PCI MKII	153b:1177
80	Hauppauge WinTV-IR Only	0070:9290
81	Leadtek WinFast DTV1800 Hybrid	107d:6654
82	WinFast DTV2000 H rev. J	107d:6f2b
83	Prof 7301 DVB-S/S2	b034:3034
84	Samsung SMT 7020 DVB-S	18ac:dc00, 18ac:dccd
85	Twinhan VP-1027 DVB-S	1822:0023
86	TeVii S464 DVB-S/S2	d464:9022
87	Leadtek WinFast DTV2000 H PLUS	107d:6f42
88	Leadtek WinFast DTV1800 H (XC4000)	107d:6f38
89	Leadtek TV2000 XP Global (SC4100)	107d:6f36
90	Leadtek TV2000 XP Global (XC4100)	107d:6f43
91	NotOnlyTV LV3H	

IVTV cards list

Card number	Card name	PCI subsystem IDs
0	Hauppauge WinTV PVR-250	IVTV16 104d:813d
1	Hauppauge WinTV PVR-350	IVTV16 104d:813d
2	Hauppauge WinTV PVR-150	IVTV16 104d:813d
3	AVerMedia M179	IVTV15 1461:a3cf, IVTV15 1461:a3ce
4	Yuan MPG600, Kuroutoshikou ITVC16-STVLP	IVTV16 12ab:fff3, IVTV16 12ab:ffff
5	YUAN MPG160, Kuroutoshikou ITVC15-STVLP, I/O Data GV-M2TV/PCI	IVTV15 10fc:40a0
6	Yuan PG600, Diamond PVR-550	IVTV16 ff92:0070, IVTV16 ffab:0600
7	Adaptec VideOh! AVC-2410	IVTV16 9005:0093
8	Adaptec VideOh! AVC-2010	IVTV16 9005:0092
9	Nagase Transgear 5000TV	IVTV16 1461:bfff
10	AOpen VA2000MAX-SNT6	IVTV16 0000:ff5f
11	Yuan MPG600GR, Kuroutoshikou CX23416GYC-STVLP	IVTV16 12ab:0600, IVTV16 fbab:0600, IVTV16 1154:0523
12	I/O Data GV-MVP/RX, GV-MVP/RX2W (dual tuner)	IVTV16 10fc:d01e, IVTV16 10fc:d038, IVTV16 10fc:d039
13	I/O Data GV-MVP/RX2E	IVTV16 10fc:d025
14	GotView PCI DVD	IVTV16 12ab:0600
15	GotView PCI DVD2 Deluxe	IVTV16 ffac:0600
16	Yuan MPC622	IVTV16 ff01:d998
17	Digital Cowboy DCT-MTVP1	IVTV16 1461:bfff

Continued on next page

Table 13 - continued from previous page

Card number	Card name	PCI subsystem IDs
18	Yuan PG600-2, GotView PCI DVD Lite	IVTV16 ffab:0600, IVTV16 ffad:0600
19	Club3D ZAP-TV1x01	IVTV16 ffab:0600
20	AVerTV MCE 116 Plus	IVTV16 1461:c439
21	ASUS Falcon2	IVTV16 1043:4b66, IVTV16 1043:462e, IVTV16 1043:4b2e
22	AVerMedia PVR-150 Plus / AVerTV M113 Partsnic (Daewoo) Tuner	IVTV16 1461:c034, IVTV16 1461:c035
23	AVerMedia EZMaker PCI Deluxe	IVTV16 1461:c03f
24	AVerMedia M104	IVTV16 1461:c136
25	Buffalo PC-MV5L/PCI	IVTV16 1154:052b
26	AVerMedia UltraTV 1500 MCE / AVerTV M113 Philips Tuner	IVTV16 1461:c019, IVTV16 1461:c01b
27	Sony VAIO Giga Pocket (ENX Kikyou)	IVTV16 104d:813d
28	Hauppauge WinTV PVR-350 (V1)	IVTV16 104d:813d
29	Yuan MPG600GR, Kuroutoshikou CX23416GYC-STVLP (no GR)	IVTV16 104d:813d
30	Yuan MPG600GR, Kuroutoshikou CX23416GYC-STVLP (no GR/YCS)	IVTV16 104d:813d

SAA7134 cards list

Card number	Card name	PCI subsystem IDs
0	UNKNOWN/GENERIC	
1	Proteus Pro [philips reference design]	1131:2001, 1131:2001

Continued on next page

Table 14 - continued from previous page

Card number	Card name	PCI subsystem IDs
2	LifeView FlyVIDEO3000	5168:0138, 4e42:0138
3	LifeView/Typhoon FlyVIDEO2000	5168:0138, 4e42:0138
4	EMPRESS	1131:6752
5	SKNet Monster TV	1131:4e85
6	Tevion MD 9717	
7	KNC One TV-Station RDS / Typhoon TV Tuner RDS	1131:fe01, 1894:fe01
8	Terratec Cinergy 400 TV	153b:1142
9	Medion 5044	
10	Kworld/KuroutoShikou SAA7130-TVPCI	
11	Terratec Cinergy 600 TV	153b:1143
12	Medion 7134	16be:0003, 16be:5000
13	Typhoon TV+Radio 90031	
14	ELSA EX-VISION 300TV	1048:226b
15	ELSA EX-VISION 500TV	1048:226a
16	ASUS TV-FM 7134	1043:4842, 1043:4830, 1043:4840
17	AOPEN VA1000 POWER	1131:7133
18	BMK MPEX No Tuner	
19	Compro VideoMate TV	185b:c100
20	Matrox CronosPlus	102B:48d0
21	10MOONS PCI TV CAPTURE CARD	1131:2001
22	AverMedia M156 / Medion 2819	1461:a70b
23	BMK MPEX Tuner	
24	KNC One TV-Station DVR	1894:a006
25	ASUS TV-FM 7133	1043:4843
26	Pinnacle PCTV Stereo (saa7134)	11bd:002b
27	Manli MuchTV M-TV002	
28	Manli MuchTV M-TV001	
29	Nagase Sangyo TransGear 3000TV	1461:050c
30	Elitegroup ECS TVP3XP FM1216 Tuner Card(PAL-BG,FM)	1019:4cb4
31	Elitegroup ECS TVP3XP FM1236 Tuner Card (NTSC,FM)	1019:4cb5
32	AVACS SmartTV	
33	AVerMedia DVD EZMaker	1461:10ff
34	Noval Prime TV 7133	
35	AverMedia AverTV Studio 305	1461:2115
36	UPMOST PURPLE TV	12ab:0800
37	Items MuchTV Plus / IT-005	
38	Terratec Cinergy 200 TV	153b:1152

Continued on next page

Table 14 - continued from previous page

Card number	Card name	PCI subsystem IDs
39	LifeView FlyTV Platinum Mini	5168:0212, 4e42:0212, 5169:1502
40	Compro VideoMate TV PVR/FM	185b:c100
41	Compro VideoMate TV Gold+	185b:c100
42	Sabrent SBT-TVFM (saa7130)	
43	:Zolid Xpert TV7134	
44	Empire PCI TV-Radio LE	
45	Avermedia AVerTV Studio 307	1461:9715
46	AVerMedia Cardbus TV/Radio (E500)	1461:d6ee
47	Terratec Cinergy 400 mobile	153b:1162
48	Terratec Cinergy 600 TV MK3	153b:1158
49	Compro VideoMate Gold+ Pal	185b:c200
50	Pinnacle PCTV 300i DVB-T + PAL	11bd:002d
51	ProVideo PV952	1540:9524
52	AverMedia AverTV/305	1461:2108
53	ASUS TV-FM 7135	1043:4845
54	LifeView FlyTV Platinum FM / Gold	5168:0214, 5168:5214, 1489:0214, 5168:0304
55	LifeView FlyDVB-T DUO / MSI TV@nywhere Duo	5168:0306, 4E42:0306
56	Avermedia AVerTV 307	1461:a70a
57	Avermedia AVerTV GO 007 FM	1461:f31f
58	ADS Tech Instant TV (saa7135)	1421:0350, 1421:0351, 1421:0370, 1421:1370
59	Kworld/Tevion V-Stream Xpert TV PVR7134	
60	LifeView/Typhoon/Genius FlyDVB-T Duo Cardbus	5168:0502, 4e42:0502, 1489:0502
61	Philips TOUGH DVB-T reference design	1131:2004
62	Compro VideoMate TV Gold+II	
63	Kworld Xpert TV PVR7134	
64	FlyTV mini Asus Digimatrix	1043:0210
65	V-Stream Studio TV Terminator	
66	Yuan TUN-900 (saa7135)	
67	Beholder BeholdTV 409 FM	0000:4091
68	GoTView 7135 PCI	5456:7135
69	Philips EUROPA V3 reference design	1131:2004
70	Compro Videomate DVB-T300	185b:c900
71	Compro Videomate DVB-T200	185b:c901
72	RTD Embedded Technologies VFG7350	1435:7350

Continued on next page

Table 14 - continued from previous page

Card number	Card name	PCI subsystem IDs
73	RTD Embedded Technologies VFG7330	1435:7330
74	LifeView FlyTV Platinum Mini2	14c0:1212
75	AVerMedia AVerTVHD MCE A180	1461:1044
76	SKNet MonsterTV Mobile	1131:4ee9
77	Pinnacle PCTV 40i/50i/110i (saa7133)	11bd:002e
78	ASUSTeK P7131 Dual	1043:4862
79	Sedna/MuchTV PC TV Cardbus TV/Radio (ITO25 Rev:2B)	
80	ASUS Digimatrix TV	1043:0210
81	Philips Tiger reference design	1131:2018
82	MSI TV@Anywhere plus	1462:6231, 1462:8624
83	Terratec Cinergy 250 PCI TV	153b:1160
84	LifeView FlyDVB Trio	5168:0319
85	AverTV DVB-T 777	1461:2c05, 1461:2c05
86	LifeView FlyDVB-T / Genius VideoWonder DVB-T	5168:0301, 1489:0301
87	ADS Instant TV Duo Cardbus PTV331	0331:1421
88	Tevion/KWorld DVB-T 220RF	17de:7201
89	ELSA EX-VISION 700TV	1048:226c
90	Kworld ATSC110/115	17de:7350, 17de:7352
91	AVerMedia A169 B	1461:7360
92	AVerMedia A169 B1	1461:6360
93	Medion 7134 Bridge #2	16be:0005
94	LifeView FlyDVB-T Hybrid Cardbus/MSI TV @nywhere A/D NB	5168:3306, 5168:3502, 5168:3307, 4e42:3502
95	LifeView FlyVIDEO3000 (NTSC)	5169:0138
96	Medion Md8800 Quadro	16be:0007, 16be:0008, 16be:000d
97	LifeView FlyDVB-S /Acorp TV134DS	5168:0300, 4e42:0300
98	Proteus Pro 2309	0919:2003
99	AVerMedia TV Hybrid A16AR	1461:2c00
100	Asus Europa2 OEM	1043:4860
101	Pinnacle PCTV 310i	11bd:002f
102	Avermedia AVerTV Studio 507	1461:9715
103	Compro Videomate DVB-T200A	

Continued on next page

Table 14 - continued from previous page

Card number	Card name	PCI subsystem IDs
104	Hauppauge WinTV-HVR1110 DVB-T/Hybrid	0070:6700, 0070:6701, 0070:6702, 0070:6703, 0070:6704, 0070:6705
105	Terratec Cinergy HT PCMCIA	153b:1172
106	Encore ENLTV	1131:2342, 1131:2341, 3016:2344
107	Encore ENLTV-FM	1131:230f
108	Terratec Cinergy HT PCI	153b:1175
109	Philips Tiger - S Reference design	
110	Avermedia M102	1461:f31e
111	ASUS P7131 4871	1043:4871
112	ASUSTeK P7131 Hybrid	1043:4876
113	Elitegroup ECS TVP3XP FM1246 Tuner Card (PAL,FM)	1019:4cb6
114	KWorld DVB-T 210	17de:7250
115	Sabrent PCMCIA TV-PCB05	0919:2003
116	10MOONS TM300 TV Card	1131:2304
117	Avermedia Super 007	1461:f01d
118	Beholder BeholdTV 401	0000:4016
119	Beholder BeholdTV 403	0000:4036
120	Beholder BeholdTV 403 FM	0000:4037
121	Beholder BeholdTV 405	0000:4050
122	Beholder BeholdTV 405 FM	0000:4051
123	Beholder BeholdTV 407	0000:4070
124	Beholder BeholdTV 407 FM	0000:4071
125	Beholder BeholdTV 409	0000:4090
126	Beholder BeholdTV 505 FM	5ace:5050
127	Beholder BeholdTV 507 FM / BeholdTV 509 FM	5ace:5070, 5ace:5090
128	Beholder BeholdTV Columbus TV/FM	0000:5201
129	Beholder BeholdTV 607 FM	5ace:6070
130	Beholder BeholdTV M6	5ace:6190
131	Twinhan Hybrid DTV-DVB 3056 PCI	1822:0022
132	Genius TVGO AM11MCE	
133	NXP Snake DVB-S reference design	
134	Medion/Creatix CTX953 Hybrid	16be:0010
135	MSI TV@nywhere A/D v1.1	1462:8625
136	AVerMedia Cardbus TV/Radio (E506R)	1461:f436
137	AVerMedia Hybrid TV/Radio (A16D)	1461:f936
138	Avermedia M115	1461:a836
139	Compro VideoMate T750	185b:c900
140	Avermedia DVB-S Pro A700	1461:a7a1

Continued on next page

Table 14 - continued from previous page

Card number	Card name	PCI subsystem IDs
141	Avermedia DVB-S Hybrid+FM A700	1461:a7a2
142	Beholder BeholdTV H6	5ace:6290
143	Beholder BeholdTV M63	5ace:6191
144	Beholder BeholdTV M6 Extra	5ace:6193
145	AVerMedia MiniPCI DVB-T Hybrid M103	1461:f636, 1461:f736
146	ASUSTeK P7131 Analog	
147	Asus Tiger 3in1	1043:4878
148	Encore ENLTV-FM v5.3	1a7f:2008
149	Avermedia PCI pure analog (M135A)	1461:f11d
150	Zogis Real Angel 220	
151	ADS Tech Instant HDTV	1421:0380
152	Asus Tiger Rev:1.00	1043:4857
153	Kworld Plus TV Analog Lite PCI	17de:7128
154	Avermedia AVerTV GO 007 FM Plus	1461:f31d
155	Hauppauge WinTV-HVR1150 ATSC/QAM-Hybrid	0070:6706, 0070:6708
156	Hauppauge WinTV-HVR1120 DVB-T/Hybrid	0070:6707, 0070:6709, 0070:670a
157	Avermedia AVerTV Studio 507UA	1461:a11b
158	AVerMedia Cardbus TV/Radio (E501R)	1461:b7e9
159	Beholder BeholdTV 505 RDS	0000:505B
160	Beholder BeholdTV 507 RDS	0000:5071
161	Beholder BeholdTV 507 RDS	0000:507B
162	Beholder BeholdTV 607 FM	5ace:6071
163	Beholder BeholdTV 609 FM	5ace:6090
164	Beholder BeholdTV 609 FM	5ace:6091
165	Beholder BeholdTV 607 RDS	5ace:6072
166	Beholder BeholdTV 607 RDS	5ace:6073
167	Beholder BeholdTV 609 RDS	5ace:6092
168	Beholder BeholdTV 609 RDS	5ace:6093
169	Compro VideoMate S350/S300	185b:c900
170	AverMedia AVerTV Studio 505	1461:a115
171	Beholder BeholdTV X7	5ace:7595
172	RoverMedia TV Link Pro FM	19d1:0138
173	Zolid Hybrid TV Tuner PCI	1131:2004
174	Asus Europa Hybrid OEM	1043:4847
175	Leadtek Winfast DTV1000S	107d:6655
176	Beholder BeholdTV 505 RDS	0000:5051
177	Hawell HW-404M7	
178	Beholder BeholdTV H7	5ace:7190
179	Beholder BeholdTV A7	5ace:7090
180	Avermedia PCI M733A	1461:4155, 1461:4255
181	TechoTrend TT-budget T-3000	13c2:2804

Continued on next page

Table 14 – continued from previous page

Card number	Card name	PCI subsystem IDs
182	Kworld PCI SBTVD/ISDB-T Full-Seg Hybrid	17de:b136
183	Compro VideoMate Vista M1F	185b:c900
184	Encore ENLTV-FM 3	1a7f:2108
185	MagicPro ProHDTV Pro2 DMB-TH/Hybrid	17de:d136
186	Beholder BeholdTV 501	5ace:5010
187	Beholder BeholdTV 503 FM	5ace:5030
188	Sensoray 811/911	6000:0811, 6000:0911
189	Kworld PC150-U	17de:a134
190	Asus My Cinema PS3-100	1043:48cd
191	Hawell HW-9004V1	
192	AverMedia AverTV Satellite Hybrid+FM A706	1461:2055
193	WIS Voyager or compatible	1905:7007
194	AverMedia AverTV/505	1461:a10a
195	Leadtek Winfast TV2100 FM	107d:6f3a
196	SnaZio* TVPVR PRO	1779:13cf

SAA7164 cards list

Card number	Card name	PCI subsystem IDs
0	Unknown	
1	Generic Rev2	
2	Generic Rev3	
3	Hauppauge WinTV-HVR2250	0070:8880, 0070:8810
4	Hauppauge WinTV-HVR2200	0070:8980
5	Hauppauge WinTV-HVR2200	0070:8900
6	Hauppauge WinTV-HVR2200	0070:8901
7	Hauppauge WinTV-HVR2250	0070:8891, 0070:8851
8	Hauppauge WinTV-HVR2250	0070:88A1
9	Hauppauge WinTV-HVR2200	0070:8940
10	Hauppauge WinTV-HVR2200	0070:8953
11	Hauppauge WinTV-HVR2255(proto)	0070:f111
12	Hauppauge WinTV-HVR2255	0070:f111
13	Hauppauge WinTV-HVR2205	0070:f123, 0070:f120

Zoran cards list

Card number	Card name	PCI subsystem IDs
0	DC10(old)	<any>
1	DC10(new)	<any>
2	DC10_PLUS	1031:7efe
3	DC30	<any>
4	DC30_PLUS	1031:d801
5	LML33	<any>
6	LML33R10	12f8:8a02
7	Buz	13ca:4231
8	6-Eyes	<any>

1.1.5.3 Platform drivers

There are several drivers that are focused on providing support for functionality that are already included at the main board, and don't use neither USB nor PCI bus. Those drivers are called platform drivers, and are very popular on embedded devices.

The current supported of platform drivers (not including staging drivers) are listed below

Driver	Name
am437x-vpfe	TI AM437x VPFE
aspeed-video	Aspeed AST2400 and AST2500
atmel-isc	ATMEL Image Sensor Controller (ISC)
atmel-isi	ATMEL Image Sensor Interface (ISI)
c8sectpfe	SDR platform devices
c8sectpfe	SDR platform devices
cafe_ccic	Marvell 88ALP01 (Cafe) CMOS Camera Controller
cdns-csi2rx	Cadence MIPI-CSI2 RX Controller
cdns-csi2tx	Cadence MIPI-CSI2 TX Controller
coda-vpu	Chips&Media Coda multi-standard codec IP
dm355_ccdc	TI DM355 CCDC video capture
dm644x_ccdc	TI DM6446 CCDC video capture
exynos-fimc-is	EXYNOS4x12 FIMC-IS (Imaging Subsystem)
exynos-fimc-lite	EXYNOS FIMC-LITE camera interface
exynos-gsc	Samsung Exynos G-Scaler
exy	Samsung S5P/EXYNOS4 SoC series Camera Subsystem
fsl-viu	Freescale VIU
imx-pxp	i.MX Pixel Pipeline (PXP)
isdf	TI DM365 ISIF video capture
mmp_camera	Marvell Armada 610 integrated camera controller
mtk_jpeg	Mediatek JPEG Codec
mtk-mdp	Mediatek MDP
mtk-vcodec-dec	Mediatek Video Codec
mtk-vpu	Mediatek Video Processor Unit

Continued on next page

Table 15 - continued from previous page

Driver	Name
mx2_emmaprp	MX2 eMMa-PrP
omap3-isp	OMAP 3 Camera
omap-vout	OMAP2/OMAP3 V4L2-Display
pxa_camera	PXA27x Quick Capture Interface
qcom-camss	Qualcomm V4L2 Camera Subsystem
rcar-csi2	R-Car MIPI CSI-2 Receiver
rcar_drif	Renesas Digital Radio Interface (DRIF)
rcar-fcp	Renesas Frame Compression Processor
rcar_fdp1	Renesas Fine Display Processor
rcar_jpu	Renesas JPEG Processing Unit
rcar-vin	R-Car Video Input (VIN)
renesas-ceu	Renesas Capture Engine Unit (CEU)
rockchip-rga	Rockchip Raster 2d Graphic Acceleration Unit
s3c-camif	Samsung S3C24XX/S3C64XX SoC Camera Interface
s5p-csis	S5P/EXYNOS MIPI-CSI2 receiver (MIPI-CSIS)
s5p-fimc	S5P/EXYNOS4 FIMC/CAMIF camera interface
s5p-g2d	Samsung S5P and EXYNOS4 G2D 2d graphics accelerator
s5p-jpeg	Samsung S5P/Exynos3250/Exynos4 JPEG codec
s5p-mfc	Samsung S5P MFC Video Codec
sh_veu	SuperH VEU mem2mem video processing
sh_vou	SuperH VOU video output
stm32-dcml	STM32 Digital Camera Memory Interface (DCMI)
stm32-dma2d	STM32 Chrom-Art Accelerator Unit
sun4i-csi	Allwinner A10 CMOS Sensor Interface Support
sun6i-csi	Allwinner V3s Camera Sensor Interface
sun8i-di	Allwinner Deinterlace
sun8i-rotate	Allwinner DE2 rotation
ti-cal	TI Memory-to-memory multimedia devices
ti-csc	TI DVB platform devices
ti-vpe	TI VPE (Video Processing Engine)
venus-enc	Qualcomm Venus V4L2 encoder/decoder
via-camera	VIAFB camera controller
video-mux	Video Multiplexer
vpif_display	TI DaVinci VPIF V4L2-Display
vpif_capture	TI DaVinci VPIF video capture
vpss	TI DaVinci VPBE V4L2-Display
vsp1	Renesas VSP1 Video Processing Engine
xilinx-tpg	Xilinx Video Test Pattern Generator
xilinx-video	Xilinx Video IP (EXPERIMENTAL)
xilinx-vtc	Xilinx Video Timing Controller

MMC/SDIO DVB adapters

Driver	Name
smssdio	Siano SMS1xxx based MDTV via SDIO interface

1.1.5.4 Radio drivers

There is also support for pure AM/FM radio, and even for some FM radio transmitters:

Driver	Name
si4713	Silicon Labs Si4713 FM Radio Transmitter
radio-aztech	Aztech/Packard Bell Radio
radio-cadet	ADS Cadet AM/FM Tuner
radio-gemtek	GemTek Radio card (or compatible)
radio-maxiradio	Guillemot MAXI Radio FM 2000 radio
radio-miropcm20	miroSOUND PCM20 radio
radio-aimslab	AIMSlab RadioTrack (aka RadioReveal)
radio-rtrack2	AIMSlab RadioTrack II
saa7706h	SAA7706H Car Radio DSP
radio-sf16fmi	SF16-FMI/SF16-FMP/SF16-FMD Radio
radio-sf16fmr2	SF16-FMR2/SF16-FMD2 Radio
radio-shark	Griffin radioSHARK USB radio receiver
shark2	Griffin radioSHARK2 USB radio receiver
radio-si470x-common	Silicon Labs Si470x FM Radio Receiver
radio-si476x	Silicon Laboratories Si476x I2C FM Radio
radio-tea5764	TEA5764 I2C FM radio
tef6862	TEF6862 Car Radio Enhanced Selectivity Tuner
radio-terratec	TerraTec ActiveRadio ISA Standalone
radio-timb	Enable the Timberdale radio driver
radio-trust	Trust FM radio card
radio-typhoon	Typhoon Radio (a.k.a. EcoRadio)
radio-wl1273	Texas Instruments WL1273 I2C FM Radio
fm_drv	ISA radio devices
fm_drv	ISA radio devices
radio-zoltrix	Zoltrix Radio
dsbr100	D-Link/GemTek USB FM radio
radio-keene	Keene FM Transmitter USB
radio-ma901	Masterkit MA901 USB FM radio
radio-mr800	AverMedia MR 800 USB FM radio
radio-raremono	Thanko's Raremono AM/FM/SW radio
radio-si470x-usb	Silicon Labs Si470x FM Radio Receiver support with USB
radio-usb-si4713	Silicon Labs Si4713 FM Radio Transmitter support with USB

1.1.5.5 I²C drivers

The I²C (Inter-Integrated Circuit) bus is a three-wires bus used internally at the media cards for communication between different chips. While the bus is not visible to the Linux Kernel, drivers need to send and receive commands via the bus. The Linux Kernel driver abstraction has support to implement different drivers for each component inside an I²C bus, as if the bus were visible to the main system board.

One of the problems with I²C devices is that sometimes the same device may work with different I²C hardware. This is common, for example, on devices that comes with a tuner for North America market, and another one for Europe. Some drivers have a `tuner=modprobe` parameter to allow using a different tuner number in order to address such issue.

The current supported of I²C drivers (not including staging drivers) are listed below.

Audio decoders, processors and mixers

Driver	Name
cs3308	Cirrus Logic CS3308 audio ADC
cs5345	Cirrus Logic CS5345 audio ADC
cs53l32a	Cirrus Logic CS53L32A audio ADC
msp3400	Micronas MSP34xx audio decoders
sony-btf-mpx	Sony BTF's internal MPX
tda1997x	NXP TDA1997x HDMI receiver
tda7432	Philips TDA7432 audio processor
tda9840	Philips TDA9840 audio processor
tea6415c	Philips TEA6415C audio processor
tea6420	Philips TEA6420 audio processor
tlv320aic23b	Texas Instruments TLV320AIC23B audio codec
tvaudio	Simple audio decoder chips
uda1342	Philips UDA1342 audio codec
vp27smpx	Panasonic VP27's internal MPX
wm8739	Wolfson Microelectronics WM8739 stereo audio ADC
wm8775	Wolfson Microelectronics WM8775 audio ADC with input mixer

Audio/Video compression chips

Driver	Name
saa6752hs	Philips SAA6752HS MPEG-2 Audio/Video Encoder

Camera sensor devices

Driver	Name
ccs	MIPI CCS compliant camera sensors (also SMIA++ and SMIA)
et8ek8	ET8EK8 camera sensor
hi556	Hynix Hi-556 sensor
hi846	Hynix Hi-846 sensor
imx208	Sony IMX208 sensor
imx214	Sony IMX214 sensor
imx219	Sony IMX219 sensor
imx258	Sony IMX258 sensor
imx274	Sony IMX274 sensor
imx290	Sony IMX290 sensor
imx319	Sony IMX319 sensor
imx334	Sony IMX334 sensor
imx355	Sony IMX355 sensor
imx412	Sony IMX412 sensor
m5mols	Fujitsu M-5MOLS 8MP sensor
mt9m001	mt9m001
mt9m032	MT9M032 camera sensor
mt9m111	mt9m111, mt9m112 and mt9m131
mt9p031	Aptina MT9P031
mt9t001	Aptina MT9T001
mt9t112	Aptina MT9T111/MT9T112
mt9v011	Micron mt9v011 sensor
mt9v032	Micron MT9V032 sensor
mt9v111	Aptina MT9V111 sensor
noon010pc30	Siliconfile NOON010PC30 sensor
ov13858	OmniVision OV13858 sensor
ov13b10	OmniVision OV13B10 sensor
ov2640	OmniVision OV2640 sensor
ov2659	OmniVision OV2659 sensor
ov2680	OmniVision OV2680 sensor
ov2685	OmniVision OV2685 sensor
ov5640	OmniVision OV5640 sensor
ov5645	OmniVision OV5645 sensor
ov5647	OmniVision OV5647 sensor
ov5670	OmniVision OV5670 sensor
ov5675	OmniVision OV5675 sensor
ov5695	OmniVision OV5695 sensor
ov6650	OmniVision OV6650 sensor

Continued on next page

Table 17 - continued from previous page

Driver	Name
ov7251	OmniVision OV7251 sensor
ov7640	OmniVision OV7640 sensor
ov7670	OmniVision OV7670 sensor
ov772x	OmniVision OV772x sensor
ov7740	OmniVision OV7740 sensor
ov8856	OmniVision OV8856 sensor
ov9640	OmniVision OV9640 sensor
ov9650	OmniVision OV9650/OV9652 sensor
rj54n1cb0c	Sharp RJ54N1CB0C sensor
s5c73m3	Samsung S5C73M3 sensor
s5k4ecgx	Samsung S5K4ECGX sensor
s5k5baf	Samsung S5K5BAF sensor
s5k6a3	Samsung S5K6A3 sensor
s5k6aa	Samsung S5K6AAFX sensor
sr030pc30	Siliconfile SR030PC30 sensor
vs6624	ST VS6624 sensor

Flash devices

Driver	Name
adp1653	ADP1653 flash
lm3560	LM3560 dual flash driver
lm3646	LM3646 dual flash driver

IR I2C driver

Driver	Name
ir-kbd-i2c	I2C module for IR

Lens drivers

Driver	Name
ad5820	AD5820 lens voice coil
ak7375	AK7375 lens voice coil
dw9714	DW9714 lens voice coil
dw9768	DW9768 lens voice coil
dw9807-vcml	DW9807 lens voice coil

Miscellaneous helper chips

Driver	Name
video-i2c	I2C transport video
m52790	Mitsubishi M52790 A/V switch
st-mipid02	STMicroelectronics MIPID02 CSI-2 to PARALLEL bridge
ths7303	THS7303/53 Video Amplifier

RDS decoders

Driver	Name
saa6588	SAA6588 Radio Chip RDS decoder

SDR tuner chips

Driver	Name
max2175	Maxim 2175 RF to Bits tuner

Video and audio decoders

Driver	Name
cx25840	Conexant CX2584x audio/video decoders
saa717x	Philips SAA7171/3/4 audio/video decoders

Video decoders

Driver	Name
adv7180	Analog Devices ADV7180 decoder
adv7183	Analog Devices ADV7183 decoder
adv748x	Analog Devices ADV748x decoder
adv7604	Analog Devices ADV7604 decoder
adv7842	Analog Devices ADV7842 decoder
bt819	BT819A VideoStream decoder
bt856	BT856 VideoStream decoder
bt866	BT866 VideoStream decoder
ks0127	KS0127 video decoder
ml86v7667	OKI ML86V7667 video decoder
saa7110	Philips SAA7110 video decoder
saa7115	Philips SAA7111/3/4/5 video decoders
tc358743	Toshiba TC358743 decoder
tvp514x	Texas Instruments TVP514x video decoder
tvp5150	Texas Instruments TVP5150 video decoder
tvp7002	Texas Instruments TVP7002 video decoder
tw2804	Techwell TW2804 multiple video decoder
tw9903	Techwell TW9903 video decoder
tw9906	Techwell TW9906 video decoder
tw9910	Techwell TW9910 video decoder
vpx3220	vpx3220a, vpx3216b & vpx3214c video decoders

Video encoders

Driver	Name
ad9389b	Analog Devices AD9389B encoder
adv7170	Analog Devices ADV7170 video encoder
adv7175	Analog Devices ADV7175 video encoder
adv7343	ADV7343 video encoder
adv7393	ADV7393 video encoder
adv7511-v4l2	Analog Devices ADV7511 encoder
ak881x	AK8813/AK8814 video encoders
saa7127	Philips SAA7127/9 digital video encoders
saa7185	Philips SAA7185 video encoder
ths8200	Texas Instruments THS8200 video encoder

Video improvement chips

Driver	Name
upd64031a	NEC Electronics uPD64031A Ghost Reduction
upd64083	NEC Electronics uPD64083 3-Dimensional Y/C separation

Tuner drivers

Driver	Name
e4000	Elonics E4000 silicon tuner
fc0011	Fitipower FC0011 silicon tuner
fc0012	Fitipower FC0012 silicon tuner
fc0013	Fitipower FC0013 silicon tuner
fc2580	FCI FC2580 silicon tuner
it913x	ITE Tech IT913x silicon tuner
m88rs6000t	Montage M88RS6000 internal tuner
max2165	Maxim MAX2165 silicon tuner
mc44s803	Freescale MC44S803 Low Power CMOS Broadband tuners
msi001	Mirics MSi001
mt2060	Microtune MT2060 silicon IF tuner
mt2063	Microtune MT2063 silicon IF tuner
mt20xx	Microtune 2032 / 2050 tuners
mt2131	Microtune MT2131 silicon tuner
mt2266	Microtune MT2266 silicon tuner
mxl301rf	MaxLinear MxL301RF tuner
mxl5005s	MaxLinear MSL5005S silicon tuner
mxl5007t	MaxLinear MxL5007T silicon tuner
qm1d1b0004	Sharp QM1D1B0004 tuner
qm1d1c0042	Sharp QM1D1C0042 tuner
qt1010	Quantek QT1010 silicon tuner
r820t	Rafael Micro R820T silicon tuner
si2157	Silicon Labs Si2157 silicon tuner
tuner-types	Simple tuner support
tda18212	NXP TDA18212 silicon tuner
tda18218	NXP TDA18218 silicon tuner
tda18250	NXP TDA18250 silicon tuner
tda18271	NXP TDA18271 silicon tuner
tda827x	Philips TDA827X silicon tuner
tda8290	TDA 8290/8295 + 8275(a)/18271 tuner combo
tda9887	TDA 9885/6/7 analog IF demodulator
tea5761	TEA 5761 radio tuner
tea5767	TEA 5767 radio tuner
tua9001	Infineon TUA9001 silicon tuner
tuner-xc2028	XCeive xc2028/xc3028 tuners
xc4000	Xceive XC4000 silicon tuner
xc5000	Xceive XC5000 silicon tuner

Tuner cards list

Tuner number	Card name
0	Temec PAL (4002 FH5)
1	Philips PAL_I (FI1246 and compatibles)
2	Philips NTSC (FI1236,FM1236 and compatibles)
3	Philips (SECAM+PAL_BG) (FI1216MF, FM1216MF, FR1216MF)
4	NoTuner
5	Philips PAL_BG (FI1216 and compatibles)
6	Temec NTSC (4032 FY5)
7	Temec PAL_I (4062 FY5)
8	Temec NTSC (4036 FY5)
9	Alps HSBH1
10	Alps TSBE1
11	Alps TSBB5
12	Alps TSBE5
13	Alps TSBC5
14	Temec PAL_BG (4006FH5)
15	Alps TSCH6
16	Temec PAL_DK (4016 FY5)
17	Philips NTSC_M (MK2)
18	Temec PAL_I (4066 FY5)
19	Temec PAL* auto (4006 FN5)
20	Temec PAL_BG (4009 FR5) or PAL_I (4069 FR5)
21	Temec NTSC (4039 FR5)
22	Temec PAL/SECAM multi (4046 FM5)
23	Philips PAL_DK (FI1256 and compatibles)
24	Philips PAL/SECAM multi (FQ1216ME)
25	LG PAL_I+FM (TAPC-I001D)
26	LG PAL_I (TAPC-I701D)
27	LG NTSC+FM (TPI8NSR01F)
28	LG PAL_BG+FM (TPI8PSB01D)
29	LG PAL_BG (TPI8PSB11D)
30	Temec PAL* auto + FM (4009 FN5)
31	SHARP NTSC_JP (2U5JF5540)
32	Samsung PAL TCPM9091PD27
33	MT20xx universal
34	Temec PAL_BG (4106 FH5)
35	Temec PAL_DK/SECAM_L (4012 FY5)
36	Temec NTSC (4136 FY5)
37	LG PAL (newer TAPC series)
38	Philips PAL/SECAM multi (FM1216ME MK3)
39	LG NTSC (newer TAPC series)
40	HITACHI V7-J180AT
41	Philips PAL_MK (FI1216 MK)
42	Philips FCV1236D ATSC/NTSC dual in
43	Philips NTSC MK3 (FM1236MK3 or FM1236/F)
44	Philips 4 in 1 (ATI TV Wonder Pro/Conexant)

Continued on next page

Table 19 - continued from previous page

Tuner number	Card name
45	Microtune 4049 FM5
46	Panasonic VP27s/ENGE4324D
47	LG NTSC (TAPE series)
48	Tenna TNF 8831 BGFF)
49	Microtune 4042 FI5 ATSC/NTSC dual in
50	TCL 2002N
51	Philips PAL/SECAM_D (FM 1256 I-H3)
52	Thomson DTT 7610 (ATSC/NTSC)
53	Philips FQ1286
54	Philips/NXP TDA 8290/8295 + 8275/8275A/18271
55	TCL 2002MB
56	Philips PAL/SECAM multi (FQ1216AME MK4)
57	Philips FQ1236A MK4
58	Ymec TVision TVF-8531MF/8831MF/8731MF
59	Ymec TVision TVF-5533MF
60	Thomson DTT 761X (ATSC/NTSC)
61	Tena TNF9533-D/IF/TNF9533-B/DF
62	Philips TEA5767HN FM Radio
63	Philips FMD1216ME MK3 Hybrid Tuner
64	LG TDVS-H06xF
65	Ymec TVF66T5-B/DFF
66	LG TALN series
67	Philips TD1316 Hybrid Tuner
68	Philips TUV1236D ATSC/NTSC dual in
69	Tena TNF 5335 and similar models
70	Samsung TCPN 2121P30A
71	Xceive xc2028/xc3028 tuner
72	Thomson FE6600
73	Samsung TCPG 6121P30A
75	Philips TEA5761 FM Radio
76	Xceive 5000 tuner
77	TCL tuner MF02GIP-5N-E
78	Philips FMD1216MEX MK3 Hybrid Tuner
79	Philips PAL/SECAM multi (FM1216 MK5)
80	Philips FQ1216LME MK3 PAL/SECAM w/active loopthrough
81	Partsnic (Daewoo) PTI-5NF05
82	Philips CU1216L
83	NXP TDA18271
84	Sony BTF-Pxn01Z
85	Philips FQ1236 MK5
86	Tena TNF5337 MFD
87	Xceive 4000 tuner
88	Xceive 5000C tuner
89	Sony BTF-PG472Z PAL/SECAM
90	Sony BTF-PK467Z NTSC-M-JP
91	Sony BTF-PB463Z NTSC-M

Frontend drivers

Note:

- 1) There is no guarantee that every frontend driver works out of the box with every card, because of different wiring.
 - 2) The demodulator chips can be used with a variety of tuner/PLL chips, and not all combinations are supported. Often the demodulator and tuner/PLL chip are inside a metal box for shielding, and the whole metal box has its own part number.
-

Common Interface (EN50221) controller drivers

Driver	Name
cxd2099	Sony CXD2099AR Common Interface driver
sp2	CIMaX SP2

ATSC (North American/Korean Terrestrial/Cable DTV) frontends

Driver	Name
au8522_dig	Auvitek AU8522 based DTV demod
au8522_decoder	Auvitek AU8522 based ATV demod
bcm3510	Broadcom BCM3510
lg2160	LG Electronics LG216x based
lgdt3305	LG Electronics LGDT3304 and LGDT3305 based
lgdt3306a	LG Electronics LGDT3306A based
lgdt330x	LG Electronics LGDT3302/LGDT3303 based
nxt200x	NxtWave Communications NXT2002/NXT2004 based
or51132	Oren OR51132 based
or51211	Oren OR51211 based
s5h1409	Samsung S5H1409 based
s5h1411	Samsung S5H1411 based

DVB-C (cable) frontends

Driver	Name
stv0297	ST STV0297 based
tda10021	Philips TDA10021 based
tda10023	Philips TDA10023 based
ves1820	VLSI VES1820 based

DVB-S (satellite) frontends

Driver	Name
cx24110	Conexant CX24110 based
cx24116	Conexant CX24116 based
cx24117	Conexant CX24117 based
cx24120	Conexant CX24120 based
cx24123	Conexant CX24123 based
ds3000	Montage Technology DS3000 based
mb86a16	Fujitsu MB86A16 based
mt312	Zarlink VP310/MT312/ZL10313 based
s5h1420	Samsung S5H1420 based
si21xx	Silicon Labs SI21XX based
stb6000	ST STB6000 silicon tuner
stv0288	ST STV0288 based
stv0299	ST STV0299 based
stv0900	ST STV0900 based
stv6110	ST STV6110 silicon tuner
tda10071	NXP TDA10071
tda10086	Philips TDA10086 based
tda8083	Philips TDA8083 based
tda8261	Philips TDA8261 based
tda826x	Philips TDA826X silicon tuner
ts2020	Montage Technology TS2020 based tuners
tua6100	Infineon TUA6100 PLL
cx24113	Conexant CX24113/CX24128 tuner for DVB-S/DSS
itd1000	Integrant ITD1000 Zero IF tuner for DVB-S/DSS
ves1x93	VLSI VES1893 or VES1993 based
zl10036	Zarlink ZL10036 silicon tuner
zl10039	Zarlink ZL10039 silicon tuner

ISDB-S (satellite) & ISDB-T (terrestrial) frontends

Driver	Name
mn88443x	Socionext MN88443x
tc90522	Toshiba TC90522

ISDB-T (terrestrial) frontends

Driver	Name
dib8000	DiBcom 8000MB/MC
mb86a20s	Fujitsu mb86a20s
s921	Sharp S921 frontend

Multistandard (cable + terrestrial) frontends

Driver	Name
drxk	Micronas DRXK based
mn88472	Panasonic MN88472
mn88473	Panasonic MN88473
si2165	Silicon Labs si2165 based
tda18271c2dd	NXP TDA18271C2 silicon tuner

Multistandard (satellite) frontends

Driver	Name
m88ds3103	Montage Technology M88DS3103
mxl5xx	MaxLinear MxL5xx based tuner-demodulators
stb0899	STB0899 based
stb6100	STB6100 based tuners
stv090x	STV0900/STV0903(A/B) based
stv0910	STV0910 based
stv6110x	STV6110/(A) based tuners
stv6111	STV6111 based tuners

SEC control devices for DVB-S

Driver	Name
a8293	Allegro A8293
af9033	Afatech AF9033 DVB-T demodulator
ascot2e	Sony Ascot2E tuner
atbm8830	AltoBeam ATBM8830/8831 DMB-TH demodulator
drx39xyj	Micronas DRX-J demodulator
helene	Sony HELENE Sat/Ter tuner (CXD2858ER)
horus3a	Sony Horus3A tuner
isl6405	ISL6405 SEC controller
isl6421	ISL6421 SEC controller
isl6423	ISL6423 SEC controller
ix2505v	Sharp IX2505V silicon tuner
lgs8gl5	Silicon Legend LGS-8GL5 demodulator (OFDM)
lgs8gxx	Legend Silicon LGS8913/LGS8GL5/LGS8GXX DMB-TH demodulator
lnbh25	LNBH25 SEC controller
lnbh29	LNBH29 SEC controller
lnbp21	LNBP21/LNBH24 SEC controllers
lnbp22	LNBP22 SEC controllers
m88rs2000	M88RS2000 DVB-S demodulator and tuner
tda665x	TDA665x tuner

Tools to develop new frontends

Driver	Name
dvb_dummy_fe	Dummy frontend driver

1.1.5.6 Firewire driver

The media subsystem also provides a firewire driver for digital TV:

Driver	Name
firedtv	FireDTV and FloppyDTV

1.1.5.7 Test drivers

In order to test userspace applications, there's a number of virtual drivers, with provide test functionality, simulating real hardware devices:

Driver	Name
vicodec	Virtual Codec Driver
vim2m	Virtual Memory-to-Memory Driver
vimc	Virtual Media Controller Driver (VIMC)
vivid	Virtual Video Test Driver

1.1.6 Video4Linux (V4L) driver-specific documentation

1.1.6.1 The bttv driver

Release notes for bttv

You'll need at least these config options for bttv:

```
./scripts/config -e PCI
./scripts/config -m I2C
./scripts/config -m INPUT
./scripts/config -m MEDIA_SUPPORT
./scripts/config -e MEDIA_PCI_SUPPORT
./scripts/config -e MEDIA_ANALOG_TV_SUPPORT
./scripts/config -e MEDIA_DIGITAL_TV_SUPPORT
./scripts/config -e MEDIA_RADIO_SUPPORT
./scripts/config -e RC_CORE
./scripts/config -m VIDEO_BT848
```

If your board has digital TV, you'll also need:

```
./scripts/config -m DVB_BT8XX
```

In this case, please see [Documentation/admin-guide/media/bt8xx.rst](#) for additional notes.

Make bttv work with your card

If you have bttv compiled and installed, just booting the Kernel should be enough for it to try probing it. However, depending on the model, the Kernel may require additional information about the hardware, as the device may not be able to provide such info directly to the Kernel.

If it doesn't bttv likely could not autodetect your card and needs some insmod options. The most important insmod option for bttv is "card=n" to select the correct card type. If you get video but no sound you've very likely specified the wrong (or no) card type. A list of supported cards is in [Documentation/admin-guide/media/bttv-cardlist.rst](#).

If bttv takes very long to load (happens sometimes with the cheap cards which have no tuner), try adding this to your modules configuration file (usually, it is either `/etc/modules.conf` or some file at `/etc/modules-load.d/`, but the actual place depends on your distribution):

```
options i2c-algo-bit bit_test=1
```

Some cards may require an extra firmware file to work. For example, for the WinTV/PVR you need one firmware file from its driver CD, called: `hwcwmc.rbf`. It is inside a self-extracting zip file called `pvr45xxx.exe`. Just placing it at the `/etc/firmware` directory should be enough for it to be autoload during the driver's probing mode (e. g. when the Kernel boots or when the driver is manually loaded via `modprobe` command).

If your card isn't listed in [Documentation/admin-guide/media/bttv-cardlist.rst](#) or if you have trouble making audio work, please read [Still doesn't work?](#).

Autodetecting cards

bttv uses the PCI Subsystem ID to autodetect the card type. `lspci` lists the Subsystem ID in the second line, looks like this:

```
00:0a.0 Multimedia video controller: Brooktree Corporation Bt878 (rev 02)
      Subsystem: Hauppauge computer works Inc. WinTV/GO
      Flags: bus master, medium devsel, latency 32, IRQ 5
      Memory at e2000000 (32-bit, prefetchable) [size=4K]
```

only bt878-based cards can have a subsystem ID (which does not mean that every card really has one). bt848 cards can't have a Subsystem ID and therefore can't be autodetected. There is a list with the ID's at [Documentation/admin-guide/media/bttv-cardlist.rst](#) (in case you are interested or want to mail patches with updates).

Still doesn't work?

I do NOT have a lab with 30+ different grabber boards and a PAL/NTSC/SECAM test signal generator at home, so I often can't reproduce your problems. This makes debugging very difficult for me.

If you have some knowledge and spare time, please try to fix this yourself (patches very welcome of course...) You know: The linux slogan is "Do it yourself".

There is a mailing list at <http://vger.kernel.org/vger-lists.html#linux-media>

If you have trouble with some specific TV card, try to ask there instead of mailing me directly. The chance that someone with the same card listens there is much higher...

For problems with sound: There are a lot of different systems used for TV sound all over the world. And there are also different chips which decode the audio signal. Reports about sound problems ("stereo doesn't work") are pretty useless unless you include some details about your hardware and the TV sound scheme used in your country (or at least the country you are living in).

Modprobe options

Note: The following argument list can be outdated, as we might add more options if ever needed. In case of doubt, please check with `modinfo <module>`.

This command prints various information about a kernel module, among them a complete and up-to-date list of `insmod` options.

bttv

The bt848/878 (grabber chip) driver

`insmod` args:

```
card=n           card type, see CARDLIST for a list.
tuner=n         tuner type, see CARDLIST for a list.
radio=0/1      card supports radio
```

```

pll=0/1/2    pll settings
                0: don't use PLL
                1: 28 MHz crystal installed
                2: 35 MHz crystal installed

triton1=0/1   for Triton1 (+others) compatibility
vsfx=0/1     yet another chipset bug compatibility bit
                see README.quirks for details on these two.

bigendian=n   Set the endianness of the gfx framebuffer.
                Default is native endian.
fieldnr=0/1  Count fields. Some TV descrambling software
                needs this, for others it only generates
                50 useless IRQs/sec. default is 0 (off).
autoload=0/1  autoload helper modules (tuner, audio).
                default is 1 (on).
bttv_verbose=0/1/2  verbose level (at insmod time, while
                looking at the hardware). default is 1.
bttv_debug=0/1  debug messages (for capture).
                default is 0 (off).
irq_debug=0/1  irq handler debug messages.
                default is 0 (off).
gbuffers=2-32  number of capture buffers for mmap'ed capture.
                default is 4.
gbufsize=     size of capture buffers. default and
                maximum value is 0x208000 (~2MB)
no_overlay=0   Enable overlay on broken hardware. There
                are some chipsets (SIS for example) which
                are known to have problems with the PCI DMA
                push used by bttv. bttv will disable overlay
                by default on this hardware to avoid crashes.
                With this insmod option you can override this.
no_overlay=1   Disable overlay. It should be used by broken
                hardware that doesn't support PCI2PCI direct
                transfers.
automute=0/1   Automatically mutes the sound if there is
                no TV signal, on by default. You might try
                to disable this if you have bad input signal
                quality which leading to unwanted sound
                dropouts.
chroma_agc=0/1  AGC of chroma signal, off by default.
adc_crush=0/1  Luminance ADC crush, on by default.
i2c_udelay=    Allow reduce I2C speed. Default is 5 usecs
                (meaning 66,67 Kbps). The default is the
                maximum supported speed by kernel bitbang
                algorithm. You may use lower numbers, if I2C
                messages are lost (16 is known to work on
                all supported cards).

bttv_gpio=0/1
gpiomask=
audioall=
audiomux=
                See Sound-FAQ for a detailed description.

remap, card, radio and pll accept up to four comma-separated arguments

```

(for multiple boards).

tuner The tuner driver. You need this unless you want to use only with a camera or the board doesn't provide analog TV tuning.

insmod args:

debug=1	print some debug info to the syslog
type=n	type of the tuner chip. n as follows: see CARDLIST for a complete list.
pal=[bdgil]	select PAL variant (used for some tuners only, important for the audio carrier).

tvaudio Provide a single driver for all simple i2c audio control chips (tda/tea*).

insmod args:

tda8425 = 1	enable/disable the support for the
tda9840 = 1	various chips.
tda9850 = 1	The tea6300 can't be autodetected and is
tda9855 = 1	therefore off by default, if you have
tda9873 = 1	this one on your card (STB uses these)
tda9874a = 1	you have to enable it explicitly.
tea6300 = 0	The two tda985x chips use the same i2c
tea6420 = 1	address and can't be distinguished from
pic16c54 = 1	each other, you might have to disable
	the wrong one.
debug = 1	print debug messages

msp3400 The driver for the msp34xx sound processor chips. If you have a stereo card, you probably want to insmod this one.

insmod args:

debug=1/2	print some debug info to the syslog, 2 is more verbose.
simple=1	Use the "short programming" method. Newer msp34xx versions support this. You need this for dbx stereo. Default is on if supported by the chip.
once=1	Don't check the TV-stations Audio mode every few seconds, but only once after channel switches.
amsound=1	Audio carrier is AM/NICAM at 6.5 Mhz. This should improve things for french people, the carrier autoscans seems to work with FM only...

If the box freezes hard with bttv

It might be a bttv driver bug. It also might be bad hardware. It also might be something else ... Just mailing me “bttv freezes” isn’t going to help much. This README has a few hints how you can help to pin down the problem.

bttv bugs

If some version works and another doesn’t it is likely to be a driver bug. It is very helpful if you can tell where exactly it broke (i.e. the last working and the first broken version).

With a hard freeze you probably doesn’t find anything in the logfiles. The only way to capture any kernel messages is to hook up a serial console and let some terminal application log the messages. /me uses screen. See Documentation/admin-guide/serial-console.rst for details on setting up a serial console.

Read Documentation/admin-guide/bug-hunting.rst to learn how to get any useful information out of a register+stack dump printed by the kernel on protection faults (so-called “kernel oops”).

If you run into some kind of deadlock, you can try to dump a call trace for each process using sysrq-t (see Documentation/admin-guide/sysrq.rst). This way it is possible to figure where *exactly* some process in “D” state is stuck.

I’ve seen reports that bttv 0.7.x crashes whereas 0.8.x works rock solid for some people. Thus probably a small buglet left somewhere in bttv 0.7.x. I have no idea where exactly, it works stable for me and a lot of other people. But in case you have problems with the 0.7.x versions you can give 0.8.x a try ...

hardware bugs

Some hardware can’t deal with PCI-PCI transfers (i.e. grabber => vga). Sometimes problems show up with bttv just because of the high load on the PCI bus. The bt848/878 chips have a few workarounds for known incompatibilities, see README.quirks.

Some folks report that increasing the pci latency helps too, although I’m not sure whenever this really fixes the problems or only makes it less likely to happen. Both bttv and btaudio have a insmod option to set the PCI latency of the device.

Some mainboard have problems to deal correctly with multiple devices doing DMA at the same time. bttv + ide seems to cause this sometimes, if this is the case you likely see freezes only with video and hard disk access at the same time. Updating the IDE driver to get the latest and greatest workarounds for hardware bugs might fix these problems.

other

If you use some binary-only junk (like nvidia module) try to reproduce the problem without.

IRQ sharing is known to cause problems in some cases. It works just fine in theory and many configurations. Nevertheless it might be worth a try to shuffle around the PCI cards to give bttv another IRQ or make it share the IRQ with some other piece of hardware. IRQ sharing with VGA cards seems to cause trouble sometimes. I've also seen funny effects with bttv sharing the IRQ with the ACPI bridge (and apci-enabled kernel).

Bttv quirks

Below is what the bt878 data book says about the PCI bug compatibility modes of the bt878 chip.

The triton1 insmod option sets the EN_TBFX bit in the control register. The vsfx insmod option does the same for EN_VSFX bit. If you have stability problems you can try if one of these options makes your box work solid.

drivers/pci/quirks.c knows about these issues, this way these bits are enabled automagically for known-buggy chipsets (look at the kernel messages, bttv tells you).

Normal PCI Mode

The PCI REQ signal is the logical-or of the incoming function requests. The internal GNT[0:1] signals are gated asynchronously with GNT and demultiplexed by the audio request signal. Thus the arbiter defaults to the video function at power-up and parks there during no requests for bus access. This is desirable since the video will request the bus more often. However, the audio will have highest bus access priority. Thus the audio will have first access to the bus even when issuing a request after the video request but before the PCI external arbiter has granted access to the Bt879. Neither function can preempt the other once on the bus. The duration to empty the entire video PCI FIFO onto the PCI bus is very short compared to the bus access latency the audio PCI FIFO can tolerate.

430FX Compatibility Mode

When using the 430FX PCI, the following rules will ensure compatibility:

- (1) Deassert REQ at the same time as asserting FRAME.
- (2) Do not reassert REQ to request another bus transaction until after finishing the previous transaction.

Since the individual bus masters do not have direct control of REQ, a simple logical-or of video and audio requests would violate the rules. Thus, both the arbiter and the initiator contain 430FX compatibility mode logic. To enable 430FX mode, set the EN_TBFX bit as indicated in Device Control Register on page 104.

When EN_TBFX is enabled, the arbiter ensures that the two compatibility rules are satisfied. Before GNT is asserted by the PCI arbiter, this internal arbiter may still logical-or the two requests. However, once the GNT is issued, this arbiter must lock in its decision and now route only the granted request to the REQ pin. The arbiter decision lock happens regardless of the

state of FRAME because it does not know when FRAME will be asserted (typically - each initiator will assert FRAME on the cycle following GNT). When FRAME is asserted, it is the initiator's responsibility to remove its request at the same time. It is the arbiter's responsibility to allow this request to flow through to REQ and not allow the other request to hold REQ asserted. The decision lock may be removed at the end of the transaction: for example, when the bus is idle (FRAME and IRDY). The arbiter decision may then continue asynchronously until GNT is again asserted.

Interfacing with Non-PCI 2.1 Compliant Core Logic

A small percentage of core logic devices may start a bus transaction during the same cycle that GNT is de-asserted. This is non PCI 2.1 compliant. To ensure compatibility when using PCs with these PCI controllers, the EN_VSFX bit must be enabled (refer to Device Control Register on page 104). When in this mode, the arbiter does not pass GNT to the internal functions unless REQ is asserted. This prevents a bus transaction from starting the same cycle as GNT is de-asserted. This also has the side effect of not being able to take advantage of bus parking, thus lowering arbitration performance. The Bt879 drivers must query for these non-compliant devices, and set the EN_VSFX bit only if required.

Other elements of the tvcards array

If you are trying to make a new card work you might find it useful to know what the other elements in the tvcards array are good for:

video_inputs	- # of video inputs the card has
audio_inputs	- historical cruft, not used any more.
tuner	- which input is the tuner
svhs	- which input is svhs (all others are labeled composite)
muxsel	- video mux, input->registervalue mapping
pll	- same as pll= insmod option
tuner_type	- same as tuner= insmod option
*_modulename	- hint whenever some card needs this or that audio module loaded to work properly.
has_radio	- whenever this TV card has a radio tuner.
no_msp34xx	- "1" disables loading of msp3400.o module
no_tda9875	- "1" disables loading of tda9875.o module
needs_tvaudio	- set to "1" to load tvaudio.o module

If some config item is specified both from the tvcards array and as insmod option, the insmod option takes precedence.

Cards

Note: For a more updated list, please check https://linuxtv.org/wiki/index.php/Hardware_Device_Information

Supported cards: Bt848/Bt848a/Bt849/Bt878/Bt879 cards

All cards with Bt848/Bt848a/Bt849/Bt878/Bt879 and normal Composite/S-VHS inputs are supported. Teletext and Intercast support (PAL only) for ALL cards via VBI sample decoding in software.

Some cards with additional multiplexing of inputs or other additional fancy chips are only partially supported (unless specifications by the card manufacturer are given). When a card is listed here it isn't necessarily fully supported.

All other cards only differ by additional components as tuners, sound decoders, EEPROMs, teletext decoders ...

MATRIX Vision

MV-Delta - Bt848A - 4 Composite inputs, 1 S-VHS input (shared with 4th composite) - EEPROM

<http://www.matrix-vision.de/>

This card has no tuner but supports all 4 composite (1 shared with an S-VHS input) of the Bt848A. Very nice card if you only have satellite TV but several tuners connected to the card via composite.

Many thanks to Matrix-Vision for giving us 2 cards for free which made Bt848a/Bt849 single crystal operation support possible!!!

Miro/Pinnacle PCTV

- Bt848 some (all??) come with 2 crystals for PAL/SECAM and NTSC
- PAL, SECAM or NTSC TV tuner (Philips or TEMIC)
- MSP34xx sound decoder on add on board decoder is supported but AFAIK does not yet work (other sound MUX setting in GPIO port needed??? somebody who fixed this???)
- 1 tuner, 1 composite and 1 S-VHS input
- tuner type is autodetected

<http://www.miro.de/> <http://www.miro.com/>

Many thanks for the free card which made first NTSC support possible back in 1997!

Hauppauge Win/TV pci

There are many different versions of the Hauppauge cards with different tuners (TV+Radio ...), teletext decoders. Note that even cards with same model numbers have (depending on the revision) different chips on it.

- Bt848 (and others but always in 2 crystal operation???) newer cards have a Bt878
- PAL, SECAM, NTSC or tuner with or without Radio support

e.g.:

- PAL:

- TDA5737: VHF, hyperband and UHF mixer/oscillator for TV and VCR 3-band tuners
- TSA5522: 1.4 GHz I2C-bus controlled synthesizer, I2C 0xc2-0xc3
- NTSC:
 - TDA5731: VHF, hyperband and UHF mixer/oscillator for TV and VCR 3-band tuners
 - TSA5518: no datasheet available on Philips site
- Philips SAA5246 or SAA5284 (or no) Teletext decoder chip with buffer RAM (e.g. Winbond W24257AS-35: 32Kx8 CMOS static RAM) SAA5246 (I2C 0x22) is supported
- 256 bytes EEPROM: Microchip 24LC02B or Philips 8582E2Y with configuration information I2C address 0xa0 (24LC02B also responds to 0xa2-0xaf)
- 1 tuner, 1 composite and (depending on model) 1 S-VHS input
- 14052B: mux for selection of sound source
- sound decoder: TDA9800, MSP34xx (stereo cards)

Askey CPH-Series

Developed by TelSignal(?), OEMed by many vendors (Typhoon, Anubis, Dynalink)

- Card series: - CPH01x: BT848 capture only - CPH03x: BT848 - CPH05x: BT878 with FM - CPH06x: BT878 (w/o FM) - CPH07x: BT878 capture only
- TV standards: - CPH0x0: NTSC-M/M - CPH0x1: PAL-B/G - CPH0x2: PAL-I/I - CPH0x3: PAL-D/K - CPH0x4: SECAM-L/L - CPH0x5: SECAM-B/G - CPH0x6: SECAM-D/K - CPH0x7: PAL-N/N - CPH0x8: PAL-B/H - CPH0x9: PAL-M/M
- CPH03x was often sold as "TV capturer".

Identifying:

- 1) 878 cards can be identified by PCI Subsystem-ID: - 144f:3000 = CPH06x - 144F:3002 = CPH05x w/ FM - 144F:3005 = CPH06x_LC (w/o remote control)
- 2) The cards have a sticker with "CPH"-model on the back.
- 3) These cards have a number printed on the PCB just above the tuner metal box: - "80-CP2000300-x" = CPH03X - "80-CP2000500-x" = CPH05X - "80-CP2000600-x" = CPH06X / CPH06x_LC

Askey sells these cards as "Magic TView series", Brand "MagicXpress". Other OEM often call these "Tview", "TView99" or else.

Lifeview Flyvideo Series:

The naming of these series differs in time and space.

Identifying:

1) Some models can be identified by PCI subsystem ID:

- 1852:1852 = Flyvideo 98 FM
- 1851:1850 = Flyvideo 98
- 1851:1851 = Flyvideo 98 EZ (capture only)

2) There is a print on the PCB:

- LR25 = Flyvideo (Zoran ZR36120, SAA7110A)
- LR26 Rev.N = Flyvideo II (Bt848)
- LR26 Rev.O = Flyvideo II (Bt878)
- LR37 Rev.C = Flyvideo EZ (Capture only, ZR36120 + SAA7110)
- LR38 Rev.A1 = Flyvideo II EZ (Bt848 capture only)
- LR50 Rev.Q = Flyvideo 98 (w/eprom and PCI subsystem ID)
- LR50 Rev.W = Flyvideo 98 (no eprom)
- LR51 Rev.E = Flyvideo 98 EZ (capture only)
- LR90 = Flyvideo 2000 (Bt878)
- LR90 Flyvideo 2000S (Bt878) w/Stereo TV (Package incl. LR91 daughterboard)
- LR91 = Stereo daughter card for LR90
- LR97 = Flyvideo DVBS
- LR99 Rev.E = Low profile card for OEM integration (only internal audio!) bt878
- LR136 = Flyvideo 2100/3100 (Low profile, SAA7130/SAA7134)
- LR137 = Flyvideo DV2000/DV3000 (SAA7130/SAA7134 + IEEE1394)
- LR138 Rev.C = Flyvideo 2000 (SAA7130)
- LR138 Flyvideo 3000 (SAA7134) w/Stereo TV
 - These exist in variations w/FM and w/Remote sometimes denoted by suffixes "FM" and "R".

3) You have a laptop (miniPCI card):

- Product = FlyTV Platinum Mini
- Model/Chip = LR212/saa7135
- Lifeview.com.tw states (Feb. 2002): "The FlyVideo2000 and FlyVideo2000s product name have renamed to FlyVideo98." Their Bt8x8 cards are listed as discontinued.
- Flyvideo 2000S was probably sold as Flyvideo 3000 in some countries(Europe?). The new Flyvideo 2000/3000 are SAA7130/SAA7134 based.

“Flyvideo II” had been the name for the 848 cards, nowadays (in Germany) this name is re-used for LR50 Rev.W.

The Lifeview website mentioned Flyvideo III at some time, but such a card has not yet been seen (perhaps it was the german name for LR90 [stereo]). These cards are sold by many OEMs too.

FlyVideo A2 (Elta 8680)= LR90 Rev.F (w/Remote, w/o FM, stereo TV by tda9821) {Germany}
 Lifeview 3000 (Elta 8681) as sold by Plus(April 2002), Germany = LR138 w/ saa7134

lifeview config coding on gpio pins 0-9

- LR50 rev. Q (“PARTS: 7031505116), Tuner wurde als Nr. 5 erkannt, Eingänge SVideo, TV, Composite, Audio, Remote:
- CP9..1=100001001 (1: 0-Ohm-Widerstand gegen GND unbestückt; 0: bestückt)

Typhoon TV card series:

These can be CPH, Flyvideo, Pixelview or KNC1 series.

Typhoon is the brand of Anubis.

Model 50680 got re-used, some model no. had different contents over time.

Models:

- 50680 “TV Tuner PCI Pal BG”(old,red package)=can be CPH03x(bt848) or CPH06x(bt878)
- 50680 “TV Tuner Pal BG” (blue package)= Pixelview PV-BT878P+ (Rev 9B)
- 50681 “TV Tuner PCI Pal I” (variant of 50680)
- 50682 “TView TV/FM Tuner Pal BG” = Flyvideo 98FM (LR50 Rev.Q)

Note: The package has a picture of CPH05x (which would be a real TView)

- 50683 “TV Tuner PCI SECAM” (variant of 50680)
- 50684 “TV Tuner Pal BG” = Pixelview 878TV(Rev.3D)
- 50686 “TV Tuner” = KNC1 TV Station
- 50687 “TV Tuner stereo” = KNC1 TV Station pro
- 50688 “TV Tuner RDS” (black package) = KNC1 TV Station RDS
- 50689 TV SAT DVB-S CARD CI PCI (SAA7146AH, SU1278?) = “KNC1 TV Station DVB-S”
- 50692 “TV/FM Tuner” (small PCB)
- 50694 TV TUNER CARD RDS (PHILIPS CHIPSET SAA7134HL)
- 50696 TV TUNER STEREO (PHILIPS CHIPSET SAA7134HL, MK3ME Tuner)

- 50804 PC-SAT TV/Audio Karte = Techni-PC-Sat (ZORAN 36120PQC, Tuner:Alps)
- 50866 TVIEW SAT RECEIVER+ADR
- 50868 “TV/FM Tuner Pal I” (variant of 50682)
- 50999 “TV/FM Tuner Secam” (variant of 50682)

Guillemot

Models:

- Maxi-TV PCI (ZR36120)
- Maxi TV Video 2 = LR50 Rev.Q (FI1216MF, PAL BG+SECAM)
- Maxi TV Video 3 = CPH064 (PAL BG + SECAM)

Mentor

Mentor TV card (“55-878TV-U1”) = Pixelview 878TV(Rev.3F) (w/FM w/Remote)

Prolink

- TV cards:
 - PixelView Play TV pro - (Model: PV-BT878P+ REV 8E)
 - PixelView Play TV pro - (Model: PV-BT878P+ REV 9D)
 - PixelView Play TV pro - (Model: PV-BT878P+ REV 4C / 8D / 10A)
 - PixelView Play TV - (Model: PV-BT848P+)
 - 878TV - (Model: PV-BT878TV)
- Multimedia TV packages (card + software pack):
 - PixelView Play TV Theater - (Model: PV-M4200) = PixelView Play TV pro + Software
 - PixelView Play TV PAK - (Model: PV-BT878P+ REV 4E)
 - PixelView Play TV/VCR - (Model: PV-M3200 REV 4C / 8D / 10A)
 - PixelView Studio PAK - (Model: M2200 REV 4C / 8D / 10A)
 - PixelView PowerStudio PAK - (Model: PV-M3600 REV 4E)
 - PixelView DigitalVCR PAK - (Model: PV-M2400 REV 4C / 8D / 10A)
 - PixelView PlayTV PAK II (TV/FM card + usb camera) PV-M3800
 - PixelView PlayTV XP PV-M4700,PV-M4700(w/FM)
 - PixelView PlayTV DVR PV-M4600 package contents:PixelView PlayTV pro, windvr & videoMail s/w
- Further Cards:
 - PV-BT878P+rev.9B (Play TV Pro, opt. w/FM w/NICAM)

- PV-BT878P+rev.2F
- PV-BT878P Rev.1D (bt878, capture only)
- XCapture PV-CX881P (cx23881)
- PlayTV HD PV-CX881PL+, PV-CX881PL+(w/FM) (cx23881)
- DTV3000 PV-DTV3000P+ DVB-S CI = Twinhan VP-1030
- DTV2000 DVB-S = Twinhan VP-1020
- Video Conferencing:
 - PixelView Meeting PAK - (Model: PV-BT878P)
 - PixelView Meeting PAK Lite - (Model: PV-BT878P)
 - PixelView Meeting PAK plus - (Model: PV-BT878P+rev 4C/8D/10A)
 - PixelView Capture - (Model: PV-BT848P)
 - PixelView PlayTV USB pro
 - Model No. PV-NT1004+, PV-NT1004+ (w/FM) = NT1004 USB decoder chip + SAA7113 video decoder chip

Dynalink

These are CPH series.

Phoebemicro

- TV Master = CPH030 or CPH060
- TV Master FM = CPH050

Genius/Kye

- Video Wonder/Genius Internet Video Kit = LR37 Rev.C
- Video Wonder Pro II (848 or 878) = LR26

Tekram

- VideoCap C205 (Bt848)
- VideoCap C210 (zr36120 +Philips)
- CaptureTV M200 (ISA)
- CaptureTV M205 (Bt848)

Lucky Star

- Image World Conference TV = LR50 Rev. Q

Leadtek

- WinView 601 (Bt848)
- WinView 610 (Zoran)
- WinFast2000
- WinFast2000 XP

Support for the Leadtek WinView 601 TV/FM

Author of this section: Jon Tombs <jon@gte.esi.us.es>

This card is basically the same as all the rest (Bt484A, Philips tuner), the main difference is that they have attached a programmable attenuator to 3 GPIO lines in order to give some volume control. They have also stuck an infra-red remote control decoded on the board, I will add support for this when I get time (it simple generates an interrupt for each key press, with the key code is placed in the GPIO port).

I don't yet have any application to test the radio support. The tuner frequency setting should work but it is possible that the audio multiplexer is wrong. If it doesn't work, send me email.

- No Thanks to Leadtek they refused to answer any questions about their hardware. The driver was written by visual inspection of the card. If you use this driver, send an email insult to them, and tell them you won't continue buying their hardware unless they support Linux.
- Little thanks to Princeton Technology Corp (<http://www.princeton.com.tw>) who make the audio attenuator. Their publicly available data-sheet available on their web site doesn't include the chip programming information! Hidden on their server are the full data-sheets, but don't ask how I found it.

To use the driver I use the following options, the tuner and pll settings might be different in your country. You can force it via modprobe parameters. For example:

```
modprobe bttv tuner=1 pll=28 radio=1 card=17
```

Sets tuner type 1 (Philips PAL_I), PLL with a 28 MHz crystal, enables FM radio and selects bttv card ID 17 (Leadtek WinView 601).

KNC One

- TV-Station
- TV-Station SE (+Software Bundle)
- TV-Station pro (+TV stereo)
- TV-Station FM (+Radio)
- TV-Station RDS (+RDS)
- TV Station SAT (analog satellite)
- TV-Station DVB-S

Note: newer Cards have saa7134, but model name stayed the same?

Provideo

- PV951 or PV-951, now named PV-951T (also are sold as: Boeder TV-FM Video Capture Card, Titanmedia Supervision TV-2400, Provideo PV951 TF, 3DeMon PV951, MediaForte TV-Vision PV951, Yoko PV951, Vivanco Tuner Card PCI Art.-Nr.: 68404)
- Surveillance Series:
 - PV-141
 - PV-143
 - PV-147
 - PV-148 (capture only)
 - PV-150
 - PV-151
- TV-FM Tuner Series:
 - PV-951TDV (tv tuner + 1394)
 - PV-951T/TF
 - PV-951PT/TF
 - PV-956T/TF Low Profile
 - PV-911

Highscreen

Models:

- TV Karte = LR50 Rev.S
- TV-Boostar = Terratec Terra TV+ Version 1.0 (Bt848, tda9821) "ceb105.pcb"

Zoltrix

Models:

- Face to Face Capture (Bt848 capture only) (PCB "VP-2848")
- Face To Face TV MAX (Bt848) (PCB "VP-8482 Rev1.3")
- Genie TV (Bt878) (PCB "VP-8790 Rev 2.1")
- Genie Wonder Pro

AVerMedia

- AVer FunTV Lite (ISA, AV3001 chipset) "M101.C"
- AVerTV
- AVerTV Stereo
- AVerTV Studio (w/FM)
- AVerMedia TV98 with Remote
- AVerMedia TV/FM98 Stereo
- AVerMedia TVCAM98
- TVCapture (Bt848)
- TVPhone (Bt848)
- TVCapture98 (= "AVerMedia TV98" in USA) (Bt878)
- TVPhone98 (Bt878, w/FM)

PCB	PCI-ID	Model-Name	Eeprom	Tuner	Sound	Country
M101.C	ISA !					
M108-B	Bt848		-	FR1236		US ^{2,3}
M1A8-A	Bt848	AVer TV-Phone		FM1216	-	
M168-T	1461:0003	AVerTV Studio	48:17	FM1216	TDA9840T	D ¹ w/FM w/Remote
M168-U	1461:0004	TVCapture98	40:11	FI1216	-	D w/Remote
M168II-B	1461:0003	Medion MD9592	48:16	FM1216	TDA9873H	D w/FM

² Sony NE41S soldered (stereo sound?)

³ Daughterboard M118-A w/ pic 16c54 and 4 MHz quartz

¹ Daughterboard MB68-A with TDA9820T and TDA9840T

- US site has different drivers for (as of 09/2002):
 - EZ Capture/InterCam PCI (BT-848 chip)
 - EZ Capture/InterCam PCI (BT-878 chip)
 - TV-Phone (BT-848 chip)
 - TV98 (BT-848 chip)
 - TV98 With Remote (BT-848 chip)
 - TV98 (BT-878 chip)
 - TV98 With Remote (BT-878)
 - TV/FM98 (BT-878 chip)
 - AVerTV
 - AVerTV Stereo
 - AVerTV Studio

DE hat diverse Treiber fuer diese Modelle (Stand 09/2002):

- TVPhone (848) mit Philips tuner FR12X6 (w/ FM radio)
- TVPhone (848) mit Philips tuner FM12X6 (w/ FM radio)
- TVCapture (848) w/Philips tuner FI12X6
- TVCapture (848) non-Philips tuner
- TVCapture98 (Bt878)
- TVPhone98 (Bt878)
- AVerTV und TVCapture98 w/VCR (Bt 878)
- AVerTVStudio und TVPhone98 w/VCR (Bt878)
- AVerTV GO Serie (Kein SVideo Input)
- AVerTV98 (BT-878 chip)
- AVerTV98 mit Fernbedienung (BT-878 chip)
- AVerTV/FM98 (BT-878 chip)
- VDOmate (www.averm.com.cn) = M168U ?

Aimslab

Models:

- Video Highway or "Video Highway TR200" (ISA)
- Video Highway Xtreme (aka "VHX") (Bt848, FM w/ TEA5757)

IXMicro (former: IMS=Integrated Micro Solutions)

Models:

- IXTV BT848 (=TurboTV)
- IXTV BT878
- IMS TurboTV (Bt848)

Lifetec/Medion/Tevion/Aldi

Models:

- LT9306/MD9306 = CPH061
- LT9415/MD9415 = LR90 Rev.F or Rev.G
- MD9592 = Avermedia TVphone98 (PCI_ID=1461:0003), PCB-Rev=M168II-B (w/TDA9873H)
- MD9717 = KNC One (Rev D4, saa7134, FM1216 MK2 tuner)
- MD5044 = KNC One (Rev D4, saa7134, FM1216ME MK3 tuner)

Modular Technologies (www.modulartech.com) UK

Models:

- MM100 PCTV (Bt848)
- MM201 PCTV (Bt878, Bt832) w/ Quartzsight camera
- MM202 PCTV (Bt878, Bt832, tda9874)
- MM205 PCTV (Bt878)
- MM210 PCTV (Bt878) (Galaxy TV, Galaxymedia ?)

Terratec

Models:

- Terra TV+ Version 1.0 (Bt848), "ceb105.PCB" printed on the PCB, TDA9821
- Terra TV+ Version 1.1 (Bt878), "LR74 Rev.E" printed on the PCB, TDA9821
- Terra TValueRadio, "LR102 Rev.C" printed on the PCB
- Terra TV/Radio+ Version 1.0, "80-CP2830100-0" TTTV3 printed on the PCB, "CPH010-E83" on the back, SAA6588T, TDA9873H
- Terra TValue Version BT878, "80-CP2830110-0 TTTV4" printed on the PCB, "CPH011-D83" on back
- Terra TValue Version 1.0 "ceb105.PCB" (really identical to Terra TV+ Version 1.0)
- Terra TValue New Revision "LR102 Rec.C"

- Terra Active Radio Upgrade (tea5757h, saa6588t)
- LR74 is a newer PCB revision of ceb105 (both incl. connector for Active Radio Upgrade)
- Cinergy 400 (saa7134), "E877 11(S)", "PM820092D" printed on PCB
- Cinergy 600 (saa7134)

Technisat

Models:

- Discos ADR PC-Karte ISA (no TV!)
- Discos ADR PC-Karte PCI (probably no TV?)
- Techni-PC-Sat (Sat. analog) Rev 1.2 (zr36120, vpx3220, stv0030, saa5246, BSJE3-494A)
- Mediafocus I (zr36120/zr36125, drp3510, Sat. analog + ADR Radio)
- Mediafocus II (saa7146, Sat. analog)
- SatADR Rev 2.1 (saa7146a, saa7113h, stv0056a, msp3400c, drp3510a, BSKE3-307A)
- SkyStar 1 DVB (AV7110) = Technotrend Premium
- SkyStar 2 DVB (B2C2) (=Sky2PC)

Siemens

Multimedia eXtension Board (MXB) (SAA7146, SAA7111)

Powercolor

Models:

- **MTV878** Package comes with different contents:
 - a) pcb "MTV878" (CARD=75)
 - b) Pixelview Rev. 4_
- MTV878R w/Remote Control
- MTV878F w/Remote Control w/FM radio

Pinnacle

PCTV models:

- Mirovideo PCTV (Bt848)
- Mirovideo PCTV SE (Bt848)
- Mirovideo PCTV Pro (Bt848 + Daughterboard for TV Stereo and FM)
- Studio PCTV Rave (Bt848 Version = Mirovideo PCTV)

- Studio PCTV Rave (Bt878 package w/o infrared)
- Studio PCTV (Bt878)
- Studio PCTV Pro (Bt878 stereo w/ FM)
- Pinnacle PCTV (Bt878, MT2032)
- Pinnacle PCTV Pro (Bt878, MT2032)
- Pinnacle PCTV Sat (bt878a, HM1821/1221) [“Conexant CX24110 with CX24108 tuner, aka HM1221/HM1811”]
- Pinnacle PCTV Sat XE

M(J)PEG capture and playback models:

- DC1+ (ISA)
- DC10 (zr36057, zr36060, saa7110, adv7176)
- DC10+ (zr36067, zr36060, saa7110, adv7176)
- DC20 (ql16x24b,zr36050, zr36016, saa7110, saa7187 ...)
- DC30 (zr36057, zr36050, zr36016, vpx3220, adv7176, ad1843, tea6415, miro FST97A1)
- DC30+ (zr36067, zr36050, zr36016, vpx3220, adv7176)
- DC50 (zr36067, zr36050, zr36016, saa7112, adv7176 (2 pcs.?), ad1843, miro FST97A1, Lattice ???)

Lenco

Models:

- MXR-9565 (=Technisat Mediafocus?)
- MXR-9571 (Bt848) (=CPH031?)
- MXR-9575
- MXR-9577 (Bt878) (=Prolink 878TV Rev.3x)
- MXTV-9578CP (Bt878) (= Prolink PV-BT878P+4E)

lomega

Buz (zr36067, zr36060, saa7111, saa7185)

LML

LML33 (zr36067, zr36060, bt819, bt856)

Grandtec

Models:

- Grand Video Capture (Bt848)
- Multi Capture Card (Bt878)

Koutech

Models:

- KW-606 (Bt848)
- KW-607 (Bt848 capture only)
- KW-606RSF
- KW-607A (capture only)
- KW-608 (Zoran capture only)

IODATA (jp)

Models:

- GV-BCTV/PCI
- GV-BCTV2/PCI
- GV-BCTV3/PCI
- GV-BCTV4/PCI
- GV-VCP/PCI (capture only)
- GV-VCP2/PCI (capture only)

Canopus (jp)

WinDVR = Kworld "KW-TVL878RF"

www.sigmacom.co.kr

Sigma Cyber TV II

www.sasem.co.kr

Litte OnAir TV

hama

TV/Radio-Tuner Card, PCI (Model 44677) = CPH051

Sigma Designs

Hollywood plus (em8300, em9010, adv7175), (PCB "M340-10") MPEG DVD decoder

Formac

Models:

- iProTV (Card for iMac Mezzanine slot, Bt848+SCSI)
- ProTV (Bt848)
- ProTV II = ProTV Stereo (Bt878) ["stereo" means FM stereo, tv is still mono]

ATI

Models:

- TV-Wonder
- TV-Wonder VE

Diamond Multimedia

DTV2000 (Bt848, tda9875)

Aopen

- VA1000 Plus (w/ Stereo)
- VA1000 Lite
- VA1000 (=LR90)

Intel

Models:

- Smart Video Recorder (ISA full-length)
- Smart Video Recorder pro (ISA half-length)
- Smart Video Recorder III (Bt848)

STB

Models:

- STB Gateway 6000704 (bt878)
- STB Gateway 6000699 (bt848)
- STB Gateway 6000402 (bt848)
- STB TV130 PCI

Videologic

Models:

- Captivator Pro/TV (ISA?)
- Captivator PCI/VC (Bt848 bundled with camera) (capture only)

Technotrend

Models:

- TT-SAT PCI (PCB "Sat-PCI Rev.:1.3.1"; zr36125, vpx3225d, stc0056a, Tuner:BSKE6-155A)
- **TT-DVB-Sat**
 - revisions 1.1, 1.3, 1.5, 1.6 and 2.1
 - This card is sold as OEM from:
 - * Siemens DVB-s Card
 - * Hauppauge WinTV DVB-S
 - * Technisat SkyStar 1 DVB
 - * Galaxis DVB Sat
 - Now this card is called TT-PCLine Premium Family
 - TT-Budget (saa7146, bsru6-701a) This card is sold as OEM from:
 - * Hauppauge WinTV Nova
 - * Satelco Standard PCI (DVB-S)
 - TT-DVB-C PCI

Teles

DVB-s (Rev. 2.2, BSRV2-301A, data only?)

Remote Vision

MX RV605 (Bt848 capture only)

Boeder

Models:

- PC ChatCam (Model 68252) (Bt848 capture only)
- Tv/Fm Capture Card (Model 68404) = PV951

Media-Surfer (esc-kathrein.de)

Models:

- Sat-Surfer (ISA)
- Sat-Surfer PCI = Techni-PC-Sat
- Cable-Surfer 1
- Cable-Surfer 2
- Cable-Surfer PCI (zr36120)
- Audio-Surfer (ISA Radio card)

Jetway (www.jetway.com.tw)

Models:

- JW-TV 878M
- JW-TV 878 = KWorld KW-TV878RF

Galaxis

Models:

- Galaxis DVB Card S CI
- Galaxis DVB Card C CI
- Galaxis DVB Card S
- Galaxis DVB Card C
- Galaxis plug.in S [neuer Name: Galaxis DVB Card S CI]

Hauppauge

Models:

- many many WinTV models ...
- WinTV DVBs = Technotrend Premium 1.3
- WinTV NOVA = Technotrend Budget 1.1 "S-DVB DATA"
- WinTV NOVA-CI "SDVBACI"
- WinTV Nova USB (=Technotrend USB 1.0)
- WinTV-Nexus-s (=Technotrend Premium 2.1 or 2.2)
- WinTV PVR
- WinTV PVR 250
- WinTV PVR 450

US models

-990 WinTV-PVR-350 (249USD) (iTVC15 chipset + radio) -980 WinTV-PVR-250 (149USD) (iTVC15 chipset) -880 WinTV-PVR-PCI (199USD) (KFIR chipset + bt878) -881 WinTV-PVR-USB -190 WinTV-GO -191 WinTV-GO-FM -404 WinTV -401 WinTV-radio -495 WinTV-Theater -602 WinTV-USB -621 WinTV-USB-FM -600 USB-Live -698 WinTV-HD -697 WinTV-D -564 WinTV-Nexus-S

Deutsche Modelle:

-603 WinTV GO -719 WinTV Primio-FM -718 WinTV PCI-FM -497 WinTV Theater -569 WinTV USB -568 WinTV USB-FM -882 WinTV PVR -981 WinTV PVR 250 -891 WinTV-PVR-USB -541 WinTV Nova -488 WinTV Nova-Ci -564 WinTV-Nexus-s -727 WinTV-DVB-c -545 Common Interface -898 WinTV-Nova-USB

UK models:

-607 WinTV Go -693,793 WinTV Primio FM -647,747 WinTV PCI FM -498 WinTV Theater -883 WinTV PVR -893 WinTV PVR USB (Duplicate entry) -566 WinTV USB (UK) -573 WinTV USB FM -429 Impact VCB (bt848) -600 USB Live (Video-In 1x Comp, 1xSVHS) -542 WinTV Nova -717 WinTV DVB-S -909 Nova-t PCI -893 Nova-t USB (Duplicate entry) -802 MyTV -804 MyView -809 MyVideo -872 MyTV2Go FM -546 WinTV Nova-S CI -543 WinTV Nova -907 Nova-S USB -908 Nova-T USB -717 WinTV Nexus-S -157 DEC3000-s Standalone + USB

Spain:

-685 WinTV-Go -690 WinTV-PrimioFM -416 WinTV-PCI Nicam Estereo -677 WinTV-PCI-FM -699 WinTV-Theater -683 WinTV-USB -678 WinTV-USB-FM -983 WinTV-PVR-250 -883 WinTV-PVR-PCI -993 WinTV-PVR-350 -893 WinTV-PVR-USB -728 WinTV-DVB-C PCI -832 MyTV2Go -869 MyTV2Go-FM -805 MyVideo (USB)

Matrix-Vision

Models:

- MATRIX-Vision MV-Delta
- MATRIX-Vision MV-Delta 2
- MVsigma-SLC (Bt848)

Conceptronic (.net)

Models:

- TVCON FM, TV card w/ FM = CPH05x
- TVCON = CPH06x

BestData

Models:

- HCC100 = VCC100rev1 + camera
- VCC100 rev1 (bt848)
- VCC100 rev2 (bt878)

Gallant (www.gallantcom.com) www.minton.com.tw

Models:

- Intervision IV-510 (capture only bt8x8)
- Intervision IV-550 (bt8x8)
- Intervision IV-100 (zoran)
- Intervision IV-1000 (bt8x8)

Asonic (www.asonic.com.cn) (website down)

SkyEye tv 878

Hoontech

878TV/FM

Teppro (www.itcteppro.com.tw)

Models:

- ITC PCITV (Card Ver 1.0) “Teppro TV1/TVFM1 Card”
- ITC PCITV (Card Ver 2.0)
- ITC PCITV (Card Ver 3.0) = “PV-BT878P+ (REV.9D)”
- ITC PCITV (Card Ver 4.0)
- TEPPRO IV-550 (For BT848 Main Chip)
- ITC DSTTV (bt878, satellite)
- ITC VideoMaker (saa7146, StreamMachine sm2110, tv tuner) “PV-SM2210P+ (REV:1C)”

Kworld (www.kworld.com.tw)

PC TV Station:

- KWORLD KW-TV878R TV (no radio)
- KWORLD KW-TV878RF TV (w/ radio)
- KWORLD KW-TVL878RF (low profile)
- KWORLD KW-TV713XRF (saa7134)
MPEG TV Station (same cards as above plus WinDVR Software MPEG en/decoder)
- KWORLD KW-TV878R -Pro TV (no Radio)
- KWORLD KW-TV878RF-Pro TV (w/ Radio)
- KWORLD KW-TV878R -Ultra TV (no Radio)
- KWORLD KW-TV878RF-Ultra TV (w/ Radio)

JTT/ Justy Corp.(<http://www.jtt.ne.jp/>)

JTT-02 (JTT TV) “TV watchmate pro” (bt848)

ADS www.adstech.com

Models:

- Channel Surfer TV (CHX-950)
- Channel Surfer TV+FM (CHX-960FM)

AVEC www.prochips.com

AVEC Intercapture (bt848, tea6320)

NoBrand

TV Excel = Australian Name for "PV-BT878P+ 8E" or "878TV Rev.3_"

Mach www.machspeed.com

Mach TV 878

Eline www.eline-net.com/

Models:

- Eline Vision TVMaster / TVMaster FM (ELV-TVM/ ELV-TVM-FM) = LR26 (bt878)
- Eline Vision TVMaster-2000 (ELV-TVM-2000, ELV-TVM-2000-FM)= LR138 (saa713x)

Spirit

- Spirit TV Tuner/Video Capture Card (bt848)

Boser www.boser.com.tw

Models:

- HS-878 Mini PCI Capture Add-on Card
- HS-879 Mini PCI 3D Audio and Capture Add-on Card (w/ ES1938 Solo-1)

Satelco www.citycom-gmbh.de, www.satelco.de

Models:

- TV-FM =KNC1 saa7134
- Standard PCI (DVB-S) = Technotrend Budget
- Standard PCI (DVB-S) w/ CI
- Satelco Highend PCI (DVB-S) = Technotrend Premium

Sensoray www.sensoray.com

Models:

- Sensoray 311 (PC/104 bus)
- Sensoray 611 (PCI)

CEI (Chartered Electronics Industries Pte Ltd [CEI] [FCC ID HBY])

Models:

- TV Tuner - HBY-33A-RAFFLES Brooktree Bt848KPF + Philips
- TV Tuner MG9910 - HBY33A-TVO CEI + Philips SAA7110 + OKI M548262 + ST STV8438CV
- Primetime TV (ISA)
 - acquired by Singapore Technologies
 - now operating as Chartered Semiconductor Manufacturing
 - Manufacturer of video cards is listed as:
 - * Cogent Electronics Industries [CEI]

AITech

Models:

- Wavewatcher TV (ISA)
- AITech WaveWatcher TV-PCI = can be LR26 (Bt848) or LR50 (BT878)
- WaveWatcher TVR-202 TV/FM Radio Card (ISA)

MAXRON

Maxron MaxTV/FM Radio (KW-TV878-FNT) = Kworld or JW-TV878-FBK

www.ids-imaging.de

Models:

- Falcon Series (capture only)

In USA: <http://www.theimagingsource.com/> - DFG/LC1

www.sknet-web.co.jp

SKnet Monster TV (saa7134)

A-Max www.amaxhk.com (Colormax, Amax, Napa)

APAC Viewcomp 878

Cybertainment

Models:

- CyberMail AV Video Email Kit w/ PCI Capture Card (capture only)
- CyberMail Xtreme

These are Flyvideo

VCR (<http://www.vcrinc.com/>)

Video Catcher 16

Twinhan

Models:

- DST Card/DST-IP (bt878, twinhan asic) VP-1020 - Sold as:
 - KWorld DVBS Satellite TV-Card
 - Powercolor DSTV Satellite Tuner Card
 - Prolink Pixelview DTV2000
 - Provideo PV-911 Digital Satellite TV Tuner Card With Common Interface ?
- DST-CI Card (DVB Satellite) VP-1030
- DCT Card (DVB cable)

MSI

Models:

- MSI [TV@nywhere](#) Tuner Card (MS-8876) (CX23881/883) Not Bt878 compatible.
- MS-8401 DVB-S

Focus www.focusinfo.com

InVideo PCI (bt878)

Sdisilk www.sdisilk.com/

Models:

- SDI Silk 100
- SDI Silk 200 SDI Input Card

www.euresys.com

PICOLO series

PMC/Pace

www.pacecom.co.uk website closed

Mercury www.kobian.com (UK and FR)

Models:

- LR50
- LR138RBG-Rx == LR138

TEC sound

TV-Mate = Zoltrix VP-8482

Though educated googling found: www.techmakers.com

(package and manuals don't have any other manufacturer info) TecSound

Lorenzen www.lorenzen.de

SL DVB-S PCI = Technotrend Budget PCI (su1278 or bsru version)

Origo (.uk) www.origo2000.com

PC TV Card = LR50

I/O Magic www.iomagic.com

PC PVR - Desktop TV Personal Video Recorder DR-PCTV100 = Pinnacle ROB2D-51009464 4.0
+ Cyberlink PowerVCR II

Arowana

TV-Karte / Poso Power TV (?) = Zoltrix VP-8482 (?)

iTVC15 boards

kuroutoshikou.com ITVC15 yuan.com MPG160 PCI TV (Internal PCI MPEG2 encoder card plus TV-tuner)

Asus www.asuscom.com

Models:

- Asus TV Tuner Card 880 NTSC (low profile, cx23880)
- Asus TV (saa7134)

Hoontech

<http://www.hoontech.de/>

- HART Vision 848 (H-ART Vision 848)
- HART Vision 878 (H-Art Vision 878)

Chips used at bttv devices

- all boards:
 - Brooktree Bt848/848A/849/878/879: video capture chip
- Board specific
 - Miro PCTV:
 - * Philips or Temic Tuner
 - Hauppauge Win/TV pci (version 405):
 - * Microchip 24LC02B or Philips 8582E2Y:
 - 256 Byte EEPROM with configuration information
 - I2C 0xa0-0xa1, (24LC02B also responds to 0xa2-0xaf)
 - * Philips SAA5246AGP/E: Videotext decoder chip, I2C 0x22-0x23
 - * TDA9800: sound decoder
 - * Winbond W24257AS-35: 32Kx8 CMOS static RAM (Videotext buffer mem)
 - * 14052B: analog switch for selection of sound source
- PAL:
 - TDA5737: VHF, hyperband and UHF mixer/oscillator for TV and VCR 3-band tuners
 - TSA5522: 1.4 GHz I2C-bus controlled synthesizer, I2C 0xc2-0xc3
- NTSC:
 - TDA5731: VHF, hyperband and UHF mixer/oscillator for TV and VCR 3-band tuners
 - TSA5518: no datasheet available on Philips site
- STB TV pci:
 - ???
 - if you want better support for STB cards send me info! Look at the board! What chips are on it?

Specs

Philips <http://www.Semiconductors.COM/pip/>

Conexant <http://www.conexant.com/>

Micronas <http://www.micronas.com/en/home/index.html>

Thanks

Many thanks to:

- Markus Schroeder <schroedm@uni-duesseldorf.de> for information on the Bt848 and tuner programming and his control program xtv.
- Martin Buck <martin-2.buck@student.uni-ulm.de> for his great Videotext package.
- Gerd Hoffmann for the MSP3400 support and the modular I2C, tuner, ... support.
- MATRIX Vision for giving us 2 cards for free, which made support of single crystal operation possible.
- MIRO for providing a free PCTV card and detailed information about the components on their cards. (E.g. how the tuner type is detected) Without their card I could not have debugged the NTSC mode.
- Hauppauge for telling how the sound input is selected and what components they do and will use on their radio cards. Also many thanks for faxing me the FM1216 data sheet.

Contributors

Michael Chu <mmchu@pobox.com> AverMedia fix and more flexible card recognition

Alan Cox <alan@lxorguk.ukuu.org.uk> Video4Linux interface and 2.1.x kernel adaptation

Chris Kleitsch Hardware I2C

Gerd Hoffmann Radio card (ITT sound processor)

bigfoot <bigfoot@net-way.net>

Ragnar Hojland Espinosa <ragnar@macula.net> ConferenceTV card

- **many more (please mail me if you are missing in this list and would like to be mentioned)**

1.1.6.2 The `cafe_ccic` driver

Author: Jonathan Corbet <corbet@lwn.net>

Introduction

“`cafe_ccic`” is a driver for the Marvell 88ALP01 “cafe” CMOS camera controller. This is the controller found in first-generation OLPC systems, and this driver was written with support from the OLPC project.

Current status: the core driver works. It can generate data in YUV422, RGB565, and RGB444 formats. (Anybody looking at the code will see RGB32 as well, but that is a debugging aid which will be removed shortly). VGA and QVGA modes work; CIF is there but the colors remain funky. Only the OV7670 sensor is known to work with this controller at this time.

To try it out: either of these commands will work:

```
$ mplayer tv:// -tv driver=v4l2:width=640:height=480 -nosound
$ mplayer tv:// -tv driver=v4l2:width=640:height=480:outfmt=bgr16 -nosound
```

The “xawtv” utility also works; gqcam does not, for unknown reasons.

Load time options

There are a few load-time options, most of which can be changed after loading via sysfs as well:

- `alloc_bufs_at_load`: Normally, the driver will not allocate any DMA buffers until the time comes to transfer data. If this option is set, then worst-case-sized buffers will be allocated at module load time. This option nails down the memory for the life of the module, but perhaps decreases the chances of an allocation failure later on.
- `dma_buf_size`: The size of DMA buffers to allocate. Note that this option is only consulted for load-time allocation; when buffers are allocated at run time, they will be sized appropriately for the current camera settings.
- `n_dma_bufs`: The controller can cycle through either two or three DMA buffers. Normally, the driver tries to use three buffers; on faster systems, however, it will work well with only two.
- `min_buffers`: The minimum number of streaming I/O buffers that the driver will consent to work with. Default is one, but, on slower systems, better behavior with mplayer can be achieved by setting to a higher value (like six).
- `max_buffers`: The maximum number of streaming I/O buffers; default is ten. That number was carefully picked out of a hat and should not be assumed to actually mean much of anything.
- `flip`: If this boolean parameter is set, the sensor will be instructed to invert the video image. Whether it makes sense is determined by how your particular camera is mounted.

1.1.6.3 The cpia2 driver

Authors: Peter Pregler <Peter.Pregler@email.com>, Scott J. Bertin <scottbertin@yahoo.com>, and Jarl Totland <Jarl.Totland@bdc.no> for the original cpia driver, which this one was modelled from.

Introduction

This is a driver for STMicroelectronics’s CPiA2 (second generation Colour Processor Interface ASIC) based cameras. This camera outputs an MJPEG stream at up to vga size. It implements the Video4Linux interface as much as possible. Since the V4L interface does not support compressed formats, only an mjpeg enabled application can be used with the camera. We have modified the gqcam application to view this stream.

The driver is implemented as two kernel modules. The cpia2 module contains the camera functions and the V4L interface. The cpia2_usb module contains usb specific functions. The main reason for this was the size of the module was getting out of hand, so I separated them. It is not likely that there will be a parallel port version.

Features

- Supports cameras with the Vision stv6410 (CIF) and stv6500 (VGA) cmos sensors. I only have the vga sensor, so can't test the other.
- Image formats: VGA, QVGA, CIF, QCIF, and a number of sizes in between. VGA and QVGA are the native image sizes for the VGA camera. CIF is done in the coprocessor by scaling QVGA. All other sizes are done by clipping.
- Palette: YCrCb, compressed with MJPEG.
- Some compression parameters are settable.
- Sensor framerate is adjustable (up to 30 fps CIF, 15 fps VGA).
- Adjust brightness, color, contrast while streaming.
- Flicker control settable for 50 or 60 Hz mains frequency.

Making and installing the stv672 driver modules

Requirements

Video4Linux must be either compiled into the kernel or available as a module. Video4Linux2 is automatically detected and made available at compile time.

Setup

Use `modprobe cpia2` to load and `modprobe -r cpia2` to unload. This may be done automatically by your distribution.

Driver options

Option	Description
<code>video_nr</code>	video device to register (0=/dev/video0, etc) range -1 to 64. default is -1 (first available) If you have more than 1 camera, this MUST be -1.
<code>buffer_size</code>	Size for each frame buffer in bytes (default 68k)
<code>num_buffers</code>	Number of frame buffers (1-32, default 3)
<code>alternate</code>	USB Alternate (2-7, default 7)
<code>flicker_freq</code>	Frequency for flicker reduction(50 or 60, default 60)
<code>flicker_mode</code>	0 to disable, or 1 to enable flicker reduction. (default 0). This is only effective if the camera uses a stv0672 coprocessor.

Setting the options

If you are using modules, edit `/etc/modules.conf` and add an options line like this:

```
options cpia2 num_buffers=3 buffer_size=65535
```

If the driver is compiled into the kernel, at boot time specify them like this:

```
cpia2.num_buffers=3 cpia2.buffer_size=65535
```

What buffer size should I use?

The maximum image size depends on the alternate you choose, and the frame rate achieved by the camera. If the compression engine is able to keep up with the frame rate, the maximum image size is given by the table below.

The compression engine starts out at maximum compression, and will increase image quality until it is close to the size in the table. As long as the compression engine can keep up with the frame rate, after a short time the images will all be about the size in the table, regardless of resolution.

At low alternate settings, the compression engine may not be able to compress the image enough and will reduce the frame rate by producing larger images.

The default of 68k should be good for most users. This will handle any alternate at frame rates down to 15fps. For lower frame rates, it may be necessary to increase the buffer size to avoid having frames dropped due to insufficient space.

Alternate	bytes/ms	15fps	30fps
2	128	8533	4267
3	384	25600	12800
4	640	42667	21333
5	768	51200	25600
6	896	59733	29867
7	1023	68200	34100

Table: Image size(bytes)

How many buffers should I use?

For normal streaming, 3 should give the best results. With only 2, it is possible for the camera to finish sending one image just after a program has started reading the other. If this happens, the driver must drop a frame. The exception to this is if you have a heavily loaded machine. In this case use 2 buffers. You are probably not reading at the full frame rate. If the camera can send multiple images before a read finishes, it could overwrite the third buffer before the read finishes, leading to a corrupt image. Single and double buffering have extra checks to avoid overwriting.

Using the camera

We are providing a modified gqcam application to view the output. In order to avoid confusion, here it is called mview. There is also the qx5view program which can also control the lights on the qx5 microscope. MJPEG Tools (<http://mjpeg.sourceforge.net>) can also be used to record from the camera.

1.1.6.4 The cx88 driver

Author: Gerd Hoffmann

This is a v4l2 device driver for the cx2388x chip.

Current status

video

- Works.
- Overlay isn't supported.

audio

- Works. The TV standard detection is made by the driver, as the hardware has bugs to auto-detect.
- audio data dma (i.e. recording without loopback cable to the sound card) is supported via cx88-alsa.

vbi

- Works.

How to add support for new cards

The driver needs some config info for the TV cards. This stuff is in cx88-cards.c. If the driver doesn't work well you likely need a new entry for your card in that file. Check the kernel log (using dmesg) to see whenever the driver knows your card or not. There is a line like this one:

```
cx8800[0]: subsystem: 0070:3400, board: Hauppauge WinTV \
          34xxx models [card=1,autodetected]
```

If your card is listed as "board: UNKNOWN/GENERIC" it is unknown to the driver. What to do then?

- 1) Try upgrading to the latest snapshot, maybe it has been added meanwhile.
- 2) You can try to create a new entry yourself, have a look at cx88-cards.c. If that worked, mail me your changes as unified diff ("diff -u").
- 3) Or you can mail me the config information. We need at least the following information to add the card:
 - the PCI Subsystem ID ("0070:3400" from the line above, "lspci -v" output is fine too).

- the tuner type used by the card. You can try to find one by trial-and-error using the `tuner=<n> insmod` option. If you know which one the card has you can also have a look at the list in `CARDLIST.tuner`

1.1.6.5 The VPBE V4L2 driver design

Functional partitioning

Consists of the following:

1. V4L2 display driver

Implements creation of `video2` and `video3` device nodes and provides v4l2 device interface to manage `VID0` and `VID1` layers.

2. Display controller

Loads up `VENC`, `OSD` and external encoders such as `ths8200`. It provides a set of API calls to V4L2 drivers to set the output/standards in the `VENC` or external sub devices. It also provides a device object to access the services from `OSD` subdevice using sub device ops. The connection of external encoders to `VENC` LCD controller port is done at init time based on default output and standard selection or at run time when application change the output through V4L2 IOCTLs.

When connected to an external encoder, `vpbe` controller is also responsible for setting up the interface between `VENC` and external encoders based on board specific settings (specified in `board-xxx-evm.c`). This allows interfacing external encoders such as `ths8200`. The `setup_if_config()` is implemented for this as well as `configure_venc()` (part of the next patch) API to set timings in `VENC` for a specific display resolution. As of this patch series, the interconnection and enabling and setting of the external encoders is not present, and would be a part of the next patch series.

3. VENC subdevice module

Responsible for setting outputs provided through internal DACs and also setting timings at LCD controller port when external encoders are connected at the port or LCD panel timings required. When external encoder/LCD panel is connected, the timings for a specific standard/preset is retrieved from the board specific table and the values are used to set the timings in `venc` using non-standard timing mode.

Support LCD Panel displays using the `VENC`. For example to support a Logic PD display, it requires setting up the LCD controller port with a set of timings for the resolution supported and setting the dot clock. So we could add the available outputs as a board specific entry (i.e add the "LogicPD" output name to `board-xxx-evm.c`). A table of timings for various LCDs supported can be maintained in the board specific setup file to support various LCD displays. As of this patch a basic driver is present, and this support for external encoders and displays forms a part of the next patch series.

4. OSD module

`OSD` module implements all `OSD` layer management and hardware specific features. The `VPBE` module interacts with the `OSD` for enabling and disabling appropriate features of the `OSD`.

Current status

A fully functional working version of the V4L2 driver is available. This driver has been tested with NTSC and PAL standards and buffer streaming.

1.1.6.6 The Samsung S5P/Exynos4 FIMC driver

Copyright © 2012 - 2013 Samsung Electronics Co., Ltd.

The FIMC (Fully Interactive Mobile Camera) device available in Samsung SoC Application Processors is an integrated camera host interface, color space converter, image resizer and rotator. It's also capable of capturing data from LCD controller (FIMD) through the SoC internal writeback data path. There are multiple FIMC instances in the SoCs (up to 4), having slightly different capabilities, like pixel alignment constraints, rotator availability, LCD writeback support, etc. The driver is located at `drivers/media/platform/exynos4-is` directory.

Supported SoCs

S5PC100 (mem-to-mem only), S5PV210, Exynos4210

Supported features

- camera parallel interface capture (ITU-R.BT601/565);
- camera serial interface capture (MIPI-CSI2);
- memory-to-memory processing (color space conversion, scaling, mirror and rotation);
- dynamic pipeline re-configuration at runtime (re-attachment of any FIMC instance to any parallel video input or any MIPI-CSI front-end);
- runtime PM and system wide suspend/resume

Not currently supported

- LCD writeback input
- per frame clock gating (mem-to-mem)

User space interfaces

Media device interface

The driver supports Media Controller API as defined at *Part IV - Media Controller API*. The media device driver name is "Samsung S5P FIMC".

The purpose of this interface is to allow changing assignment of FIMC instances to the SoC peripheral camera input at runtime and optionally to control internal connections of the MIPI-CSIS device(s) to the FIMC entities.

The media device interface allows to configure the SoC for capturing image data from the sensor through more than one FIMC instance (e.g. for simultaneous viewfinder and still capture setup). Reconfiguration is done by enabling/disabling media links created by the driver during initialization. The internal device topology can be easily discovered through media entity and links enumeration.

Memory-to-memory video node

V4L2 memory-to-memory interface at `/dev/video?` device node. This is standalone video device, it has no media pads. However please note the mem-to-mem and capture video node operation on same FIMC instance is not allowed. The driver detects such cases but the applications should prevent them to avoid an undefined behaviour.

Capture video node

The driver supports V4L2 Video Capture Interface as defined at *Interfaces*.

At the capture and mem-to-mem video nodes only the multi-planar API is supported. For more details see: *Single- and multi-planar APIs*.

Camera capture subdevs

Each FIMC instance exports a sub-device node (`/dev/v4l-subdev?`), a sub-device node is also created per each available and enabled at the platform level MIPI-CSI receiver device (currently up to two).

sysfs

In order to enable more precise camera pipeline control through the sub-device API the driver creates a sysfs entry associated with "s5p-fimc-md" platform device. The entry path is: `/sys/platform/devices/s5p-fimc-md/subdev_conf_mode`.

In typical use case there could be a following capture pipeline configuration: sensor subdev -> mipi-csi subdev -> fimc subdev -> video node

When we configure these devices through sub-device API at user space, the configuration flow must be from left to right, and the video node is configured as last one.

When we don't use sub-device user space API the whole configuration of all devices belonging to the pipeline is done at the video node driver. The sysfs entry allows to instruct the capture node driver not to configure the sub-devices (format, crop), to avoid resetting the subdevs' configuration when the last configuration steps at the video node is performed.

For full sub-device control support (subdevs configured at user space before starting streaming):

```
# echo "sub-dev" > /sys/platform/devices/s5p-fimc-md/subdev_conf_mode
```

For V4L2 video node control only (subdevs configured internally by the host driver):

```
# echo "vid-dev" > /sys/platform/devices/s5p-fimc-md/subdev_conf_mode
```

This is a default option.

5. Device mapping to video and subdev device nodes

There are associated two video device nodes with each device instance in hardware - video capture and mem-to-mem and additionally a subdev node for more precise FIMC capture sub-system control. In addition a separate v4l2 sub-device node is created per each MIPI-CSIS device.

How to find out which /dev/video? or /dev/v4l-subdev? is assigned to which device?

You can either grep through the kernel log to find relevant information, i.e.

```
# dmesg | grep -i fimc
```

(note that udev, if present, might still have rearranged the video nodes),

or retrieve the information from /dev/media? with help of the media-ctl tool:

```
# media-ctl -p
```

7. Build

If the driver is built as a loadable kernel module (`CONFIG_VIDEO_SAMSUNG_S5P_FIMC=m`) two modules are created (in addition to the core v4l2 modules): `s5p-fimc.ko` and optional `s5p-csis.ko` (MIPI-CSI receiver subdev).

1.1.6.7 i.MX Video Capture Driver

Introduction

The Freescale i.MX5/6 contains an Image Processing Unit (IPU), which handles the flow of image frames to and from capture devices and display devices.

For image capture, the IPU contains the following internal subunits:

- Image DMA Controller (IDMAC)
- Camera Serial Interface (CSI)
- Image Converter (IC)
- Sensor Multi-FIFO Controller (SMFC)
- Image Rotator (IRT)
- Video De-Interlacing or Combining Block (VDIC)

The IDMAC is the DMA controller for transfer of image frames to and from memory. Various dedicated DMA channels exist for both video capture and display paths. During transfer, the IDMAC is also capable of vertical image flip, 8x8 block transfer (see IRT description), pixel component re-ordering (for example UYVY to YUYV) within the same colorspace, and packed

<-> planar conversion. The IDMAC can also perform a simple de-interlacing by interweaving even and odd lines during transfer (without motion compensation which requires the VDIC).

The CSI is the backend capture unit that interfaces directly with camera sensors over Parallel, BT.656/1120, and MIPI CSI-2 buses.

The IC handles color-space conversion, resizing (downscaling and upscaling), horizontal flip, and 90/270 degree rotation operations.

There are three independent “tasks” within the IC that can carry out conversions concurrently: pre-process encoding, pre-process viewfinder, and post-processing. Within each task, conversions are split into three sections: downsizing section, main section (upsizing, flip, colorspace conversion, and graphics plane combining), and rotation section.

The IPU time-shares the IC task operations. The time-slice granularity is one burst of eight pixels in the downsizing section, one image line in the main processing section, one image frame in the rotation section.

The SMFC is composed of four independent FIFOs that each can transfer captured frames from sensors directly to memory concurrently via four IDMAC channels.

The IRT carries out 90 and 270 degree image rotation operations. The rotation operation is carried out on 8x8 pixel blocks at a time. This operation is supported by the IDMAC which handles the 8x8 block transfer along with block reordering, in coordination with vertical flip.

The VDIC handles the conversion of interlaced video to progressive, with support for different motion compensation modes (low, medium, and high motion). The deinterlaced output frames from the VDIC can be sent to the IC pre-process viewfinder task for further conversions. The VDIC also contains a Combiner that combines two image planes, with alpha blending and color keying.

In addition to the IPU internal subunits, there are also two units outside the IPU that are also involved in video capture on i.MX:

- MIPI CSI-2 Receiver for camera sensors with the MIPI CSI-2 bus interface. This is a Synopsys DesignWare core.
- Two video multiplexers for selecting among multiple sensor inputs to send to a CSI.

For more info, refer to the latest versions of the i.MX5/6 reference manuals¹ and².

Features

Some of the features of this driver include:

- Many different pipelines can be configured via media controller API, that correspond to the hardware video capture pipelines supported in the i.MX.
- Supports parallel, BT.565, and MIPI CSI-2 interfaces.
- Concurrent independent streams, by configuring pipelines to multiple video capture interfaces using independent entities.
- Scaling, color-space conversion, horizontal and vertical flip, and image rotation via IC task subdevs.

¹ <http://www.nxp.com/assets/documents/data/en/reference-manuals/IMX6DQRM.pdf>

² <http://www.nxp.com/assets/documents/data/en/reference-manuals/IMX6SDLRM.pdf>

- Many pixel formats supported (RGB, packed and planar YUV, partial planar YUV).
- The VDIC subdev supports motion compensated de-interlacing, with three motion compensation modes: low, medium, and high motion. Pipelines are defined that allow sending frames to the VDIC subdev directly from the CSI. There is also support in the future for sending frames to the VDIC from memory buffers via a output/mem2mem devices.
- Includes a Frame Interval Monitor (FIM) that can correct vertical sync problems with the ADV718x video decoders.

Topology

The following shows the media topologies for the i.MX6Q SabreSD and i.MX6Q SabreAuto. Refer to these diagrams in the entity descriptions in the next section.

The i.MX5/6 topologies can differ upstream from the IPUv3 CSI video multiplexers, but the internal IPUv3 topology downstream from there is common to all i.MX5/6 platforms. For example, the SabreSD, with the MIPI CSI-2 OV5640 sensor, requires the i.MX6 MIPI CSI-2 receiver. But the SabreAuto has only the ADV7180 decoder on a parallel bt.656 bus, and therefore does not require the MIPI CSI-2 receiver, so it is missing in its graph.

Entities

imx6-mipi-csi2

This is the MIPI CSI-2 receiver entity. It has one sink pad to receive the MIPI CSI-2 stream (usually from a MIPI CSI-2 camera sensor). It has four source pads, corresponding to the four MIPI CSI-2 demuxed virtual channel outputs. Multiple source pads can be enabled to independently stream from multiple virtual channels.

This entity actually consists of two sub-blocks. One is the MIPI CSI-2 core. This is a Synopsys Designware MIPI CSI-2 core. The other sub-block is a “CSI-2 to IPU gasket”. The gasket acts as a demultiplexer of the four virtual channels streams, providing four separate parallel buses containing each virtual channel that are routed to CSIs or video multiplexers as described below.

On i.MX6 solo/dual-lite, all four virtual channel buses are routed to two video multiplexers. Both CSI0 and CSI1 can receive any virtual channel, as selected by the video multiplexers.

On i.MX6 Quad, virtual channel 0 is routed to IPU1-CSI0 (after selected by a video mux), virtual channels 1 and 2 are hard-wired to IPU1-CSI1 and IPU2-CSI0, respectively, and virtual channel 3 is routed to IPU2-CSI1 (again selected by a video mux).

ipuX_csiY_mux

These are the video multiplexers. They have two or more sink pads to select from either camera sensors with a parallel interface, or from MIPI CSI-2 virtual channels from imx6-mipi-csi2 entity. They have a single source pad that routes to a CSI (ipuX_csiY entities).

On i.MX6 solo/dual-lite, there are two video mux entities. One sits in front of IPU1-CSI0 to select between a parallel sensor and any of the four MIPI CSI-2 virtual channels (a total of five sink pads). The other mux sits in front of IPU1-CSI1, and again has five sink pads to select between a parallel sensor and any of the four MIPI CSI-2 virtual channels.

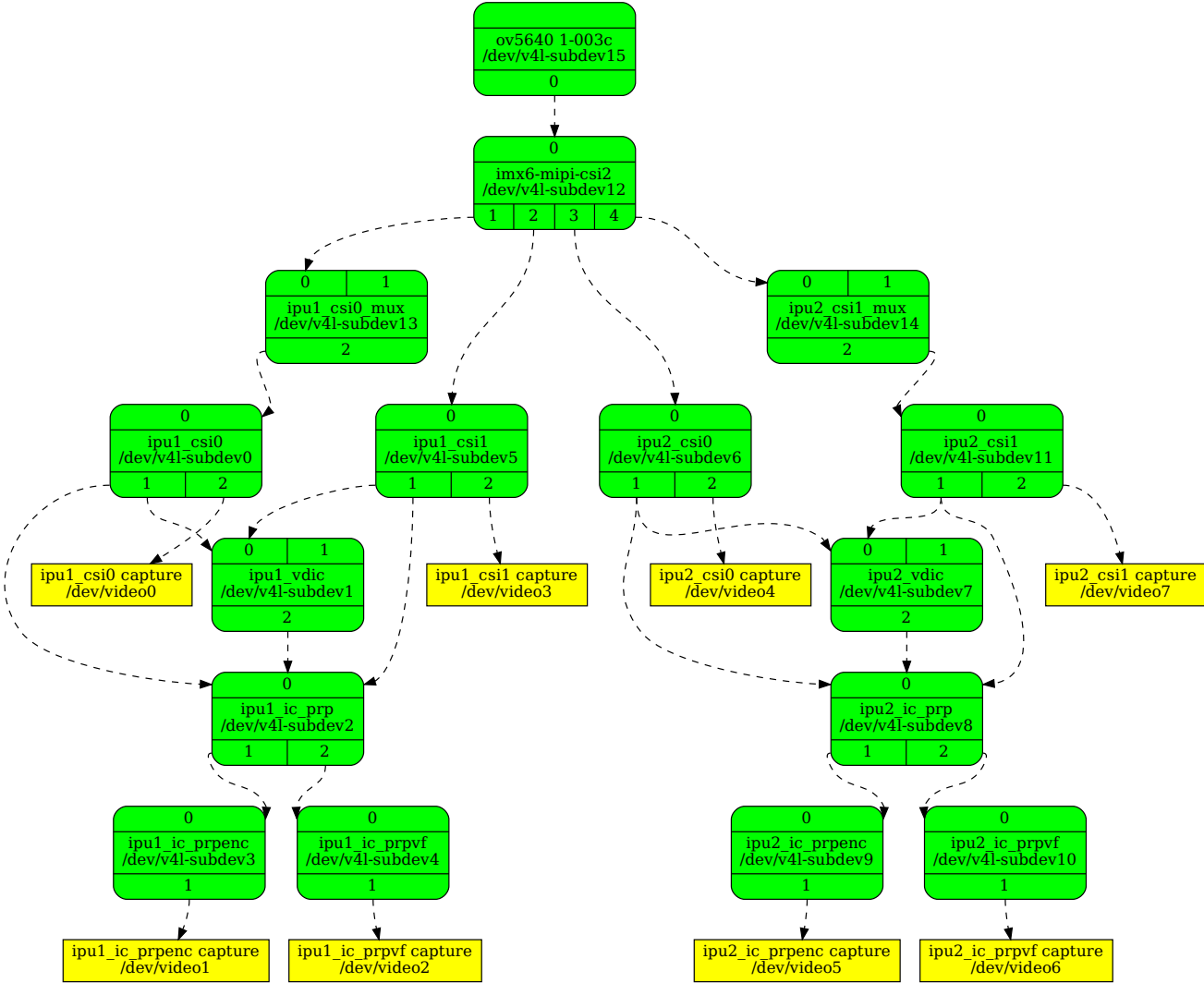


Fig. 1: Media pipeline graph on i.MX6Q SabreSD

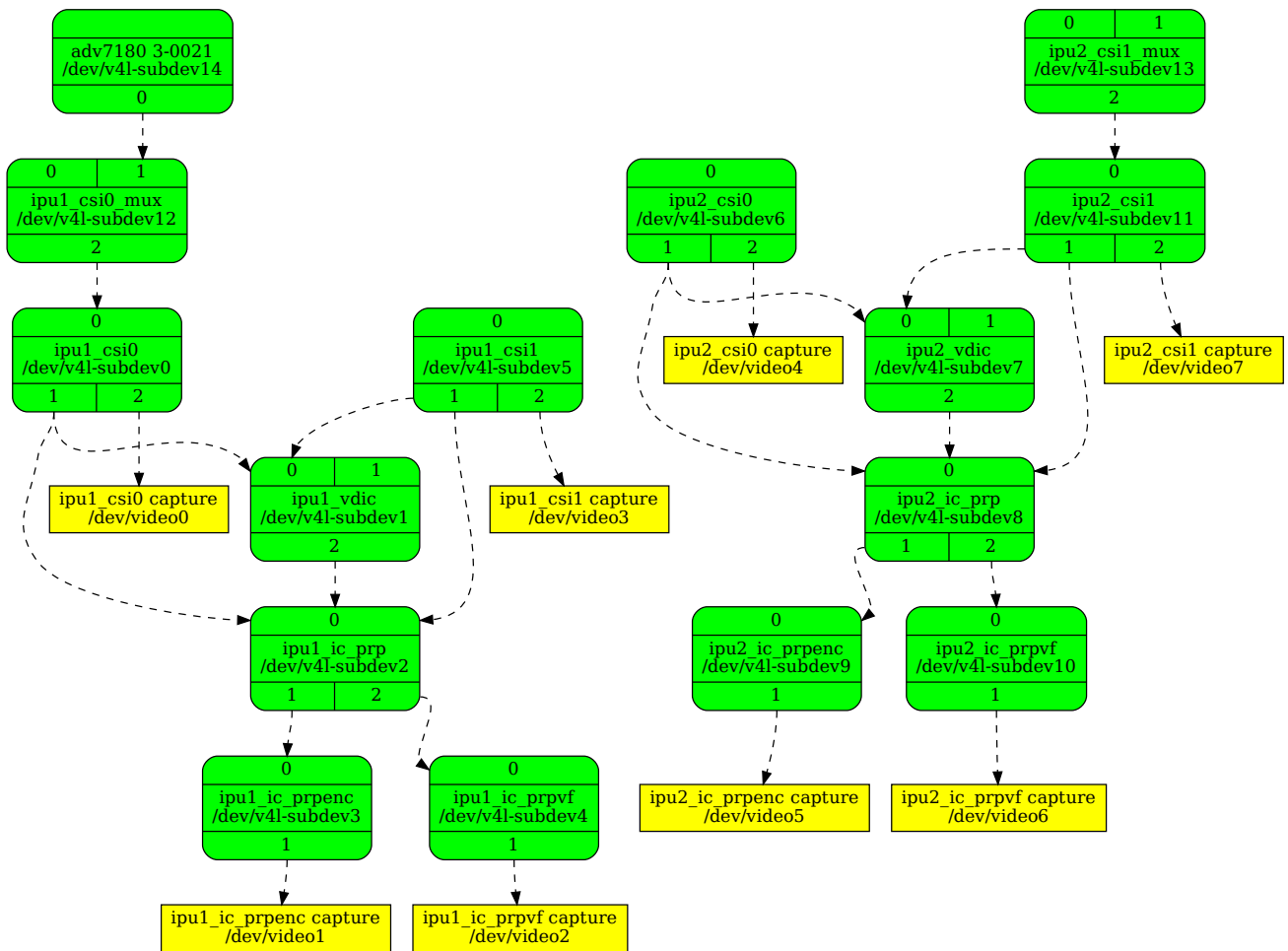


Fig. 2: Media pipeline graph on i.MX6Q SabreAuto

On i.MX6 Quad, there are two video mux entities. One sits in front of IPU1-CSI0 to select between a parallel sensor and MIPI CSI-2 virtual channel 0 (two sink pads). The other mux sits in front of IPU2-CSI1 to select between a parallel sensor and MIPI CSI-2 virtual channel 3 (two sink pads).

ipuX_csiY

These are the CSI entities. They have a single sink pad receiving from either a video mux or from a MIPI CSI-2 virtual channel as described above.

This entity has two source pads. The first source pad can link directly to the ipuX_vdic entity or the ipuX_ic_prp entity, using hardware links that require no IDMAC memory buffer transfer.

When the direct source pad is routed to the ipuX_ic_prp entity, frames from the CSI can be processed by one or both of the IC pre-processing tasks.

When the direct source pad is routed to the ipuX_vdic entity, the VDIC will carry out motion-compensated de-interlace using “high motion” mode (see description of ipuX_vdic entity).

The second source pad sends video frames directly to memory buffers via the SMFC and an IDMAC channel, bypassing IC pre-processing. This source pad is routed to a capture device node, with a node name of the format “ipuX_csiY capture”.

Note that since the IDMAC source pad makes use of an IDMAC channel, pixel reordering within the same colorspace can be carried out by the IDMAC channel. For example, if the CSI sink pad is receiving in UYVY order, the capture device linked to the IDMAC source pad can capture in YUYV order. Also, if the CSI sink pad is receiving a packed YUV format, the capture device can capture a planar YUV format such as YUV420.

The IDMAC channel at the IDMAC source pad also supports simple interweave without motion compensation, which is activated if the source pad’s field type is sequential top-bottom or bottom-top, and the requested capture interface field type is set to interlaced (t-b, b-t, or unqualified interlaced). The capture interface will enforce the same field order as the source pad field order (interlaced-bt if source pad is seq-bt, interlaced-tb if source pad is seq-tb).

For events produced by ipuX_csiY, see ref:imx_api_ipuX_csiY.

Cropping in ipuX_csiY

The CSI supports cropping the incoming raw sensor frames. This is implemented in the ipuX_csiY entities at the sink pad, using the crop selection subdev API.

The CSI also supports fixed divide-by-two downscaling independently in width and height. This is implemented in the ipuX_csiY entities at the sink pad, using the compose selection subdev API.

The output rectangle at the ipuX_csiY source pad is the same as the compose rectangle at the sink pad. So the source pad rectangle cannot be negotiated, it must be set using the compose selection API at sink pad (if /2 downscale is desired, otherwise source pad rectangle is equal to incoming rectangle).

To give an example of crop and /2 downscale, this will crop a 1280x960 input frame to 640x480, and then /2 downscale in both dimensions to 320x240 (assumes ipu1_csi0 is linked to ipu1_csi0_mux):

```
media-ctl -V "'ipu1_csi0_mux':2[fmt:UYVY2X8/1280x960]"
media-ctl -V "'ipu1_csi0':0[crop:(0,0)/640x480]"
media-ctl -V "'ipu1_csi0':0[compose:(0,0)/320x240]"
```

Frame Skipping in ipuX_csiY

The CSI supports frame rate decimation, via frame skipping. Frame rate decimation is specified by setting the frame intervals at sink and source pads. The ipuX_csiY entity then applies the best frame skip setting to the CSI to achieve the desired frame rate at the source pad.

The following example reduces an assumed incoming 60 Hz frame rate by half at the IDMAC output source pad:

```
media-ctl -V "'ipu1_csi0':0[fmt:UYVY2X8/640x480@1/60]"
media-ctl -V "'ipu1_csi0':2[fmt:UYVY2X8/640x480@1/30]"
```

Frame Interval Monitor in ipuX_csiY

See ref:*imx_api_FIM*.

ipuX_vdic

The VDIC carries out motion compensated de-interlacing, with three motion compensation modes: low, medium, and high motion. The mode is specified with the menu control V4L2_CID_DEINTERLACING_MODE. The VDIC has two sink pads and a single source pad.

The direct sink pad receives from an ipuX_csiY direct pad. With this link the VDIC can only operate in high motion mode.

When the IDMAC sink pad is activated, it receives from an output or mem2mem device node. With this pipeline, the VDIC can also operate in low and medium modes, because these modes require receiving frames from memory buffers. Note that an output or mem2mem device is not implemented yet, so this sink pad currently has no links.

The source pad routes to the IC pre-processing entity ipuX_ic_prp.

ipuX_ic_prp

This is the IC pre-processing entity. It acts as a router, routing data from its sink pad to one or both of its source pads.

This entity has a single sink pad. The sink pad can receive from the ipuX_csiY direct pad, or from ipuX_vdic.

This entity has two source pads. One source pad routes to the pre-process encode task entity (ipuX_ic_prpenc), the other to the pre-process viewfinder task entity (ipuX_ic_prpvf). Both source pads can be activated at the same time if the sink pad is receiving from ipuX_csiY. Only the source pad to the pre-process viewfinder task entity can be activated if the sink pad is receiving from ipuX_vdic (frames from the VDIC can only be processed by the pre-process viewfinder task).

ipuX_ic_prpenc

This is the IC pre-processing encode entity. It has a single sink pad from ipuX_ic_prp, and a single source pad. The source pad is routed to a capture device node, with a node name of the format "ipuX_ic_prpenc capture".

This entity performs the IC pre-process encode task operations: color-space conversion, resizing (downscaling and upscaling), horizontal and vertical flip, and 90/270 degree rotation. Flip and rotation are provided via standard V4L2 controls.

Like the ipuX_csiY IDMAC source, this entity also supports simple de-interlace without motion compensation, and pixel reordering.

ipuX_ic_prpvf

This is the IC pre-processing viewfinder entity. It has a single sink pad from ipuX_ic_prp, and a single source pad. The source pad is routed to a capture device node, with a node name of the format "ipuX_ic_prpvf capture".

This entity is identical in operation to ipuX_ic_prpenc, with the same resizing and CSC operations and flip/rotation controls. It will receive and process de-interlaced frames from the ipuX_vdic if ipuX_ic_prp is receiving from ipuX_vdic.

Like the ipuX_csiY IDMAC source, this entity supports simple interweaving without motion compensation. However, note that if the ipuX_vdic is included in the pipeline (ipuX_ic_prp is receiving from ipuX_vdic), it's not possible to use interweave in ipuX_ic_prpvf, since the ipuX_vdic has already carried out de-interlacing (with motion compensation) and therefore the field type output from ipuX_vdic can only be none (progressive).

Capture Pipelines

The following describe the various use-cases supported by the pipelines.

The links shown do not include the backend sensor, video mux, or mipi csi-2 receiver links. This depends on the type of sensor interface (parallel or mipi csi-2). So these pipelines begin with:

```
sensor -> ipuX_csiY_mux -> ...
```

for parallel sensors, or:

```
sensor -> imx6-mipi-csi2 -> (ipuX_csiY_mux) -> ...
```

for mipi csi-2 sensors. The imx6-mipi-csi2 receiver may need to route to the video mux (ipuX_csiY_mux) before sending to the CSI, depending on the mipi csi-2 virtual channel, hence ipuX_csiY_mux is shown in parenthesis.

Unprocessed Video Capture:

Send frames directly from sensor to camera device interface node, with no conversions, via ipuX_csiY IDMAC source pad:

```
-> ipuX_csiY:2 -> ipuX_csiY capture
```

IC Direct Conversions:

This pipeline uses the preprocess encode entity to route frames directly from the CSI to the IC, to carry out scaling up to 1024x1024 resolution, CSC, flipping, and image rotation:

```
-> ipuX_csiY:1 -> 0:ipuX_ic_prp:1 -> 0:ipuX_ic_prpenc:1 -> ipuX_ic_prpenc capture
```

Motion Compensated De-interlace:

This pipeline routes frames from the CSI direct pad to the VDIC entity to support motion-compensated de-interlacing (high motion mode only), scaling up to 1024x1024, CSC, flip, and rotation:

```
-> ipuX_csiY:1 -> 0:ipuX_vdic:2 -> 0:ipuX_ic_prp:2 -> 0:ipuX_ic_prpvf:1 -> ipuX_ic_prpvf capture
```

Usage Notes

To aid in configuration and for backward compatibility with V4L2 applications that access controls only from video device nodes, the capture device interfaces inherit controls from the active entities in the current pipeline, so controls can be accessed either directly from the subdev or from the active capture device interface. For example, the FIM controls are available either from the ipuX_csiY subdevs or from the active capture device.

The following are specific usage notes for the Sabre* reference boards:

i.MX6Q SabreLite with OV5642 and OV5640

This platform requires the OmniVision OV5642 module with a parallel camera interface, and the OV5640 module with a MIPI CSI-2 interface. Both modules are available from Boundary Devices:

- https://boundarydevices.com/product/nit6x_5mp
- https://boundarydevices.com/product/nit6x_5mp_mipi

Note that if only one camera module is available, the other sensor node can be disabled in the device tree.

The OV5642 module is connected to the parallel bus input on the i.MX internal video mux to IPU1 CSI0. It's i2c bus connects to i2c bus 2.

The MIPI CSI-2 OV5640 module is connected to the i.MX internal MIPI CSI-2 receiver, and the four virtual channel outputs from the receiver are routed as follows: vc0 to the IPU1 CSI0 mux, vc1 directly to IPU1 CSI1, vc2 directly to IPU2 CSI0, and vc3 to the IPU2 CSI1 mux. The

OV5640 is also connected to i2c bus 2 on the SabreLite, therefore the OV5642 and OV5640 must not share the same i2c slave address.

The following basic example configures unprocessed video capture pipelines for both sensors. The OV5642 is routed to `ipu1_csi0`, and the OV5640, transmitting on MIPI CSI-2 virtual channel 1 (which is `imx6-mipi-csi2 pad 2`), is routed to `ipu1_csi1`. Both sensors are configured to output 640x480, and the OV5642 outputs YUYV2X8, the OV5640 UYVY2X8:

```
# Setup links for OV5642
media-ctl -l "'ov5642 1-0042':0 -> 'ipu1_csi0_mux':1[1]"
media-ctl -l "'ipu1_csi0_mux':2 -> 'ipu1_csi0':0[1]"
media-ctl -l "'ipu1_csi0':2 -> 'ipu1_csi0 capture':0[1]"
# Setup links for OV5640
media-ctl -l "'ov5640 1-0040':0 -> 'imx6-mipi-csi2':0[1]"
media-ctl -l "'imx6-mipi-csi2':2 -> 'ipu1_csi1':0[1]"
media-ctl -l "'ipu1_csi1':2 -> 'ipu1_csi1 capture':0[1]"
# Configure pads for OV5642 pipeline
media-ctl -v "'ov5642 1-0042':0 [fmt:YUYV2X8/640x480 field:none]"
media-ctl -v "'ipu1_csi0_mux':2 [fmt:YUYV2X8/640x480 field:none]"
media-ctl -v "'ipu1_csi0':2 [fmt:AYUV32/640x480 field:none]"
# Configure pads for OV5640 pipeline
media-ctl -v "'ov5640 1-0040':0 [fmt:UYVY2X8/640x480 field:none]"
media-ctl -v "'imx6-mipi-csi2':2 [fmt:UYVY2X8/640x480 field:none]"
media-ctl -v "'ipu1_csi1':2 [fmt:AYUV32/640x480 field:none]"
```

Streaming can then begin independently on the capture device nodes “`ipu1_csi0 capture`” and “`ipu1_csi1 capture`”. The `v4l2-ctl` tool can be used to select any supported YUV pixelformat on the capture device nodes, including planar.

i.MX6Q SabreAuto with ADV7180 decoder

On the i.MX6Q SabreAuto, an on-board ADV7180 SD decoder is connected to the parallel bus input on the internal video mux to IPU1 CSI0.

The following example configures a pipeline to capture from the ADV7180 video decoder, assuming NTSC 720x480 input signals, using simple interweave (unconverted and without motion compensation). The `adv7180` must output sequential or alternating fields (field type ‘`seq-bt`’ for NTSC, or ‘`alternate`’):

```
# Setup links
media-ctl -l "'adv7180 3-0021':0 -> 'ipu1_csi0_mux':1[1]"
media-ctl -l "'ipu1_csi0_mux':2 -> 'ipu1_csi0':0[1]"
media-ctl -l "'ipu1_csi0':2 -> 'ipu1_csi0 capture':0[1]"
# Configure pads
media-ctl -v "'adv7180 3-0021':0 [fmt:UYVY2X8/720x480 field:seq-bt]"
media-ctl -v "'ipu1_csi0_mux':2 [fmt:UYVY2X8/720x480]"
media-ctl -v "'ipu1_csi0':2 [fmt:AYUV32/720x480]"
# Configure "ipu1_csi0 capture" interface (assumed at /dev/video4)
v4l2-ctl -d4 --set-fmt-video=field=interlaced_bt
```

Streaming can then begin on `/dev/video4`. The `v4l2-ctl` tool can also be used to select any supported YUV pixelformat on `/dev/video4`.

This example configures a pipeline to capture from the ADV7180 video decoder, assuming PAL 720x576 input signals, with Motion Compensated de-interlacing. The `adv7180` must output

sequential or alternating fields (field type 'seq-tb' for PAL, or 'alternate').

```
# Setup links
media-ctl -l "'adv7180 3-0021':0 -> 'ipu1_csi0_mux':1[1]"
media-ctl -l "'ipu1_csi0_mux':2 -> 'ipu1_csi0':0[1]"
media-ctl -l "'ipu1_csi0':1 -> 'ipu1_vdic':0[1]"
media-ctl -l "'ipu1_vdic':2 -> 'ipu1_ic_prp':0[1]"
media-ctl -l "'ipu1_ic_prp':2 -> 'ipu1_ic_prpvf':0[1]"
media-ctl -l "'ipu1_ic_prpvf':1 -> 'ipu1_ic_prpvf capture':0[1]"
# Configure pads
media-ctl -V "'adv7180 3-0021':0 [fmt:UYVY2X8/720x576 field:seq-tb]"
media-ctl -V "'ipu1_csi0_mux':2 [fmt:UYVY2X8/720x576]"
media-ctl -V "'ipu1_csi0':1 [fmt:AYUV32/720x576]"
media-ctl -V "'ipu1_vdic':2 [fmt:AYUV32/720x576 field:none]"
media-ctl -V "'ipu1_ic_prp':2 [fmt:AYUV32/720x576 field:none]"
media-ctl -V "'ipu1_ic_prpvf':1 [fmt:AYUV32/720x576 field:none]"
# Configure "ipu1_ic_prpvf capture" interface (assumed at /dev/video2)
v4l2-ctl -d2 --set-fmt-video=field=none
```

Streaming can then begin on /dev/video2. The v4l2-ctl tool can also be used to select any supported YUV pixelformat on /dev/video2.

This platform accepts Composite Video analog inputs to the ADV7180 on Ain1 (connector J42).

i.MX6DL SabreAuto with ADV7180 decoder

On the i.MX6DL SabreAuto, an on-board ADV7180 SD decoder is connected to the parallel bus input on the internal video mux to IPU1 CSI0.

The following example configures a pipeline to capture from the ADV7180 video decoder, assuming NTSC 720x480 input signals, using simple interweave (unconverted and without motion compensation). The adv7180 must output sequential or alternating fields (field type 'seq-bt' for NTSC, or 'alternate'):

```
# Setup links
media-ctl -l "'adv7180 4-0021':0 -> 'ipu1_csi0_mux':4[1]"
media-ctl -l "'ipu1_csi0_mux':5 -> 'ipu1_csi0':0[1]"
media-ctl -l "'ipu1_csi0':2 -> 'ipu1_csi0 capture':0[1]"
# Configure pads
media-ctl -V "'adv7180 4-0021':0 [fmt:UYVY2X8/720x480 field:seq-bt]"
media-ctl -V "'ipu1_csi0_mux':5 [fmt:UYVY2X8/720x480]"
media-ctl -V "'ipu1_csi0':2 [fmt:AYUV32/720x480]"
# Configure "ipu1_csi0 capture" interface (assumed at /dev/video0)
v4l2-ctl -d0 --set-fmt-video=field=interlaced_bt
```

Streaming can then begin on /dev/video0. The v4l2-ctl tool can also be used to select any supported YUV pixelformat on /dev/video0.

This example configures a pipeline to capture from the ADV7180 video decoder, assuming PAL 720x576 input signals, with Motion Compensated de-interlacing. The adv7180 must output sequential or alternating fields (field type 'seq-tb' for PAL, or 'alternate').

```
# Setup links
media-ctl -l "'adv7180 4-0021':0 -> 'ipu1_csi0_mux':4[1]"
media-ctl -l "'ipu1_csi0_mux':5 -> 'ipu1_csi0':0[1]"
media-ctl -l "'ipu1_csi0':1 -> 'ipu1_vdic':0[1]"
```

```
media-ctl -l "'ipu1_vdic':2 -> 'ipu1_ic_prp':0[1]"
media-ctl -l "'ipu1_ic_prp':2 -> 'ipu1_ic_prpvf':0[1]"
media-ctl -l "'ipu1_ic_prpvf':1 -> 'ipu1_ic_prpvf capture':0[1]"
# Configure pads
media-ctl -V "'adv7180 4-0021':0 [fmt:UYVY2X8/720x576 field:seq-tb]"
media-ctl -V "'ipu1_csi0_mux':5 [fmt:UYVY2X8/720x576]"
media-ctl -V "'ipu1_csi0':1 [fmt:AYUV32/720x576]"
media-ctl -V "'ipu1_vdic':2 [fmt:AYUV32/720x576 field:none]"
media-ctl -V "'ipu1_ic_prp':2 [fmt:AYUV32/720x576 field:none]"
media-ctl -V "'ipu1_ic_prpvf':1 [fmt:AYUV32/720x576 field:none]"
# Configure "ipu1_ic_prpvf capture" interface (assumed at /dev/video2)
v4l2-ctl -d2 --set-fmt-video=field=none
```

Streaming can then begin on `/dev/video2`. The `v4l2-ctl` tool can also be used to select any supported YUV pixelformat on `/dev/video2`.

This platform accepts Composite Video analog inputs to the ADV7180 on Ain1 (connector J42).

i.MX6Q SabreSD with MIPI CSI-2 OV5640

Similarly to i.MX6Q SabreLite, the i.MX6Q SabreSD supports a parallel interface OV5642 module on IPU1 CSI0, and a MIPI CSI-2 OV5640 module. The OV5642 connects to i2c bus 1 and the OV5640 to i2c bus 2.

The device tree for SabreSD includes OF graphs for both the parallel OV5642 and the MIPI CSI-2 OV5640, but as of this writing only the MIPI CSI-2 OV5640 has been tested, so the OV5642 node is currently disabled. The OV5640 module connects to MIPI connector J5. The NXP part number for the OV5640 module that connects to the SabreSD board is H120729.

The following example configures unprocessed video capture pipeline to capture from the OV5640, transmitting on MIPI CSI-2 virtual channel 0:

```
# Setup links
media-ctl -l "'ov5640 1-003c':0 -> 'imx6-mipi-csi2':0[1]"
media-ctl -l "'imx6-mipi-csi2':1 -> 'ipu1_csi0_mux':0[1]"
media-ctl -l "'ipu1_csi0_mux':2 -> 'ipu1_csi0':0[1]"
media-ctl -l "'ipu1_csi0':2 -> 'ipu1_csi0 capture':0[1]"
# Configure pads
media-ctl -V "'ov5640 1-003c':0 [fmt:UYVY2X8/640x480]"
media-ctl -V "'imx6-mipi-csi2':1 [fmt:UYVY2X8/640x480]"
media-ctl -V "'ipu1_csi0_mux':0 [fmt:UYVY2X8/640x480]"
media-ctl -V "'ipu1_csi0':0 [fmt:AYUV32/640x480]"
```

Streaming can then begin on “`ipu1_csi0 capture`” node. The `v4l2-ctl` tool can be used to select any supported pixelformat on the capture device node.

To determine what is the `/dev/video` node correspondent to “`ipu1_csi0 capture`”:

```
media-ctl -e "ipu1_csi0 capture"
/dev/video0
```

`/dev/video0` is the streaming element in this case.

Starting the streaming via `v4l2-ctl`:


```
v4l2-ctl --stream-mmap -d /dev/video0
```

Starting the streaming via Gstreamer and sending the content to the display:

```
gst-launch-1.0 v4l2src device=/dev/video0 ! kmssink
```

The following example configures a direct conversion pipeline to capture from the OV5640, transmitting on MIPI CSI-2 virtual channel 0. It also shows colorspace conversion and scaling at IC output.

```
# Setup links
media-ctl -l "'ov5640 1-003c':0 -> 'imx6-mipi-csi2':0[1]"
media-ctl -l "'imx6-mipi-csi2':1 -> 'ipu1_csi0_mux':0[1]"
media-ctl -l "'ipu1_csi0_mux':2 -> 'ipu1_csi0':0[1]"
media-ctl -l "'ipu1_csi0':1 -> 'ipu1_ic_prp':0[1]"
media-ctl -l "'ipu1_ic_prp':1 -> 'ipu1_ic_prpenc':0[1]"
media-ctl -l "'ipu1_ic_prpenc':1 -> 'ipu1_ic_prpenc capture':0[1]"
# Configure pads
media-ctl -v "'ov5640 1-003c':0 [fmt:UYVY2X8/640x480]"
media-ctl -v "'imx6-mipi-csi2':1 [fmt:UYVY2X8/640x480]"
media-ctl -v "'ipu1_csi0_mux':2 [fmt:UYVY2X8/640x480]"
media-ctl -v "'ipu1_csi0':1 [fmt:AYUV32/640x480]"
media-ctl -v "'ipu1_ic_prp':1 [fmt:AYUV32/640x480]"
media-ctl -v "'ipu1_ic_prpenc':1 [fmt:ARGB8888_1X32/800x600]"
# Set a format at the capture interface
v4l2-ctl -d /dev/video1 --set-fmt-video=pixelformat=RGB3
```

Streaming can then begin on “ipu1_ic_prpenc capture” node.

To determine what is the /dev/video node correspondent to “ipu1_ic_prpenc capture”:

```
media-ctl -e "ipu1_ic_prpenc capture"
/dev/video1
```

/dev/video1 is the streaming element in this case.

Starting the streaming via v4l2-ctl:

```
v4l2-ctl --stream-mmap -d /dev/video1
```

Starting the streaming via Gstreamer and sending the content to the display:

```
gst-launch-1.0 v4l2src device=/dev/video1 ! kmssink
```

Known Issues

1. When using 90 or 270 degree rotation control at capture resolutions near the IC resizer limit of 1024x1024, and combined with planar pixel formats (YUV420, YUV422p), frame capture will often fail with no end-of-frame interrupts from the IDMAC channel. To work around this, use lower resolution and/or packed formats (YUYV, RGB3, etc.) when 90 or 270 rotations are needed.

File list

drivers/staging/media/imx/ include/media/imx.h include/linux/imx-media.h

References

Authors

- Steve Longerbeam <steve_longerbeam@mentor.com>
- Philipp Zabel <kernel@pengutronix.de>
- Russell King <linux@armlinux.org.uk>

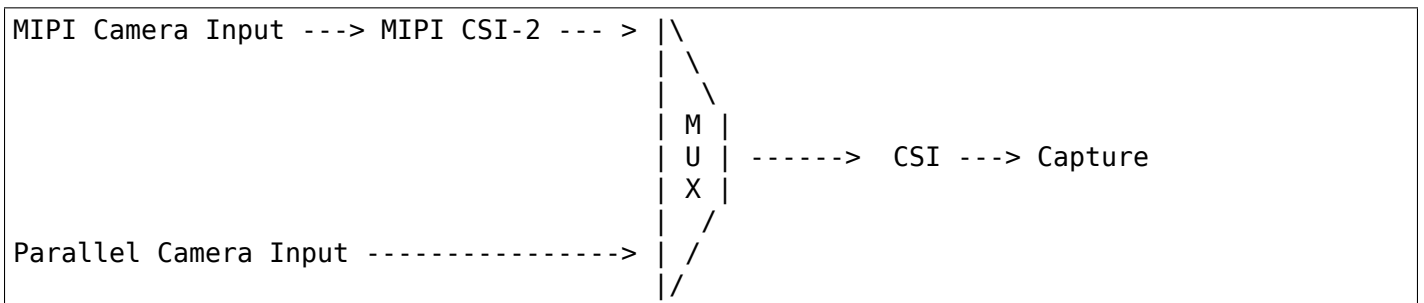
Copyright (C) 2012-2017 Mentor Graphics Inc.

1.1.6.8 i.MX7 Video Capture Driver

Introduction

The i.MX7 contrary to the i.MX5/6 family does not contain an Image Processing Unit (IPU); because of that the capabilities to perform operations or manipulation of the capture frames are less feature rich.

For image capture the i.MX7 has three units: - CMOS Sensor Interface (CSI) - Video Multiplexer - MIPI CSI-2 Receiver



For additional information, please refer to the latest versions of the i.MX7 reference manual¹.

Entities

imx7-mipi-csi2

This is the MIPI CSI-2 receiver entity. It has one sink pad to receive the pixel data from MIPI CSI-2 camera sensor. It has one source pad, corresponding to the virtual channel 0. This module is compliant to previous version of Samsung D-phy, and supports two D-PHY Rx Data lanes.

¹ <https://www.nxp.com/docs/en/reference-manual/IMX7SRM.pdf>

csi-mux

This is the video multiplexer. It has two sink pads to select from either camera sensor with a parallel interface or from MIPI CSI-2 virtual channel 0. It has a single source pad that routes to the CSI.

csi

The CSI enables the chip to connect directly to external CMOS image sensor. CSI can interface directly with Parallel and MIPI CSI-2 buses. It has 256 x 64 FIFO to store received image pixel data and embedded DMA controllers to transfer data from the FIFO through AHB bus.

This entity has one sink pad that receives from the csi-mux entity and a single source pad that routes video frames directly to memory buffers. This pad is routed to a capture device node.

Usage Notes

To aid in configuration and for backward compatibility with V4L2 applications that access controls only from video device nodes, the capture device interfaces inherit controls from the active entities in the current pipeline, so controls can be accessed either directly from the subdev or from the active capture device interface. For example, the sensor controls are available either from the sensor subdevs or from the active capture device.

Warp7 with OV2680

On this platform an OV2680 MIPI CSI-2 module is connected to the internal MIPI CSI-2 receiver. The following example configures a video capture pipeline with an output of 800x600, and BGGR 10 bit bayer format:

```
# Setup links
media-ctl -l "'ov2680 1-0036':0 -> 'imx7-mipi-csis.0':0[1]"
media-ctl -l "'imx7-mipi-csis.0':1 -> 'csi-mux':1[1]"
media-ctl -l "'csi-mux':2 -> 'csi':0[1]"
media-ctl -l "'csi':1 -> 'csi capture':0[1]"

# Configure pads for pipeline
media-ctl -v "'ov2680 1-0036':0 [fmt:SBGGR10_1X10/800x600 field:none]"
media-ctl -v "'csi-mux':1 [fmt:SBGGR10_1X10/800x600 field:none]"
media-ctl -v "'csi-mux':2 [fmt:SBGGR10_1X10/800x600 field:none]"
media-ctl -v "'imx7-mipi-csis.0':0 [fmt:SBGGR10_1X10/800x600 field:none]"
media-ctl -v "'csi':0 [fmt:SBGGR10_1X10/800x600 field:none]"
```

After this streaming can start. The v4l2-ctl tool can be used to select any of the resolutions supported by the sensor.

```
# media-ctl -p
Media controller API version 5.2.0

Media device information
-----
driver            imx7-csi
```

```

model          imx-media
serial
bus info
hw revision    0x0
driver version 5.2.0

Device topology
- entity 1: csi (2 pads, 2 links)
    type V4L2 subdev subtype Unknown flags 0
    device node name /dev/v4l-subdev0
    pad0: Sink
        [fmt:SBGGR10_1X10/800x600 field:none colorspace:srgb xfer:srgb
        ↪ycbcr:601 quantization:full-range]
        <- "csi-mux":2 [ENABLED]
    pad1: Source
        [fmt:SBGGR10_1X10/800x600 field:none colorspace:srgb xfer:srgb
        ↪ycbcr:601 quantization:full-range]
        -> "csi capture":0 [ENABLED]

- entity 4: csi capture (1 pad, 1 link)
    type Node subtype V4L flags 0
    device node name /dev/video0
    pad0: Sink
        <- "csi":1 [ENABLED]

- entity 10: csi-mux (3 pads, 2 links)
    type V4L2 subdev subtype Unknown flags 0
    device node name /dev/v4l-subdev1
    pad0: Sink
        [fmt:Y8_1X8/1x1 field:none]
    pad1: Sink
        [fmt:SBGGR10_1X10/800x600 field:none]
        <- "imx7-mipi-csis.0":1 [ENABLED]
    pad2: Source
        [fmt:SBGGR10_1X10/800x600 field:none]
        -> "csi":0 [ENABLED]

- entity 14: imx7-mipi-csis.0 (2 pads, 2 links)
    type V4L2 subdev subtype Unknown flags 0
    device node name /dev/v4l-subdev2
    pad0: Sink
        [fmt:SBGGR10_1X10/800x600 field:none]
        <- "ov2680 1-0036":0 [ENABLED]
    pad1: Source
        [fmt:SBGGR10_1X10/800x600 field:none]
        -> "csi-mux":1 [ENABLED]

- entity 17: ov2680 1-0036 (1 pad, 1 link)
    type V4L2 subdev subtype Sensor flags 0
    device node name /dev/v4l-subdev3
    pad0: Source
        [fmt:SBGGR10_1X10/800x600@1/30 field:none colorspace:srgb]
        -> "imx7-mipi-csis.0":0 [ENABLED]

```

i.MX6ULL-EVK with OV5640

On this platform a parallel OV5640 sensor is connected to the CSI port. The following example configures a video capture pipeline with an output of 640x480 and UYVY8_2X8 format:

```
# Setup links
media-ctl -l "'ov5640 1-003c':0 -> 'csi':0[1]"
media-ctl -l "'csi':1 -> 'csi capture':0[1]"

# Configure pads for pipeline
media-ctl -v -V "'ov5640 1-003c':0 [fmt:UYVY8_2X8/640x480 field:none]"
```

After this streaming can start:

```
gst-launch-1.0 -v v4l2src device=/dev/video1 ! video/x-raw,format=UYVY,width=640,
↳height=480 ! v4l2convert ! fbdevsink
```

```
# media-ctl -p
Media controller API version 5.14.0

Media device information
-----
driver          imx7-csi
model           imx-media
serial
bus info
hw revision     0x0
driver version  5.14.0

Device topology
- entity 1: csi (2 pads, 2 links)
  type V4L2 subdev subtype Unknown flags 0
  device node name /dev/v4l-subdev0
  pad0: Sink
    [fmt:UYVY8_2X8/640x480 field:none colorspace:srgb xfer:srgb ycbcr:601_
↳quantization:full-range]
    <- "ov5640 1-003c":0 [ENABLED,IMMUTABLE]
  pad1: Source
    [fmt:UYVY8_2X8/640x480 field:none colorspace:srgb xfer:srgb ycbcr:601_
↳quantization:full-range]
    -> "csi capture":0 [ENABLED,IMMUTABLE]

- entity 4: csi capture (1 pad, 1 link)
  type Node subtype V4L flags 0
  device node name /dev/video1
  pad0: Sink
    <- "csi":1 [ENABLED,IMMUTABLE]

- entity 10: ov5640 1-003c (1 pad, 1 link)
  type V4L2 subdev subtype Sensor flags 0
  device node name /dev/v4l-subdev1
  pad0: Source
    [fmt:UYVY8_2X8/640x480@1/30 field:none colorspace:srgb xfer:srgb_
↳ycbcr:601 quantization:full-range]
    -> "csi":0 [ENABLED,IMMUTABLE]
```

References

1.1.6.9 Intel Image Processing Unit 3 (IPU3) Imaging Unit (ImgU) driver

Copyright © 2018 Intel Corporation

Introduction

This file documents the Intel IPU3 (3rd generation Image Processing Unit) Imaging Unit drivers located under `drivers/media/pci/intel/ipu3` (CIO2) as well as under `drivers/staging/media/ipu3` (ImgU).

The Intel IPU3 found in certain Kaby Lake (as well as certain Sky Lake) platforms (U/Y processor lines) is made up of two parts namely the Imaging Unit (ImgU) and the CIO2 device (MIPI CSI2 receiver).

The CIO2 device receives the raw Bayer data from the sensors and outputs the frames in a format that is specific to the IPU3 (for consumption by the IPU3 ImgU). The CIO2 driver is available as `drivers/media/pci/intel/ipu3/ipu3-cio2*` and is enabled through the `CONFIG_VIDEO_IPU3_CIO2` config option.

The Imaging Unit (ImgU) is responsible for processing images captured by the IPU3 CIO2 device. The ImgU driver sources can be found under `drivers/staging/media/ipu3` directory. The driver is enabled through the `CONFIG_VIDEO_IPU3_IMGU` config option.

The two driver modules are named `ipu3_csi2` and `ipu3_imgu`, respectively.

The drivers has been tested on Kaby Lake platforms (U/Y processor lines).

Both of the drivers implement V4L2, Media Controller and V4L2 sub-device interfaces. The IPU3 CIO2 driver supports camera sensors connected to the CIO2 MIPI CSI-2 interfaces through V4L2 sub-device sensor drivers.

CIO2

The CIO2 is represented as a single V4L2 subdev, which provides a V4L2 subdev interface to the user space. There is a video node for each CSI-2 receiver, with a single media controller interface for the entire device.

The CIO2 contains four independent capture channel, each with its own MIPI CSI-2 receiver and DMA engine. Each channel is modelled as a V4L2 sub-device exposed to userspace as a V4L2 sub-device node and has two pads:

Pad	Direction	Purpose
0	sink	MIPI CSI-2 input, connected to the sensor subdev
1	source	Raw video capture, connected to the V4L2 video interface

The V4L2 video interfaces model the DMA engines. They are exposed to userspace as V4L2 video device nodes.

Capturing frames in raw Bayer format

CIO2 MIPI CSI2 receiver is used to capture frames (in packed raw Bayer format) from the raw sensors connected to the CSI2 ports. The captured frames are used as input to the ImgU driver.

Image processing using IPU3 ImgU requires tools such as `raw2pnm`², and `yavta`³ due to the following unique requirements and / or features specific to IPU3.

- The IPU3 CSI2 receiver outputs the captured frames from the sensor in packed raw Bayer format that is specific to IPU3.

- Multiple video nodes have to be operated simultaneously.

Let us take the example of `ov5670` sensor connected to CSI2 port 0, for a 2592x1944 image capture.

Using the media controller APIs, the `ov5670` sensor is configured to send frames in packed raw Bayer format to IPU3 CSI2 receiver.

```
# This example assumes /dev/media0 as the CIO2 media device
export MDEV=/dev/media0

# and that ov5670 sensor is connected to i2c bus 10 with address 0x36
export SDEV=$(media-ctl -d $MDEV -e "ov5670 10-0036")

# Establish the link for the media devices using media-ctl [#f3]_
media-ctl -d $MDEV -l "ov5670:0 -> ipu3-csi2 0:0[1]"

# Set the format for the media devices
media-ctl -d $MDEV -V "ov5670:0 [fmt:SRBG10/2592x1944]"
media-ctl -d $MDEV -V "ipu3-csi2 0:0 [fmt:SRBG10/2592x1944]"
media-ctl -d $MDEV -V "ipu3-csi2 0:1 [fmt:SRBG10/2592x1944]"
```

Once the media pipeline is configured, desired sensor specific settings (such as exposure and gain settings) can be set, using the `yavta` tool.

e.g

```
yavta -w 0x009e0903 444 $SDEV
yavta -w 0x009e0913 1024 $SDEV
yavta -w 0x009e0911 2046 $SDEV
```

Once the desired sensor settings are set, frame captures can be done as below.

e.g

```
yavta --data-prefix -u -c10 -n5 -I -s2592x1944 --file=/tmp/frame-#.bin \
-f IPU3_SRGB10 $(media-ctl -d $MDEV -e "ipu3-cio2 0")
```

With the above command, 10 frames are captured at 2592x1944 resolution, with sRGB10 format and output as IPU3_SRGB10 format.

² <https://github.com/intel/nvt>

³ <http://git.ideasonboard.org/yavta.git>

The captured frames are available as `/tmp/frame-#.bin` files.

ImgU

The ImgU is represented as two V4L2 subdevs, each of which provides a V4L2 subdev interface to the user space.

Each V4L2 subdev represents a pipe, which can support a maximum of 2 streams. This helps to support advanced camera features like Continuous View Finder (CVF) and Snapshot During Video(SDV).

The ImgU contains two independent pipes, each modelled as a V4L2 sub-device exposed to userspace as a V4L2 sub-device node.

Each pipe has two sink pads and three source pads for the following purpose:

Pad	Direction	Purpose
0	sink	Input raw video stream
1	sink	Processing parameters
2	source	Output processed video stream
3	source	Output viewfinder video stream
4	source	3A statistics

Each pad is connected to a corresponding V4L2 video interface, exposed to userspace as a V4L2 video device node.

Device operation

With ImgU, once the input video node ("`ipu3-imgu 0/1`":0, in `<entity>:<pad-number>` format) is queued with buffer (in packed raw Bayer format), ImgU starts processing the buffer and produces the video output in YUV format and statistics output on respective output nodes. The driver is expected to have buffers ready for all of parameter, output and statistics nodes, when input video node is queued with buffer.

At a minimum, all of input, main output, 3A statistics and viewfinder video nodes should be enabled for IPU3 to start image processing.

Each ImgU V4L2 subdev has the following set of video nodes.

input, output and viewfinder video nodes

The frames (in packed raw Bayer format specific to the IPU3) received by the input video node is processed by the IPU3 Imaging Unit and are output to 2 video nodes, with each targeting a different purpose (main output and viewfinder output).

Details on and the Bayer format specific to the IPU3 can be found in `V4L2_PIX_FMT_IPU3_SBGGR10 ('ip3b')`, `V4L2_PIX_FMT_IPU3_SGBRG10 ('ip3g')`, `V4L2_PIX_FMT_IPU3_SGRBG10 ('ip3G')`, `V4L2_PIX_FMT_IPU3_SRGG10 ('ip3r')`.

The driver supports V4L2 Video Capture Interface as defined at *Interfaces*.

Only the multi-planar API is supported. More details can be found at *Single- and multi-planar APIs*.

Parameters video node

The parameters video node receives the ImgU algorithm parameters that are used to configure how the ImgU algorithms process the image.

Details on processing parameters specific to the IPU3 can be found in `V4L2_META_FMT_IPU3_PARAMS ('ip3p')`, `V4L2_META_FMT_IPU3_3A ('ip3s')`.

3A statistics video node

3A statistics video node is used by the ImgU driver to output the 3A (auto focus, auto exposure and auto white balance) statistics for the frames that are being processed by the ImgU to user space applications. User space applications can use this statistics data to compute the desired algorithm parameters for the ImgU.

Configuring the Intel IPU3

The IPU3 ImgU pipelines can be configured using the Media Controller, defined at *Part IV - Media Controller API*.

Running mode and firmware binary selection

ImgU works based on firmware, currently the ImgU firmware support run 2 pipes in time-sharing with single input frame data. Each pipe can run at certain mode - "VIDEO" or "STILL", "VIDEO" mode is commonly used for video frames capture, and "STILL" is used for still frame capture. However, you can also select "VIDEO" to capture still frames if you want to capture images with less system load and power. For "STILL" mode, ImgU will try to use smaller BDS factor and output larger bayer frame for further YUV processing than "VIDEO" mode to get high quality images. Besides, "STILL" mode need XNR3 to do noise reduction, hence "STILL" mode will need more power and memory bandwidth than "VIDEO" mode. TNR will be enabled in "VIDEO" mode and bypassed by "STILL" mode. ImgU is running at "VIDEO" mode by default, the user can use v4l2 control `V4L2_CID_INTEL_IPU3_MODE` (currently defined in `drivers/staging/media/ipu3/include/uapi/intel-ipu3.h`) to query and set the running mode. For user, there is no difference for buffer queueing between the "VIDEO" and "STILL" mode,

mandatory input and main output node should be enabled and buffers need be queued, the statistics and the view-finder queues are optional.

The firmware binary will be selected according to current running mode, such log “using binary if_to_osys_striped ” or “using binary if_to_osys_primary_striped” could be observed if you enable the ImgU dynamic debug, the binary if_to_osys_striped is selected for “VIDEO” and the binary “if_to_osys_primary_striped” is selected for “STILL”.

Processing the image in raw Bayer format

Configuring ImgU V4L2 subdev for image processing

The ImgU V4L2 subdevs have to be configured with media controller APIs to have all the video nodes setup correctly.

Let us take “ipu3-imgu 0” subdev as an example.

```
media-ctl -d $MDEV -r
media-ctl -d $MDEV -l "ipu3-imgu 0 input":0 -> "ipu3-imgu 0":0[1]
media-ctl -d $MDEV -l "ipu3-imgu 0":2 -> "ipu3-imgu 0 output":0[1]
media-ctl -d $MDEV -l "ipu3-imgu 0":3 -> "ipu3-imgu 0 viewfinder":0[1]
media-ctl -d $MDEV -l "ipu3-imgu 0":4 -> "ipu3-imgu 0 3a stat":0[1]
```

Also the pipe mode of the corresponding V4L2 subdev should be set as desired (e.g 0 for video mode or 1 for still mode) through the control id 0x009819a1 as below.

```
yavta -w "0x009819A1 1" /dev/v4l-subdev7
```

Certain hardware blocks in ImgU pipeline can change the frame resolution by cropping or scaling, these hardware blocks include Input Feeder(IF), Bayer Down Scaler (BDS) and Geometric Distortion Correction (GDC). There is also a block which can change the frame resolution - YUV Scaler, it is only applicable to the secondary output.

RAW Bayer frames go through these ImgU pipeline hardware blocks and the final processed image output to the DDR memory.

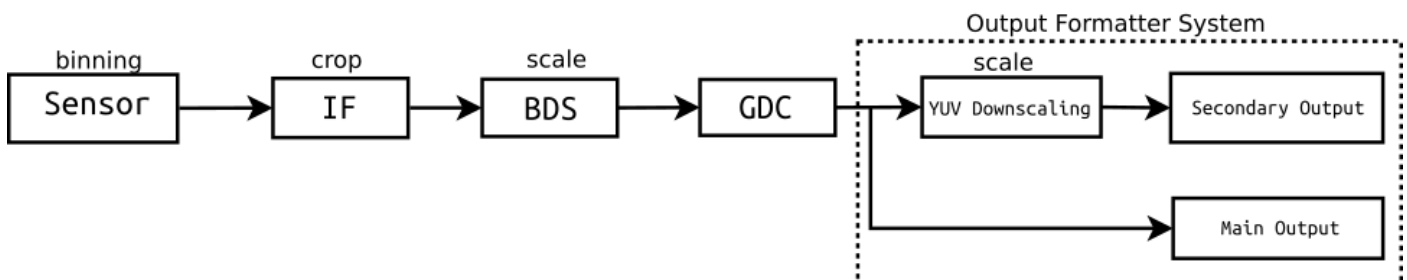


Fig. 3: IPU3 resolution change hardware blocks

Input Feeder

Input Feeder gets the Bayer frame data from the sensor, it can enable cropping of lines and columns from the frame and then store pixels into device’s internal pixel buffer which are ready to readout by following blocks.

Bayer Down Scaler

Bayer Down Scaler is capable of performing image scaling in Bayer domain, the downscale factor can be configured from 1X to 1/4X in each axis with configuration steps of 0.03125 (1/32).

Geometric Distortion Correction

Geometric Distortion Correction is used to perform correction of distortions and image filtering. It needs some extra filter and envelope padding pixels to work, so the input resolution of GDC should be larger than the output resolution.

YUV Scaler

YUV Scaler which similar with BDS, but it is mainly do image down scaling in YUV domain, it can support up to 1/12X down scaling, but it can not be applied to the main output.

The ImgU V4L2 subdev has to be configured with the supported resolutions in all the above hardware blocks, for a given input resolution. For a given supported resolution for an input frame, the Input Feeder, Bayer Down Scaler and GDC blocks should be configured with the supported resolutions as each hardware block has its own alignment requirement.

You must configure the output resolution of the hardware blocks smartly to meet the hardware requirement along with keeping the maximum field of view. The intermediate resolutions can be generated by specific tool -

<https://github.com/intel/intel-ipu3-pipecfg>

This tool can be used to generate intermediate resolutions. More information can be obtained by looking at the following IPU3 ImgU configuration table.

https://chromium.googlesource.com/chromiumos/overlays/board-overlays/+/_/master

Under `baseboard-poppy/media-libs/cros-camera-hal-configs-poppy/files/gcss` directory, `graph_settings_ov5670.xml` can be used as an example.

The following steps prepare the ImgU pipeline for the image processing.

1. The ImgU V4L2 subdev data format should be set by using the `VIDIOC_SUBDEV_S_FMT` on pad 0, using the GDC width and height obtained above.
2. The ImgU V4L2 subdev cropping should be set by using the `VIDIOC_SUBDEV_S_SELECTION` on pad 0, with `V4L2_SEL_TGT_CROP` as the target, using the input feeder height and width.
3. The ImgU V4L2 subdev compositing should be set by using the `VIDIOC_SUBDEV_S_SELECTION` on pad 0, with `V4L2_SEL_TGT_COMPOSE` as the target, using the BDS height and width.

For the ov5670 example, for an input frame with a resolution of 2592x1944 (which is input to the ImgU subdev pad 0), the corresponding resolutions for input feeder, BDS and GDC are 2592x1944, 2592x1944 and 2560x1920 respectively.

Once this is done, the received raw Bayer frames can be input to the ImgU V4L2 subdev as below, using the open source application `v4l2n2`.

For an image captured with 2592x1944⁵ resolution, with desired output resolution as 2560x1920 and viewfinder resolution as 2560x1920, the following `v4l2n` command can be used. This helps process the raw Bayer frames and produces the desired results for the main output image and the viewfinder output, in NV12 format.

⁵ ImgU limitation requires an additional 16x16 for all input resolutions

```
v4l2n --pipe=4 --load=/tmp/frame-#.bin --open=/dev/video4
--fmt=type:VIDEO_OUTPUT_MPLANE,width=2592,height=1944,pixelformat=0X47337069 \
--reqbufs=type:VIDEO_OUTPUT_MPLANE,count:1 --pipe=1 \
--output=/tmp/frames.out --open=/dev/video5 \
--fmt=type:VIDEO_CAPTURE_MPLANE,width=2560,height=1920,pixelformat=NV12 \
--reqbufs=type:VIDEO_CAPTURE_MPLANE,count:1 --pipe=2 \
--output=/tmp/frames.vf --open=/dev/video6 \
--fmt=type:VIDEO_CAPTURE_MPLANE,width=2560,height=1920,pixelformat=NV12 \
--reqbufs=type:VIDEO_CAPTURE_MPLANE,count:1 --pipe=3 --open=/dev/video7 \
--output=/tmp/frames.3A --fmt=type:META_CAPTURE,? \
--reqbufs=count:1,type:META_CAPTURE --pipe=1,2,3,4 --stream=5
```

You can also use `yavta`³ command to do same thing as above:

```
yavta --data-prefix -Bcapture-mplane -c10 -n5 -I -s2592x1944 \
--file=frame-#.out -f NV12 /dev/video5 & \
yavta --data-prefix -Bcapture-mplane -c10 -n5 -I -s2592x1944 \
--file=frame-#.vf -f NV12 /dev/video6 & \
yavta --data-prefix -Bmeta-capture -c10 -n5 -I \
--file=frame-#.3a /dev/video7 & \
yavta --data-prefix -Boutput-mplane -c10 -n5 -I -s2592x1944 \
--file=/tmp/frame-in.cio2 -f IPU3_SGRBG10 /dev/video4
```

where `/dev/video4`, `/dev/video5`, `/dev/video6` and `/dev/video7` devices point to input, output, viewfinder and 3A statistics video nodes respectively.

Converting the raw Bayer image into YUV domain

The processed images after the above step, can be converted to YUV domain as below.

Main output frames

```
raw2pnm -x2560 -y1920 -fNV12 /tmp/frames.out /tmp/frames.out.ppm
```

where 2560x1920 is output resolution, NV12 is the video format, followed by input frame and output PNM file.

Viewfinder output frames

```
raw2pnm -x2560 -y1920 -fNV12 /tmp/frames.vf /tmp/frames.vf.ppm
```

where 2560x1920 is output resolution, NV12 is the video format, followed by input frame and output PNM file.

Example user space code for IPU3

User space code that configures and uses IPU3 is available here.

<https://chromium.googlesource.com/chromiumos/platform/arc-camera/+master/>

The source can be located under hal/intel directory.

Overview of IPU3 pipeline

IPU3 pipeline has a number of image processing stages, each of which takes a set of parameters as input. The major stages of pipelines are shown here:

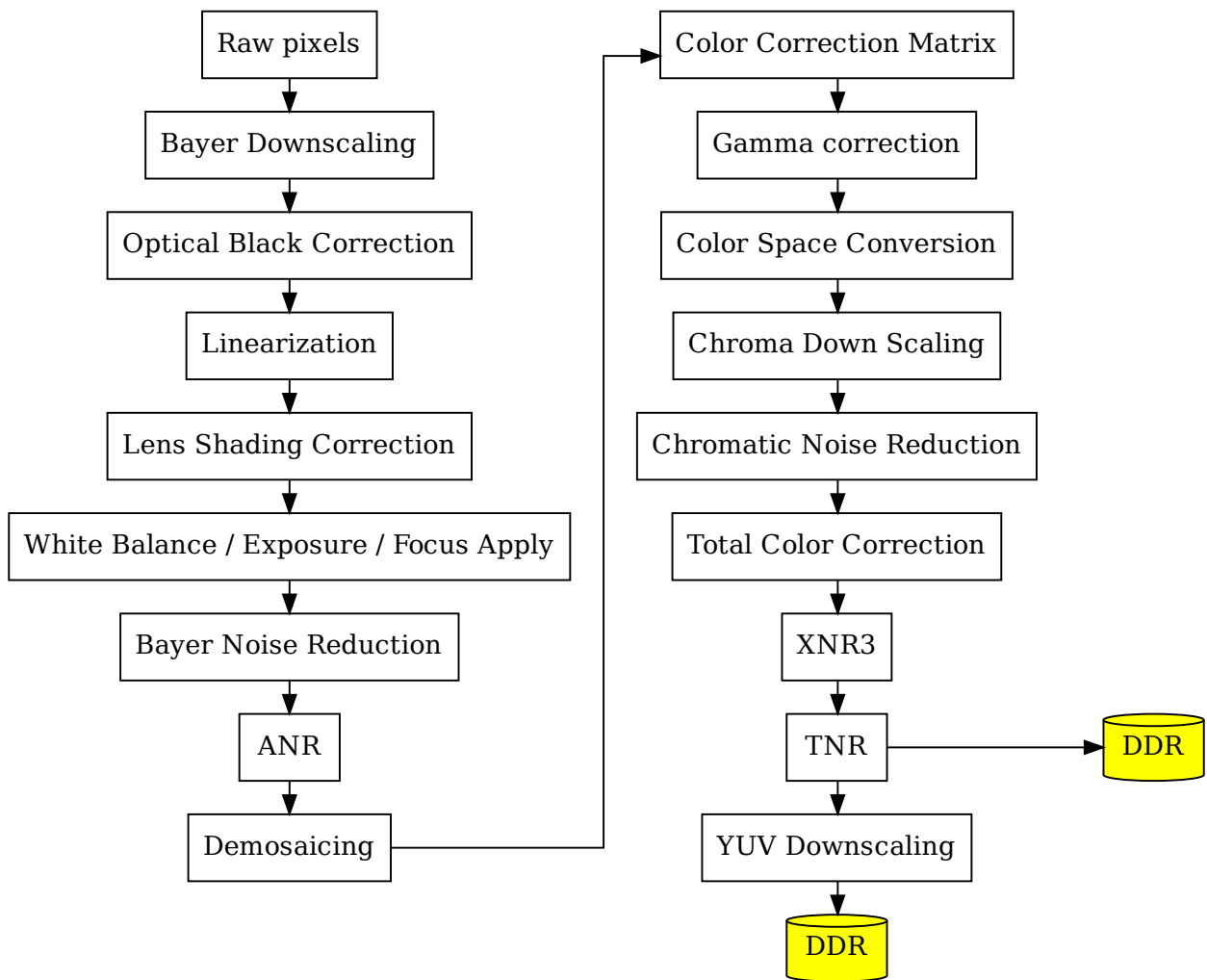


Fig. 4: IPU3 ImgU Pipeline Diagram

The table below presents a description of the above algorithms.

Name	Description
Optical Black Correction	Optical Black Correction block subtracts a pre-defined value from the respective pixel values to obtain better image quality. Defined in <i>struct ipu3_uapi_obgrid_param</i> .
Linearization	This algo block uses linearization parameters to address non-linearity sensor effects. The Lookup table table is defined in <i>struct ipu3_uapi_isp_lin_vmem_params</i> .
SHD	Lens shading correction is used to correct spatial non-uniformity of the pixel response due to optical lens shading. This is done by applying a different gain for each pixel. The gain, black level etc are configured in <i>struct ipu3_uapi_shd_config_static</i> .
BNR	Bayer noise reduction block removes image noise by applying a bilateral filter. See <i>struct ipu3_uapi_bnr_static_config</i> for details.
ANR	Advanced Noise Reduction is a block based algorithm that performs noise reduction in the Bayer domain. The convolution matrix etc can be found in <i>struct ipu3_uapi_anr_config</i> .
DM	Demosaicing converts raw sensor data in Bayer format into RGB (Red, Green, Blue) presentation. Then add outputs of estimation of Y channel for following stream processing by Firmware. The struct is defined as <i>struct ipu3_uapi_dm_config</i> .
Color Correction	Color Correction algo transforms sensor specific color space to the standard "sRGB" color space. This is done by applying 3x3 matrix defined in <i>struct ipu3_uapi_ccm_mat_config</i> .
Gamma correction	Gamma correction <i>struct ipu3_uapi_gamma_config</i> is a basic non-linear tone mapping correction that is applied per pixel for each pixel component.
CSC	Color space conversion transforms each pixel from the RGB primary presentation to YUV (Y: brightness, UV: Luminance) presentation. This is done by applying a 3x3 matrix defined in <i>struct ipu3_uapi_csc_mat_config</i>
CDS	Chroma down sampling After the CSC is performed, the Chroma Down Sampling is applied for a UV plane down sampling by a factor of 2 in each direction for YUV 4:2:0 using a 4x2 configurable filter <i>struct ipu3_uapi_cds_params</i> .
CHNR	Chroma noise reduction This block processes only the chrominance pixels and performs noise reduction by cleaning the high frequency noise. See struct struct ipu3_uapi_yuvp1_chnr_config.
TCC	Total color correction as defined in struct struct ipu3_uapi_yuvp2_tcc_static_config.
XNR3	eXtreme Noise Reduction V3 is the third revision of noise reduction algorithm used to improve image quality. This removes the low frequency noise in the captured image. Two related structs are being defined, <i>struct ipu3_uapi_isp_xnr3_params</i> for ISP data memory and <i>struct ipu3_uapi_isp_xnr3_vmem_params</i> for vector memory.
TNR	Temporal Noise Reduction block compares successive frames in time to remove anomalies / noise in pixel values. <i>struct ipu3_uapi_isp_tnr3_vmem_params</i> and <i>struct ipu3_uapi_isp_tnr3_params</i> are defined for ISP vector and data memory respectively.

Other often encountered acronyms not listed in above table:

- ACC** Accelerator cluster
- AWB_FR** Auto white balance filter response statistics
- BDS** Bayer downscaler parameters
- CCM** Color correction matrix coefficients
- IEFd** Image enhancement filter directed
- Obgrid** Optical black level compensation
- OSYS** Output system configuration
- ROI** Region of interest
- YDS** Y down sampling
- YTM** Y-tone mapping

A few stages of the pipeline will be executed by firmware running on the ISP processor, while many others will use a set of fixed hardware blocks also called accelerator cluster (ACC) to crunch pixel data and produce statistics.

ACC parameters of individual algorithms, as defined by *struct ipu3_uapi_acc_param*, can be chosen to be applied by the user space through struct *ipu3_uapi_flags* embedded in *struct ipu3_uapi_params* structure. For parameters that are configured as not enabled by the user space, the corresponding structs are ignored by the driver, in which case the existing configuration of the algorithm will be preserved.

References

1.1.6.10 The ivtv driver

Author: Hans Verkuil <hverkuil@xs4all.nl>

This is a v4l2 device driver for the Conexant cx23415/6 MPEG encoder/decoder. The cx23415 can do both encoding and decoding, the cx23416 can only do MPEG encoding. Currently the only card featuring full decoding support is the Hauppauge PVR-350.

Note:

- 1) This driver requires the latest encoder firmware (version 2.06.039, size 376836 bytes). Get the firmware from here:

<https://linuxtv.org/downloads/firmware/#conexant>

- 2) 'normal' TV applications do not work with this driver, you need an application that can handle MPEG input such as mplayer, xine, MythTV, etc.

The primary goal of the IVTV project is to provide a "clean room" Linux Open Source driver implementation for video capture cards based on the iCompression iTVC15 or Conexant CX23415/CX23416 MPEG Codec.

Features

- Hardware mpeg2 capture of broadcast video (and sound) via the tuner or S-Video/Composite and audio line-in.
- Hardware mpeg2 capture of FM radio where hardware support exists
- Supports NTSC, PAL, SECAM with stereo sound
- Supports SAP and bilingual transmissions.
- Supports raw VBI (closed captions and teletext).
- Supports sliced VBI (closed captions and teletext) and is able to insert this into the captured MPEG stream.
- Supports raw YUV and PCM input.

Additional features for the PVR-350 (CX23415 based)

- Provides hardware mpeg2 playback
- Provides comprehensive OSD (On Screen Display: ie. graphics overlaying the video signal)
- Provides a framebuffer (allowing X applications to appear on the video device)
- Supports raw YUV output.

IMPORTANT: In case of problems first read this page: https://help.ubuntu.com/community/Install_IVTV_Troubleshooting

See also

<https://linuxtv.org>

IRC

<irc://irc.freenode.net/#v4l>

Devices

A maximum of 12 ivtv boards are allowed at the moment.

Cards that don't have a video output capability (i.e. non PVR350 cards) lack the vbi8, vbi16, video16 and video48 devices. They also do not support the framebuffer device /dev/fbx for OSD.

The radio0 device may or may not be present, depending on whether the card has a radio tuner or not.

Here is a list of the base v4l devices:

```
crw-rw----  1 root    video    81,   0 Jun 19 22:22 /dev/video0
crw-rw----  1 root    video    81,  16 Jun 19 22:22 /dev/video16
crw-rw----  1 root    video    81,  24 Jun 19 22:22 /dev/video24
crw-rw----  1 root    video    81,  32 Jun 19 22:22 /dev/video32
crw-rw----  1 root    video    81,  48 Jun 19 22:22 /dev/video48
crw-rw----  1 root    video    81,  64 Jun 19 22:22 /dev/radio0
crw-rw----  1 root    video    81, 224 Jun 19 22:22 /dev/vbi0
crw-rw----  1 root    video    81, 228 Jun 19 22:22 /dev/vbi8
crw-rw----  1 root    video    81, 232 Jun 19 22:22 /dev/vbi16
```

Base devices

For every extra card you have the numbers increased by one. For example, `/dev/video0` is listed as the 'base' encoding capture device so we have:

- `/dev/video0` is the encoding capture device for the first card (card 0)
- `/dev/video1` is the encoding capture device for the second card (card 1)
- `/dev/video2` is the encoding capture device for the third card (card 2)

Note that if the first card doesn't have a feature (eg no decoder, so no `video16`, the second card will still use `video17`. The simple rule is 'add the card number to the base device number'. If you have other capture cards (e.g. WinTV PCI) that are detected first, then you have to tell the `ivtv` module about it so that it will start counting at 1 (or 2, or whatever). Otherwise the device numbers can get confusing. The `ivtv` `'ivtv_first_minor'` module option can be used for that.

- `/dev/video0`

The encoding capture device(s).

Read-only.

Reading from this device gets you the MPEG1/2 program stream. Example:

```
cat /dev/video0 > my.mpg (you need to hit ctrl-c to exit)
```

- `/dev/video16`

The decoder output device(s)

Write-only. Only present if the MPEG decoder (i.e. CX23415) exists.

An mpeg2 stream sent to this device will appear on the selected video display, audio will appear on the line-out/audio out. It is only available for cards that support video out. Example:

```
cat my.mpg >/dev/video16
```

- `/dev/video24`

The raw audio capture device(s).

Read-only

The raw audio PCM stereo stream from the currently selected tuner or audio line-in. Reading from this device results in a raw (signed 16 bit Little Endian, 48000 Hz, stereo pcm) capture. This device only captures audio. This should be replaced by an ALSA device in the

future. Note that there is no corresponding raw audio output device, this is not supported in the decoder firmware.

- /dev/video32

The raw video capture device(s)

Read-only

The raw YUV video output from the current video input. The YUV format is a 16x16 linear tiled NV12 format (V4L2_PIX_FMT_NV12_16L16)

Note that the YUV and PCM streams are not synchronized, so they are of limited use.

- /dev/video48

The raw video display device(s)

Write-only. Only present if the MPEG decoder (i.e. CX23415) exists.

Writes a YUV stream to the decoder of the card.

- /dev/radio0

The radio tuner device(s)

Cannot be read or written.

Used to enable the radio tuner and tune to a frequency. You cannot read or write audio streams with this device. Once you use this device to tune the radio, use /dev/video24 to read the raw pcm stream or /dev/video0 to get an mpeg2 stream with black video.

- /dev/vbi0

The 'vertical blank interval' (Teletext, CC, WSS etc) capture device(s)

Read-only

Captures the raw (or sliced) video data sent during the Vertical Blank Interval. This data is used to encode teletext, closed captions, VPS, widescreen signalling, electronic program guide information, and other services.

- /dev/vbi8

Processed vbi feedback device(s)

Read-only. Only present if the MPEG decoder (i.e. CX23415) exists.

The sliced VBI data embedded in an MPEG stream is reproduced on this device. So while playing back a recording on /dev/video16, you can read the embedded VBI data from /dev/vbi8.

- /dev/vbi16

The vbi 'display' device(s)

Write-only. Only present if the MPEG decoder (i.e. CX23415) exists.

Can be used to send sliced VBI data to the video-out connector.

1.1.6.11 Vaio Picturebook Motion Eye Camera Driver

Copyright © 2001-2004 Stelian Pop <stelian@popies.net>

Copyright © 2001-2002 Alcôve <www.alcove.com>

Copyright © 2000 Andrew Tridge <tridge@samba.org>

This driver enable the use of video4linux compatible applications with the Motion Eye camera. This driver requires the “Sony Laptop Extras” driver (which can be found in the “Misc devices” section of the kernel configuration utility) to be compiled and installed (using its “camera=1” parameter).

It can do at maximum 30 fps @ 320x240 or 15 fps @ 640x480.

Grabbing is supported in packed YUV colorspace only.

MJPEG hardware grabbing is supported via a private API (see below).

Hardware supported

This driver supports the ‘second’ version of the MotionEye camera :)

The first version was connected directly on the video bus of the Neomagic video card and is unsupported.

The second one, made by Kawasaki Steel is fully supported by this driver (PCI vendor/device is 0x136b/0xff01)

The third one, present in recent (more or less last year) Picturebooks (C1M* models), is not supported. The manufacturer has given the specs to the developers under a NDA (which allows the development of a GPL driver however), but things are not moving very fast (see <http://r-engine.sourceforge.net/>) (PCI vendor/device is 0x10cf/0x2011).

There is a forth model connected on the USB bus in TR1* Vaio laptops. This camera is not supported at all by the current driver, in fact little information if any is available for this camera (USB vendor/device is 0x054c/0x0107).

Driver options

Several options can be passed to the meye driver using the standard module argument syntax (<param>=<value> when passing the option to the module or meye.<param>=<value> on the kernel boot line when meye is statically linked into the kernel). Those options are:

gbuffers:	number of capture buffers, default is 2 (32 max)
gbufsize:	size of each capture buffer, default is 614400
video_nr:	video device to register (0 = /dev/video0, etc)

Module use

In order to automatically load the meye module on use, you can put those lines in your `/etc/modprobe.d/meye.conf` file:

```
alias char-major-81 videodev
alias char-major-81-0 meye
options meye gbuffers=32
```

Usage:

```
xawtv >= 3.49 (<http://bytesex.org/xawtv/>)
    for display and uncompressed video capture:

        xawtv -c /dev/video0 -geometry 640x480
            or
        xawtv -c /dev/video0 -geometry 320x240

motioneye (<http://popies.net/meye/>)
    for getting ppm or jpg snapshots, mjpeg video
```

Bugs / Todo

- ‘motioneye’ still uses the meye private v4l1 API extensions.

1.1.6.12 OMAP 3 Image Signal Processor (ISP) driver

Copyright © 2010 Nokia Corporation

Copyright © 2009 Texas Instruments, Inc.

Contacts: Laurent Pinchart <laurent.pinchart@ideasonboard.com>, Sakari Ailus <sakari.ailus@iki.fi>, David Cohen <dacohen@gmail.com>

Introduction

This file documents the Texas Instruments OMAP 3 Image Signal Processor (ISP) driver located under `drivers/media/platform/omap3isp`. The original driver was written by Texas Instruments but since that it has been rewritten (twice) at Nokia.

The driver has been successfully used on the following versions of OMAP 3:

- 3430
- 3530
- 3630

The driver implements V4L2, Media controller and v4l2_subdev interfaces. Sensor, lens and flash drivers using the v4l2_subdev interface in the kernel are supported.

Split to subdevs

The OMAP 3 ISP is split into V4L2 subdevs, each of the blocks inside the ISP having one subdev to represent it. Each of the subdevs provide a V4L2 subdev interface to userspace.

- OMAP3 ISP CCP2
- OMAP3 ISP CSI2a
- OMAP3 ISP CCDC
- OMAP3 ISP preview
- OMAP3 ISP resizer
- OMAP3 ISP AEWB
- OMAP3 ISP AF
- OMAP3 ISP histogram

Each possible link in the ISP is modelled by a link in the Media controller interface. For an example program see¹.

Controlling the OMAP 3 ISP

In general, the settings given to the OMAP 3 ISP take effect at the beginning of the following frame. This is done when the module becomes idle during the vertical blanking period on the sensor. In memory-to-memory operation the pipe is run one frame at a time. Applying the settings is done between the frames.

All the blocks in the ISP, excluding the CSI-2 and possibly the CCP2 receiver, insist on receiving complete frames. Sensors must thus never send the ISP partial frames.

Autoidle does have issues with some ISP blocks on the 3430, at least. Autoidle is only enabled on 3630 when the omap3isp module parameter autoidle is non-zero.

Technical reference manuals (TRMs) and other documentation

OMAP 3430 TRM: <URL:http://focus.ti.com/pdfs/wtbu/OMAP34xx_ES3.1.x_PUBLIC_TRM_vZM.zip>
Referenced 2011-03-05.

OMAP 35xx TRM: <URL:<http://www.ti.com/litv/pdf/spruf98o>> Referenced 2011-03-05.

OMAP 3630 TRM: <URL:http://focus.ti.com/pdfs/wtbu/OMAP36xx_ES1.x_PUBLIC_TRM_vQ.zip>
Referenced 2011-03-05.

DM 3730 TRM: <URL:<http://www.ti.com/litv/pdf/sprugn4h>> Referenced 2011-03-06.

¹ <http://git.ideasonboard.org/?p=media-ctl.git;a=summary>

References

1.1.6.13 OMAP4 ISS Driver

Author: Sergio Aguirre <sergio.a.aguirre@gmail.com>

Copyright (C) 2012, Texas Instruments

Introduction

The OMAP44XX family of chips contains the Imaging SubSystem (a.k.a. ISS), Which contains several components that can be categorized in 3 big groups:

- Interfaces (2 Interfaces: CSI2-A & CSI2-B/CCP2)
- ISP (Image Signal Processor)
- SIMCOP (Still Image Coprocessor)

For more information, please look in¹ for latest version of: “OMAP4430 Multimedia Device Silicon Revision 2.x”

As of Revision AB, the ISS is described in detail in section 8.

This driver is supporting **only** the CSI2-A/B interfaces for now.

It makes use of the Media Controller framework², and inherited most of the code from OMAP3 ISP driver (found under `drivers/media/platform/omap3isp/*`), except that it doesn't need an IOMMU now for ISS buffers memory mapping.

Supports usage of MMAP buffers only (for now).

Tested platforms

- OMAP4430SDP, w/ ES2.1 GP & SEVM4430-CAM-V1-0 (Contains IMX060 & OV5640, in which only the last one is supported, outputting YUV422 frames).
- TI Blaze MDP, w/ OMAP4430 ES2.2 EMU (Contains 1 IMX060 & 2 OV5650 sensors, in which only the OV5650 are supported, outputting RAW10 frames).
- PandaBoard, Rev. A2, w/ OMAP4430 ES2.1 GP & OV adapter board, tested with following sensors: * OV5640 * OV5650
- Tested on mainline kernel:

<http://git.kernel.org/?p=linux/kernel/git/torvalds/linux.git;a=summary>

Tag: v3.3 (commit c16fa4f2ad19908a47c63d8fa436a1178438c7e7)

¹ <http://focus.ti.com/general/docs/wtbu/wtbudocumentcenter.tsp?navigationId=12037&templateId=6123#62>

² <http://lwn.net/Articles/420485/>

File list

drivers/staging/media/omap4iss/ include/linux/platform_data/media/omap4iss.h

References

1.1.6.14 Philips webcams (pwc driver)

This file contains some additional information for the Philips and OEM webcams. E-mail: webcam@smcc.demon.nl Last updated: 2004-01-19 Site: <http://www.smcc.demon.nl/webcam/>

As of this moment, the following cameras are supported:

- Philips PCA645
- Philips PCA646
- Philips PCVC675
- Philips PCVC680
- Philips PCVC690
- Philips PCVC720/40
- Philips PCVC730
- Philips PCVC740
- Philips PCVC750
- Askey VC010
- Creative Labs Webcam 5
- Creative Labs Webcam Pro Ex
- Logitech QuickCam 3000 Pro
- Logitech QuickCam 4000 Pro
- Logitech QuickCam Notebook Pro
- Logitech QuickCam Zoom
- Logitech QuickCam Orbit
- Logitech QuickCam Sphere
- Samsung MPC-C10
- Samsung MPC-C30
- Sotec Afina Eye
- AME CU-001
- Visionite VCS-UM100
- Visionite VCS-UC300

The main webpage for the Philips driver is at the address above. It contains a lot of extra information, a FAQ, and the binary plugin 'PWCX'. This plugin contains decompression routines that allow you to use higher image sizes and framerates; in addition the webcam uses less bandwidth on the USB bus (handy if you want to run more than 1 camera simultaneously). These routines fall under a NDA, and may therefore not be distributed as source; however, its use is completely optional.

You can build this code either into your kernel, or as a module. I recommend the latter, since it makes troubleshooting a lot easier. The built-in microphone is supported through the USB Audio class.

When you load the module you can set some default settings for the camera; some programs depend on a particular image-size or -format and don't know how to set it properly in the driver. The options are:

size Can be one of 'sqcif', 'qsif', 'qcif', 'sif', 'cif' or 'vga', for an image size of resp. 128x96, 160x120, 176x144, 320x240, 352x288 and 640x480 (of course, only for those cameras that support these resolutions).

fps Specifies the desired framerate. Is an integer in the range of 4-30.

fbufs This parameter specifies the number of internal buffers to use for storing frames from the cam. This will help if the process that reads images from the cam is a bit slow or momentarily busy. However, on slow machines it only introduces lag, so choose carefully. The default is 3, which is reasonable. You can set it between 2 and 5.

mbufs This is an integer between 1 and 10. It will tell the module the number of buffers to reserve for mmap(), VIDIOCCGMBUF, VIDIOCMCAPTURE and friends. The default is 2, which is adequate for most applications (double buffering).

Should you experience a lot of 'Dumping frame...' messages during grabbing with a tool that uses mmap(), you might want to increase it. However, it doesn't really buffer images, it just gives you a bit more slack when your program is behind. But you need a multi-threaded or forked program to really take advantage of these buffers.

The absolute maximum is 10, but don't set it too high! Every buffer takes up 460 KB of RAM, so unless you have a lot of memory setting this to something more than 4 is an absolute waste. This memory is only allocated during open(), so nothing is wasted when the camera is not in use.

power_save When power_save is enabled (set to 1), the module will try to shut down the cam on close() and re-activate on open(). This will save power and turn off the LED. Not all cameras support this though (the 645 and 646 don't have power saving at all), and some models don't work either (they will shut down, but never wake up). Consider this experimental. By default this option is disabled.

compression (only useful with the plugin) With this option you can control the compression factor that the camera uses to squeeze the image through the USB bus. You can set the parameter between 0 and 3:

```
0 = prefer uncompressed images; if the requested mode is not available
  in an uncompressed format, the driver will silently switch to low
  compression.
1 = low compression.
2 = medium compression.
3 = high compression.
```

High compression takes less bandwidth of course, but it could also introduce some unwanted artefacts. The default is 2, medium compression. See the FAQ on the website for an overview of which modes require compression.

The compression parameter does not apply to the 645 and 646 cameras and OEM models derived from those (only a few). Most cams honour this parameter.

leds This settings takes 2 integers, that define the on/off time for the LED (in milliseconds). One of the interesting things that you can do with this is let the LED blink while the camera is in use. This:

```
leds=500,500
```

will blink the LED once every second. But with:

```
leds=0,0
```

the LED never goes on, making it suitable for silent surveillance.

By default the camera's LED is on solid while in use, and turned off when the camera is not used anymore.

This parameter works only with the ToUCam range of cameras (720, 730, 740, 750) and OEMs. For other cameras this command is silently ignored, and the LED cannot be controlled.

Finally: this parameters does not take effect UNTIL the first time you open the camera device. Until then, the LED remains on.

dev_hint A long standing problem with USB devices is their dynamic nature: you never know what device a camera gets assigned; it depends on module load order, the hub configuration, the order in which devices are plugged in, and the phase of the moon (i.e. it can be random). With this option you can give the driver a hint as to what video device node (/dev/videoX) it should use with a specific camera. This is also handy if you have two cameras of the same model.

A camera is specified by its type (the number from the camera model, like PCA645, PCVC750VC, etc) and optionally the serial number (visible in /sys/kernel/debug/usb/devices). A hint consists of a string with the following format:

```
[type[.serialnumber]:]node
```

The square brackets mean that both the type and the serialnumber are optional, but a serialnumber cannot be specified without a type (which would be rather pointless). The serialnumber is separated from the type by a '.'; the node number by a ':'.

This somewhat cryptic syntax is best explained by a few examples:

dev_hint=3,5	The first detected cam gets assigned /dev/video3, the second /dev/video5. Any other cameras will get the first free available slot (see below).
dev_hint=645:1,680:2	The PCA645 camera will get /dev/video1, and a PCVC680 /dev/video2.
dev_hint=645.0123:3,645.4567:0	The PCA645 camera with serialnumber

	0123 goes to /dev/video3, the same camera model with the 4567 serial gets /dev/video0.
dev_hint=750:1,4,5,6	The PCVC750 camera will get /dev/video1, the next 3 Philips cams will use /dev/video4 through /dev/video6.

Some points worth knowing:

- Serialnumbers are case sensitive and must be written full, including leading zeroes (it's treated as a string).
- If a device node is already occupied, registration will fail and the webcam is not available.
- You can have up to 64 video devices; be sure to make enough device nodes in /dev if you want to spread the numbers. After /dev/video9 comes /dev/video10 (not /dev/videoA).
- If a camera does not match any dev_hint, it will simply get assigned the first available device node, just as it used to be.

trace In order to better detect problems, it is now possible to turn on a 'trace' of some of the calls the module makes; it logs all items in your kernel log at debug level.

The trace variable is a bitmask; each bit represents a certain feature. If you want to trace something, look up the bit value(s) in the table below, add the values together and supply that to the trace variable.

Value (dec)	Value (hex)	Description	Default
1	0x1	Module initialization; this will log messages while loading and unloading the module	On
2	0x2	probe() and disconnect() traces	On
4	0x4	Trace open() and close() calls	Off
8	0x8	read(), mmap() and associated ioctl() calls	Off
16	0x10	Memory allocation of buffers, etc.	Off
32	0x20	Showing underflow, overflow and Dumping frame messages	On
64	0x40	Show viewport and image sizes	Off
128	0x80	PWCX debugging	Off

For example, to trace the open() & read() functions, sum $8 + 4 = 12$, so you would supply trace=12 during insmod or modprobe. If you want to turn the initialization and probing tracing off, set trace=0. The default value for trace is 35 (0x23).

Example:

```
# modprobe pwc size=cif fps=15 power_save=1
```

The fbufs, mbufs and trace parameters are global and apply to all connected cameras. Each camera has its own set of buffers.

size and fps only specify defaults when you open() the device; this is to accommodate some tools that don't set the size. You can change these settings after open() with the Video4Linux ioctl() calls. The default of defaults is QCIF size at 10 fps.

The compression parameter is semiglobal; it sets the initial compression preference for all camera's, but this parameter can be set per camera with the VIDIOCPWCSCQUAL ioctl() call. All parameters are optional.

1.1.6.15 Qualcomm Camera Subsystem driver

Introduction

This file documents the Qualcomm Camera Subsystem driver located under `drivers/media/platform/qcom/camss`.

The current version of the driver supports the Camera Subsystem found on Qualcomm MSM8916/APQ8016 and MSM8996/APQ8096 processors.

The driver implements V4L2, Media controller and V4L2 subdev interfaces. Camera sensor using V4L2 subdev interface in the kernel is supported.

The driver is implemented using as a reference the Qualcomm Camera Subsystem driver for Android as found in Code Aurora^{1,2}.

Qualcomm Camera Subsystem hardware

The Camera Subsystem hardware found on 8x16 / 8x96 processors and supported by the driver consists of:

- 2 / 3 CSIPHY modules. They handle the Physical layer of the CSI2 receivers. A separate camera sensor can be connected to each of the CSIPHY module;
- 2 / 4 CSID (CSI Decoder) modules. They handle the Protocol and Application layer of the CSI2 receivers. A CSID can decode data stream from any of the CSIPHY. Each CSID also contains a TG (Test Generator) block which can generate artificial input data for test purposes;
- ISPIF (ISP Interface) module. Handles the routing of the data streams from the CSIDs to the inputs of the VFE;
- 1 / 2 VFE (Video Front End) module(s). Contain a pipeline of image processing hardware blocks. The VFE has different input interfaces. The PIX (Pixel) input interface feeds the input data to the image processing pipeline. The image processing pipeline contains also a scale and crop module at the end. Three RDI (Raw Dump Interface) input interfaces bypass the image processing pipeline. The VFE also contains the AXI bus interface which writes the output data to memory.

¹ <https://source.codeaurora.org/quic/la/kernel/msm-3.10/>

² <https://source.codeaurora.org/quic/la/kernel/msm-3.18/>

Supported functionality

The current version of the driver supports:

- Input from camera sensor via CSIPHY;
- Generation of test input data by the TG in CSID;
- RDI interface of VFE
 - Raw dump of the input data to memory.

Supported formats:

- * YUYV/UYVY/YVYU/VYUY (packed YUV 4:2:2 - V4L2_PIX_FMT_YUYV / V4L2_PIX_FMT_UYVY / V4L2_PIX_FMT_YVYU / V4L2_PIX_FMT_VYUY);
- * MIPI RAW8 (8bit Bayer RAW - V4L2_PIX_FMT_SRGB8 / V4L2_PIX_FMT_SGRBG8 / V4L2_PIX_FMT_SGBRG8 / V4L2_PIX_FMT_SBGGR8);
- * MIPI RAW10 (10bit packed Bayer RAW - V4L2_PIX_FMT_SBGGR10P / V4L2_PIX_FMT_SGBRG10P / V4L2_PIX_FMT_SGRBG10P / V4L2_PIX_FMT_SRGB10P / V4L2_PIX_FMT_Y10P);
- * MIPI RAW12 (12bit packed Bayer RAW - V4L2_PIX_FMT_SRGB12P / V4L2_PIX_FMT_SGBRG12P / V4L2_PIX_FMT_SGRBG12P / V4L2_PIX_FMT_SRGB12P);
- * (8x96 only) MIPI RAW14 (14bit packed Bayer RAW - V4L2_PIX_FMT_SRGB14P / V4L2_PIX_FMT_SGBRG14P / V4L2_PIX_FMT_SGRBG14P / V4L2_PIX_FMT_SRGB14P).

- (8x96 only) Format conversion of the input data.

Supported input formats:

- * MIPI RAW10 (10bit packed Bayer RAW - V4L2_PIX_FMT_SBGGR10P / V4L2_PIX_FMT_Y10P).

Supported output formats:

- * Plain16 RAW10 (10bit unpacked Bayer RAW - V4L2_PIX_FMT_SBGGR10 / V4L2_PIX_FMT_Y10).

- PIX interface of VFE

- Format conversion of the input data.

Supported input formats:

- * YUYV/UYVY/YVYU/VYUY (packed YUV 4:2:2 - V4L2_PIX_FMT_YUYV / V4L2_PIX_FMT_UYVY / V4L2_PIX_FMT_YVYU / V4L2_PIX_FMT_VYUY).

Supported output formats:

- * NV12/NV21 (two plane YUV 4:2:0 - V4L2_PIX_FMT_NV12 / V4L2_PIX_FMT_NV21);
- * NV16/NV61 (two plane YUV 4:2:2 - V4L2_PIX_FMT_NV16 / V4L2_PIX_FMT_NV61).
- * (8x96 only) YUYV/UYVY/YVYU/VYUY (packed YUV 4:2:2 - V4L2_PIX_FMT_YUYV / V4L2_PIX_FMT_UYVY / V4L2_PIX_FMT_YVYU / V4L2_PIX_FMT_VYUY).

- Scaling support. Configuration of the VFE Encoder Scale module for downscaling with ratio up to 16x.
- Cropping support. Configuration of the VFE Encoder Crop module.
- Concurrent and independent usage of two (8x96: three) data inputs - could be camera sensors and/or TG.

Driver Architecture and Design

The driver implements the V4L2 subdev interface. With the goal to model the hardware links between the modules and to expose a clean, logical and usable interface, the driver is split into V4L2 sub-devices as follows (8x16 / 8x96):

- 2 / 3 CSIPHY sub-devices - each CSIPHY is represented by a single sub-device;
- 2 / 4 CSID sub-devices - each CSID is represented by a single sub-device;
- 2 / 4 ISPIF sub-devices - ISPIF is represented by a number of sub-devices equal to the number of CSID sub-devices;
- 4 / 8 VFE sub-devices - VFE is represented by a number of sub-devices equal to the number of the input interfaces (3 RDI and 1 PIX for each VFE).

The considerations to split the driver in this particular way are as follows:

- representing CSIPHY and CSID modules by a separate sub-device for each module allows to model the hardware links between these modules;
- representing VFE by a separate sub-devices for each input interface allows to use the input interfaces concurrently and independently as this is supported by the hardware;
- representing ISPIF by a number of sub-devices equal to the number of CSID sub-devices allows to create linear media controller pipelines when using two cameras simultaneously. This avoids branches in the pipelines which otherwise will require a) userspace and b) media framework (e.g. power on/off operations) to make assumptions about the data flow from a sink pad to a source pad on a single media entity.

Each VFE sub-device is linked to a separate video device node.

The media controller pipeline graph is as follows (with connected two / three OV5645 camera sensors):

Implementation

Runtime configuration of the hardware (updating settings while streaming) is not required to implement the currently supported functionality. The complete configuration on each hardware module is applied on STREAMON ioctl based on the current active media links, formats and controls set.

The output size of the scaler module in the VFE is configured with the actual compose selection rectangle on the sink pad of the 'msm_vfe0_pix' entity.

The crop output area of the crop module in the VFE is configured with the actual crop selection rectangle on the source pad of the 'msm_vfe0_pix' entity.

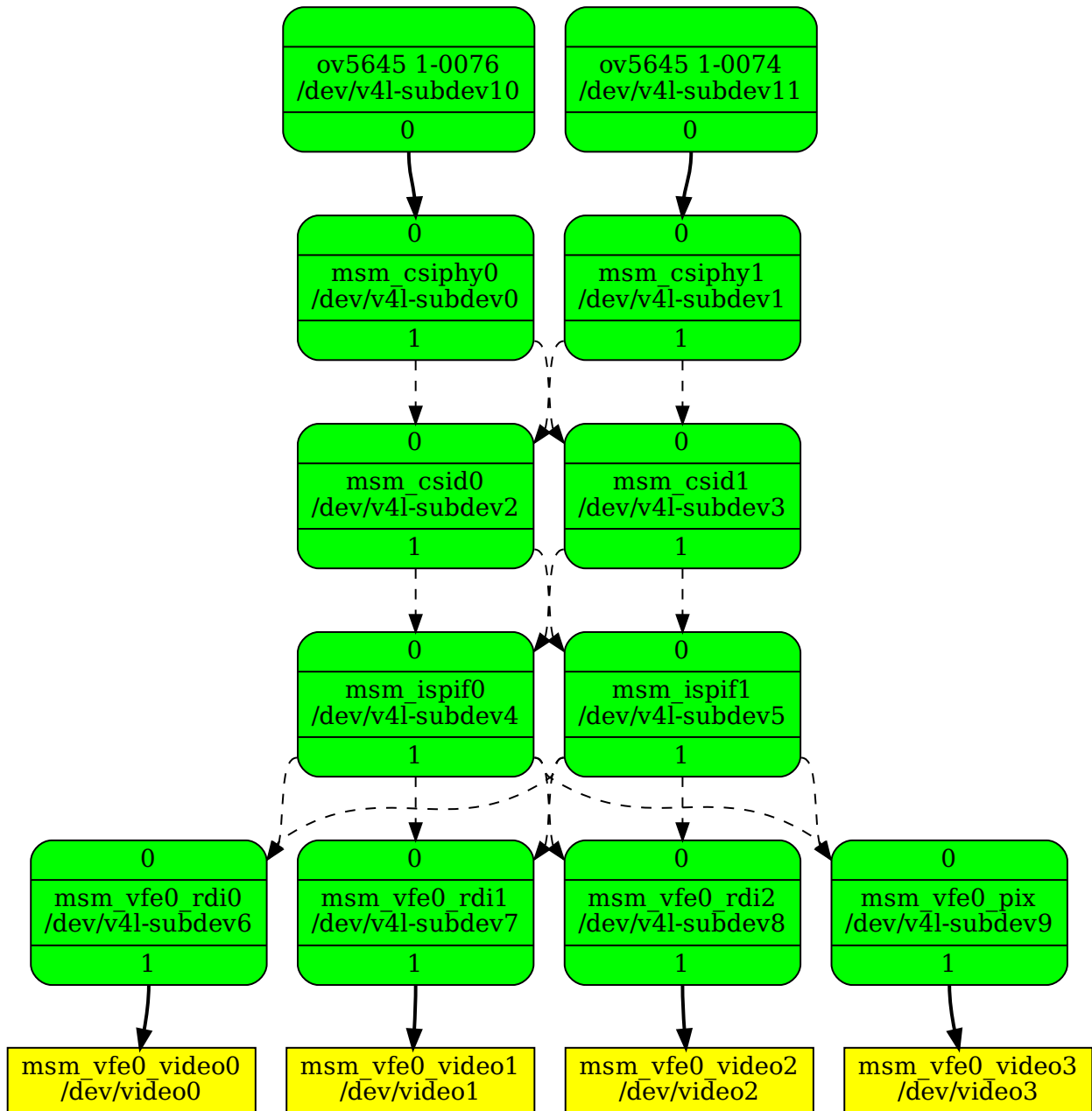


Fig. 5: Media pipeline graph 8x16

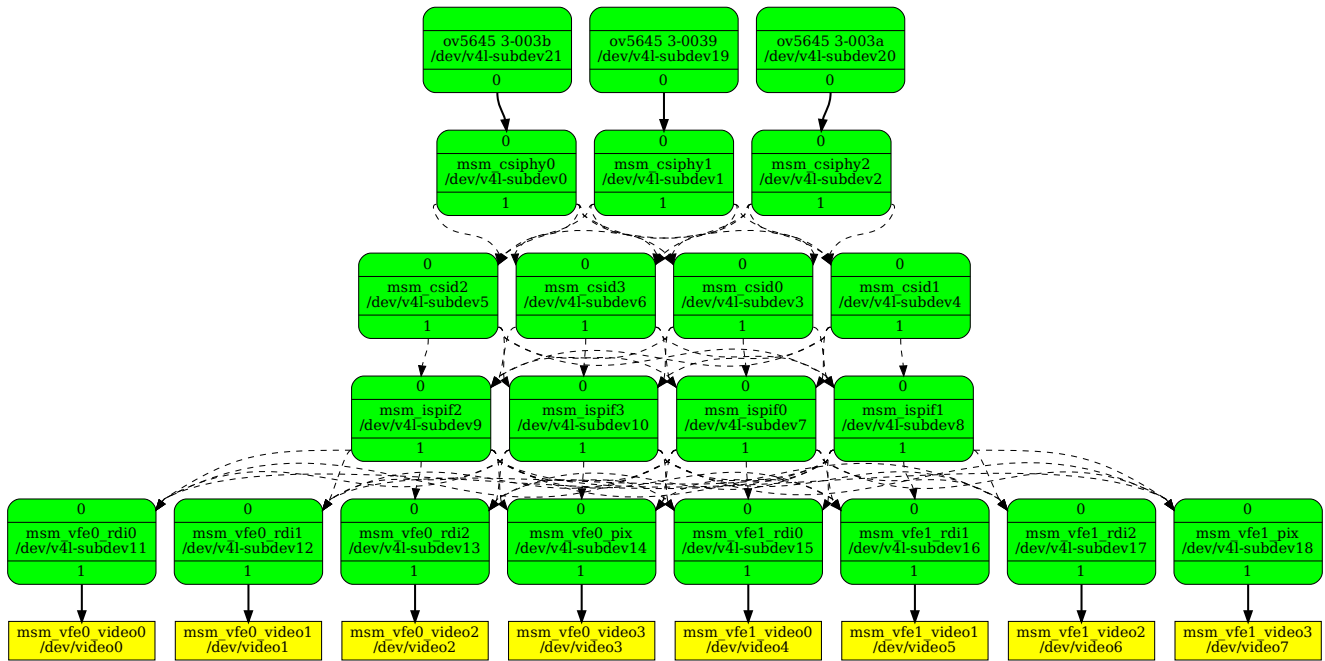


Fig. 6: Media pipeline graph 8x96

Documentation

APQ8016 Specification: <https://developer.qualcomm.com/download/sd410/snapdragon-410-processor-device-specification.pdf> Referenced 2016-11-24.

APQ8096 Specification: <https://developer.qualcomm.com/download/sd820e/qualcomm-snapdragon-820e-processor-apq8096sge-device-specification.pdf> Referenced 2018-06-22.

References

1.1.6.16 Renesas R-Car Fine Display Processor (FDP1) Driver

The R-Car FDP1 driver implements driver-specific controls as follows.

V4L2_CID_DEINTERLACING_MODE (menu) The video deinterlacing mode (such as Bob, Weave, ...). The R-Car FDP1 driver implements the following modes.

"Progressive" (0)	The input image video stream is progressive (not interlaced). No deinterlacing is performed. Apart from (optional) format and encoding conversion output frames are identical to the input frames.
"Adaptive 2D/3D" (1)	Motion adaptive version of 2D and 3D deinterlacing. Use 3D deinterlacing in the presence of fast motion and 2D deinterlacing with diagonal interpolation otherwise.
"Fixed 2D" (2)	The current field is scaled vertically by averaging adjacent lines to recover missing lines. This method is also known as blending or Line Averaging (LAV).
"Fixed 3D" (3)	The previous and next fields are averaged to recover lines missing from the current field. This method is also known as Field Averaging (FAV).
"Previous field" (4)	The current field is weaved with the previous field, i.e. the previous field is used to fill missing lines from the current field. This method is also known as weave deinterlacing.
"Next field" (5)	The current field is weaved with the next field, i.e. the next field is used to fill missing lines from the current field. This method is also known as weave deinterlacing.

1.1.6.17 Rockchip Image Signal Processor (rkisp1)

Introduction

This file documents the driver for the Rockchip ISP1 that is part of RK3288 and RK3399 SoCs. The driver is located under `drivers/staging/media/rkisp1` and uses the Media-Controller API.

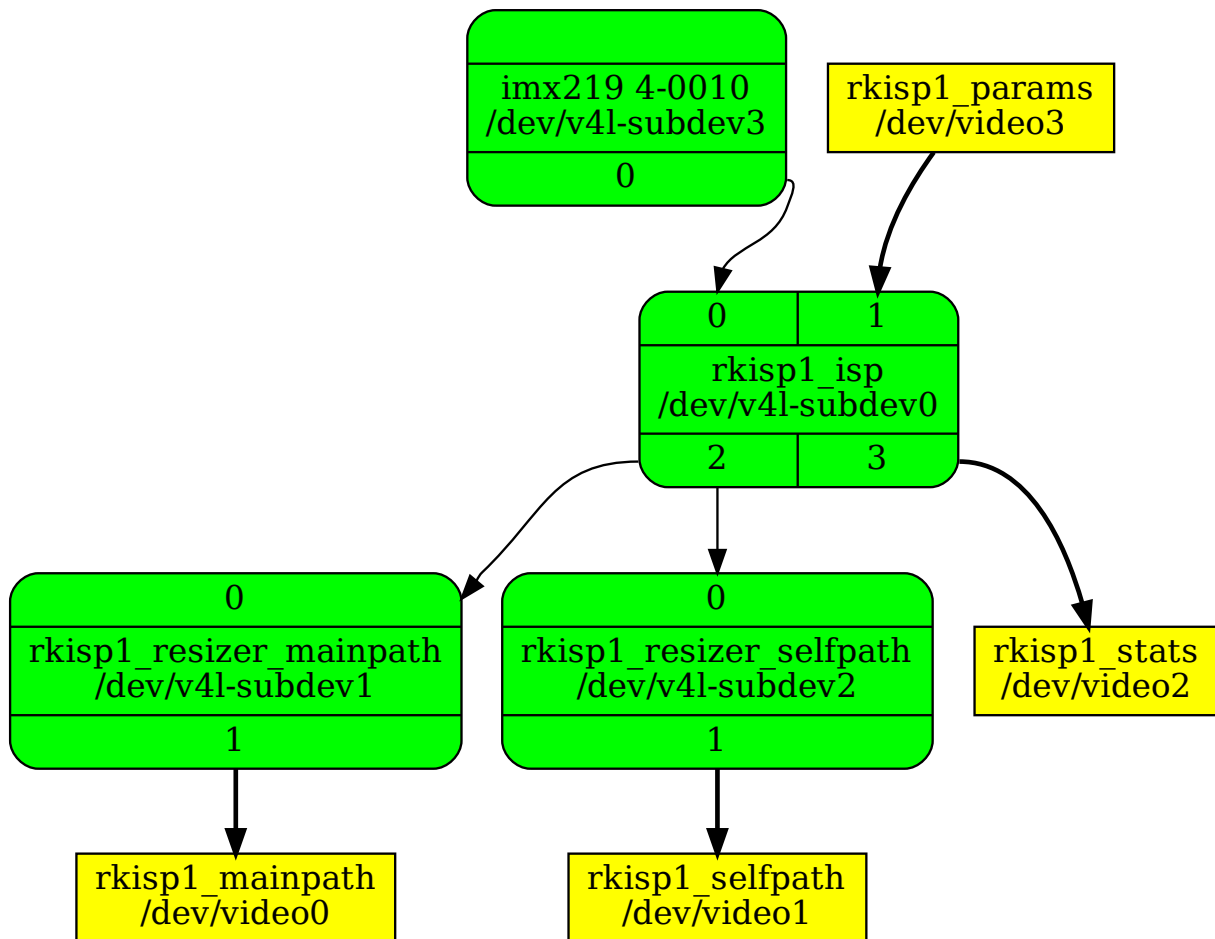
Revisions

There exist multiple smaller revisions to this ISP that got introduced in later SoCs. Revisions can be found in the enum `rkisp1_cif_isp_version` in the UAPI and the revision of the ISP inside the running SoC can be read in the field `hw_revision` of struct `media_device_info` as returned by `ioctl MEDIA_IOC_DEVICE_INFO`.

Versions in use are:

- RKISP1_V10: used at least in rk3288 and rk3399
- RKISP1_V11: declared in the original vendor code, but not used
- RKISP1_V12: used at least in rk3326 and px30
- RKISP1_V13: used at least in rk1808

Topology



The driver has 4 video devices:

- rkisp1_mainpath: capture device for retrieving images, usually in higher resolution.
- rkisp1_selfpath: capture device for retrieving images.
- rkisp1_stats: a metadata capture device that sends statistics.
- rkisp1_params: a metadata output device that receives parameters configurations from userspace.

The driver has 3 subdevices:

- rkisp1_resizer_mainpath: used to resize and downsample frames for the mainpath capture device.
- rkisp1_resizer_selfpath: used to resize and downsample frames for the selfpath capture device.
- rkisp1_isp: is connected to the sensor and is responsible for all the isp operations.

rkisp1_mainpath, rkisp1_selfpath - Frames Capture Video Nodes

Those are the *mainpath* and *selfpath* capture devices to capture frames. Those entities are the DMA engines that write the frames to memory. The selfpath video device can capture YUV/RGB formats. Its input is YUV encoded stream and it is able to convert it to RGB. The selfpath is not able to capture bayer formats. The mainpath can capture both bayer and YUV formats but it is not able to capture RGB formats. Both capture videos support the `V4L2_CAP_IO_MC` *capability*.

rkisp1_resizer_mainpath, rkisp1_resizer_selfpath - Resizers Subdevices Nodes

Those are resizer entities for the mainpath and the selfpath. Those entities can scale the frames up and down and also change the YUV sampling (for example YUV4:2:2 -> YUV4:2:0). They also have cropping capability on the sink pad. The resizers entities can only operate on YUV:4:2:2 format (`MEDIA_BUS_FMT_YUYV8_2X8`). The mainpath capture device supports capturing video in bayer formats. In that case the resizer of the mainpath is set to 'bypass' mode - it just forward the frame without operating on it.

rkisp1_isp - Image Signal Processing Subdevice Node

This is the isp entity. It is connected to the sensor on sink pad 0 and receives the frames using the CSI-2 protocol. It is responsible of configuring the CSI-2 protocol. It has a cropping capability on sink pad 0 that is connected to the sensor and on source pad 2 connected to the resizer entities. Cropping on sink pad 0 defines the image region from the sensor. Cropping on source pad 2 defines the region for the Image Stabilizer (IS).

rkisp1_stats - Statistics Video Node

The statistics video node outputs the 3A (auto focus, auto exposure and auto white balance) statistics, and also histogram statistics for the frames that are being processed by the rkisp1 to userspace applications. Using these data, applications can implement algorithms and re-parameterize the driver through the rkisp_params node to improve image quality during a video stream. The buffer format is defined by struct `rkisp1_stat_buffer`, and userspace should set `V4L2_META_FMT_RK_ISP1_STAT_3A` as the dataformat.

rkisp1_params - Parameters Video Node

The rkisp1_params video node receives a set of parameters from userspace to be applied to the hardware during a video stream, allowing userspace to dynamically modify values such as black level, cross talk corrections and others.

The buffer format is defined by struct `rkisp1_params_cfg`, and userspace should set `V4L2_META_FMT_RK_ISP1_PARAMS` as the dataformat.

Capturing Video Frames Example

In the following example, the sensor connected to pad 0 of 'rkisp1_isp' is imx219.

The following commands can be used to capture video from the selfpath video node with dimension 900x800 planar format YUV 4:2:2. It uses all cropping capabilities possible, (see explanation right below)

```
# set the links
"media-ctl" "-d" "platform:rkisp1" "-r"
"media-ctl" "-d" "platform:rkisp1" "-l" "'imx219 4-0010':0 -> 'rkisp1_isp':0 [1]"
"media-ctl" "-d" "platform:rkisp1" "-l" "'rkisp1_isp':2 -> 'rkisp1_resizer_selfpath':0
↳ [1]"
"media-ctl" "-d" "platform:rkisp1" "-l" "'rkisp1_isp':2 -> 'rkisp1_resizer_mainpath':0
↳ [0]"

# set format for imx219 4-0010:0
"media-ctl" "-d" "platform:rkisp1" "--set-v4l2" "'imx219 4-0010':0 [fmt:SRGB10_1X10/
↳ 1640x1232]"

# set format for rkisp1_isp pads:
"media-ctl" "-d" "platform:rkisp1" "--set-v4l2" "'rkisp1_isp':0 [fmt:SRGB10_1X10/
↳ 1640x1232 crop: (0,0)/1600x1200]"
"media-ctl" "-d" "platform:rkisp1" "--set-v4l2" "'rkisp1_isp':2 [fmt:YUYV8_2X8/
↳ 1600x1200 crop: (0,0)/1500x1100]"

# set format for rkisp1_resizer_selfpath pads:
"media-ctl" "-d" "platform:rkisp1" "--set-v4l2" "'rkisp1_resizer_selfpath':0
↳ [fmt:YUYV8_2X8/1500x1100 crop: (300,400)/1400x1000]"
"media-ctl" "-d" "platform:rkisp1" "--set-v4l2" "'rkisp1_resizer_selfpath':1
↳ [fmt:YUYV8_2X8/900x800]"

# set format for rkisp1_selfpath:
"v4l2-ctl" "-z" "platform:rkisp1" "-d" "rkisp1_selfpath" "-v" "width=900,height=800,"
"v4l2-ctl" "-z" "platform:rkisp1" "-d" "rkisp1_selfpath" "-v" "pixelformat=422P"

# start streaming:
v4l2-ctl "-z" "platform:rkisp1" "-d" "rkisp1_selfpath" "--stream-mmap" "--stream-count
↳ " "10"
```

In the above example the sensor is configured to bayer format: *SRGB10_1X10/1640x1232*. The *rkisp1_isp:0* pad should be configured to the same mbus format and dimensions as the sensor, otherwise streaming will fail with 'EPIPE' error. So it is also configured to *SRGB10_1X10/1640x1232*. In addition, the *rkisp1_isp:0* pad is configured to cropping *(0,0)/1600x1200*.

The cropping dimensions are automatically propagated to be the format of the isp source pad *rkisp1_isp:2*. Another cropping operation is configured on the isp source pad: *(0,0)/1500x1100*.

The resizer's sink pad *rkisp1_resizer_selfpath* should be configured to format *YUYV8_2X8/1500x1100* in order to match the format on the other side of the link. In addition a cropping *(300,400)/1400x1000* is configured on it.

The source pad of the resizer, *rkisp1_resizer_selfpath:1* is configured to format *YUYV8_2X8/900x800*. That means that the resizer first crop a window of *(300,400)/1400x1000* from the received frame and then scales this window to dimension *900x800*.

Note that the above example does not use the stats-params control loop. Therefore the capture frames will not go through the 3A algorithms and probably won't have a good quality, and can even look dark and greenish.

Configuring Quantization

The driver supports limited and full range quantization on YUV formats, where limited is the default. To switch between one or the other, userspace should use the Colorspace Conversion API (CSC) for subdevices on source pad 2 of the isp (*rkisp1_isp:2*). The quantization configured on this pad is the quantization of the captured video frames on the main-path and selfpath video nodes. Note that the resizer and capture entities will always report `V4L2_QUANTIZATION_DEFAULT` even if the quantization is configured to full range on *rkisp1_isp:2*. So in order to get the configured quantization, application should get it from pad *rkisp1_isp:2*.

1.1.6.18 The saa7134 driver

Author Gerd Hoffmann

This is a v4l2/oss device driver for saa7130/33/34/35 based capture / TV boards.

Status

Almost everything is working. video, sound, tuner, radio, mpeg ts, ...

As with bttv, card-specific tweaks are needed. Check `CARDLIST` for a list of known TV cards and `saa7134-cards.c` for the drivers card configuration info.

Build

Once you pick up a Kernel source, you should configure, build, install and boot the new kernel. You'll need at least these config options:

```
./scripts/config -e PCI
./scripts/config -e INPUT
./scripts/config -m I2C
./scripts/config -m MEDIA_SUPPORT
./scripts/config -e MEDIA_PCI_SUPPORT
./scripts/config -e MEDIA_ANALOG_TV_SUPPORT
./scripts/config -e MEDIA_DIGITAL_TV_SUPPORT
./scripts/config -e MEDIA_RADIO_SUPPORT
./scripts/config -e RC_CORE
./scripts/config -e MEDIA_SUBDRV_AUTOSELECT
./scripts/config -m VIDEO_SAA7134
./scripts/config -e SAA7134_ALSA
./scripts/config -e VIDEO_SAA7134_RC
./scripts/config -e VIDEO_SAA7134_DVB
./scripts/config -e VIDEO_SAA7134_G07007
```

To build and install, you should run:

```
make && make modules_install && make install
```

Once the new Kernel is booted, saa7134 driver should be loaded automatically.

Depending on the card you might have to pass `card=<nr>` as `insmod` option. If so, please check `Documentation/admin-guide/media/saa7134-cardlist.rst` for valid choices.

Once you have your card type number, you can pass a modules configuration via a file (usually, it is either `/etc/modules.conf` or some file at `/etc/modules-load.d/`, but the actual place depends on your distribution), with this content:

```
options saa7134 card=13 # Assuming that your card type is #13
```

Changes / Fixes

Please mail to `linux-media AT vger.kernel.org` unified diffs against the linux media git tree:

https://git.linuxtv.org/media_tree.git/

This is done by committing a patch at a clone of the git tree and submitting the patch using `git send-email`. Don't forget to describe at the lots what it changes / which problem it fixes / whatever it is good for ...

Known Problems

- The tuner for the flyvideos isn't detected automatically and the default might not work for you depending on which version you have. There is a `tuner= insmod` option to override the driver's default.

Credits

andrew.stevens@philips.com + werner.leeb@philips.com for providing saa7134 hardware specs and sample board.

1.1.6.19 The Silicon Labs Si470x FM Radio Receivers driver

Copyright © 2009 Tobias Lorenz <tobias.lorenz@gmx.net>

Information from Silicon Labs

Silicon Laboratories is the manufacturer of the radio ICs, that nowadays are the most often used radio receivers in cell phones. Usually they are connected with I2C. But SiLabs also provides a reference design, which integrates this IC, together with a small microcontroller C8051F321, to form a USB radio. Part of this reference design is also a radio application in binary and source code. The software also contains an automatic firmware upgrade to the most current version. Information on these can be downloaded here: <http://www.silabs.com/usbradio>

Supported ICs

The following ICs have a very similar register set, so that they are or will be supported some-when by the driver:

- Si4700: FM radio receiver
- Si4701: FM radio receiver, RDS Support
- Si4702: FM radio receiver
- Si4703: FM radio receiver, RDS Support
- Si4704: FM radio receiver, no external antenna required
- Si4705: FM radio receiver, no external antenna required, RDS support, Dig I/O
- **Si4706: Enhanced FM RDS/TMC radio receiver, no external antenna required, RDS Support**
- Si4707: Dedicated weather band radio receiver with SAME decoder, RDS Support
- Si4708: Smallest FM receivers
- Si4709: Smallest FM receivers, RDS Support

More information on these can be downloaded here: <http://www.silabs.com/products/mcu/Pages/USBFMRadioRD.aspx>

Supported USB devices

Currently the following USB radios (vendor:product) with the Silicon Labs si470x chips are known to work:

- 10c4:818a: Silicon Labs USB FM Radio Reference Design
- 06e1:a155: ADS/Tech FM Radio Receiver (formerly Instant FM Music) (RDX-155-EF)
- 1b80:d700: KWorld USB FM Radio SnapMusic Mobile 700 (FM700)
- 10c5:819a: Sanei Electric, Inc. FM USB Radio (sold as DealExtreme.com PCear)

Software

Testing is usually done with most application under Debian/testing:

- ftools - Utility for managing FM tuner cards
- gnomeradio - FM-radio tuner for the GNOME desktop
- gradio - GTK FM radio tuner
- kradio - Comfortable Radio Application for KDE
- radio - ncurses-based radio application
- mplayer - The Ultimate Movie Player For Linux
- v4l2-ctl - Collection of command line video4linux utilities

For example, you can use:

```
v4l2-ctl -d /dev/radio0 --set-ctrl=volume=10,mute=0 --set-freq=95.21 --all
```

There is also a library `libv4l`, which can be used. It's going to have a function for frequency seeking, either by using hardware functionality as in `radio-si470x` or by implementing a function as we currently have in every of the mentioned programs. Somewhen the radio programs should make use of `libv4l`.

For processing RDS information, there is a project ongoing at: <http://rdsd.berlios.de/>

There is currently no project for making TMC sentences human readable.

Audio Listing

USB Audio is provided by the ALSA `snd_usb_audio` module. It is recommended to also select `SND_USB_AUDIO`, as this is required to get sound from the radio. For listing you have to redirect the sound, for example using one of the following commands. Please adjust the audio devices to your needs (`/dev/dsp*` and `hw:x,x`).

If you just want to test audio (very poor quality):

```
cat /dev/dsp1 > /dev/dsp
```

If you use `sox` + `OSS` try:

```
sox -2 --endian little -r 96000 -t oss /dev/dsp1 -t oss /dev/dsp
```

or using `sox` + `alsa`:

```
sox --endian little -c 2 -S -r 96000 -t alsa hw:1 -t alsa -r 96000 hw:0
```

If you use `arts` try:

```
arecord -D hw:1,0 -r96000 -c2 -f S16_LE | artsdsp aplay -B -
```

If you use `mplayer` try:

```
mplayer -radio adevice=hw=1.0:arate=96000 \  
-rawaudio rate=96000 \  
radio://<frequency>/capture
```

Module Parameters

After loading the module, you still have access to some of them in the `sysfs` mount under `/sys/module/radio_si470x/parameters`. The contents of read-only files (0444) are not updated, even if `space`, `band` and `de` are changed using private video controls. The others are runtime changeable.

Errors

Increase `tune_timeout`, if you often get -EIO errors.

When timed out or band limit is reached, `hw_freq_seek` returns -EAGAIN.

If you get any errors from `snd_usb_audio`, please report them to the ALSA people.

Open Issues

V4L minor device allocation and parameter setting is not perfect. A solution is currently under discussion.

There is an USB interface for downloading/uploading new firmware images. Support for it can be implemented using the `request_firmware` interface.

There is a RDS interrupt mode. The driver is already using the same interface for polling RDS information, but is currently not using the interrupt mode.

There is a LED interface, which can be used to override the LED control programmed in the firmware. This can be made available using the LED support functions in the kernel.

Other useful information and links

<http://www.silabs.com/usbradio>

1.1.6.20 The Silicon Labs Si4713 FM Radio Transmitter Driver

Copyright © 2009 Nokia Corporation

Contact: Eduardo Valentin <eduardo.valentin@nokia.com>

Information about the Device

This chip is a Silicon Labs product. It is a I2C device, currently on 0x63 address. Basically, it has transmission and signal noise level measurement features.

The Si4713 integrates transmit functions for FM broadcast stereo transmission. The chip also allows integrated receive power scanning to identify low signal power FM channels.

The chip is programmed using commands and responses. There are also several properties which can change the behavior of this chip.

Users must comply with local regulations on radio frequency (RF) transmission.

Device driver description

There are two modules to handle this device. One is a I2C device driver and the other is a platform driver.

The I2C device driver exports a v4l2-subdev interface to the kernel. All properties can also be accessed by v4l2 extended controls interface, by using the v4l2-subdev calls (`g_ext_ctrls`, `s_ext_ctrls`).

The platform device driver exports a v4l2 radio device interface to user land. So, it uses the I2C device driver as a sub device in order to send the user commands to the actual device. Basically it is a wrapper to the I2C device driver.

Applications can use v4l2 radio API to specify frequency of operation, mute state, etc. But mostly of its properties will be present in the extended controls.

When the v4l2 mute property is set to 1 (true), the driver will turn the chip off.

Properties description

The properties can be accessed using v4l2 extended controls. Here is an output from `v4l2-ctl` util:

```
/ # v4l2-ctl -d /dev/radio0 --all -L
Driver Info:
  Driver name      : radio-si4713
  Card type       : Silicon Labs Si4713 Modulator
  Bus info        :
  Driver version  : 0
  Capabilities    : 0x00080800
                   RDS Output
                   Modulator
Audio output: 0 (FM Modulator Audio Out)
Frequency: 1408000 (88.000000 MHz)
Video Standard = 0x00000000
Modulator:
  Name             : FM Modulator
  Capabilities     : 62.5 Hz stereo rds
  Frequency range  : 76.0 MHz - 108.0 MHz
  Subchannel modulation: stereo+rds

User Controls

          mute (bool) : default=1 value=0

FM Radio Modulator Controls

  rds_signal_deviation (int) : min=0 max=90000 step=10 default=200 value=200
  ↪ flags=slider
  rds_program_id (int) : min=0 max=65535 step=1 default=0 value=0
  rds_program_type (int) : min=0 max=31 step=1 default=0 value=0
  rds_ps_name (str) : min=0 max=96 step=8 value='si4713 '
  rds_radio_text (str) : min=0 max=384 step=32 value=''
audio_limiter_feature_enabled (bool) : default=1 value=1
audio_limiter_release_time (int) : min=250 max=102390 step=50 default=5010 value=5010
  ↪ flags=slider
```



```

    audio_limiter_deviation (int) : min=0 max=90000 step=10 default=66250
↳value=66250 flags=slider
audio_compression_feature_enabl (bool) : default=1 value=1
    audio_compression_gain (int) : min=0 max=20 step=1 default=15 value=15
↳flags=slider
audio_compression_threshold (int) : min=-40 max=0 step=1 default=-40 value=-40
↳flags=slider
audio_compression_attack_time (int) : min=0 max=5000 step=500 default=0 value=0
↳flags=slider
audio_compression_release_time (int) : min=100000 max=1000000 step=100000
↳default=1000000 value=1000000 flags=slider
pilot_tone_feature_enabled (bool) : default=1 value=1
    pilot_tone_deviation (int) : min=0 max=90000 step=10 default=6750 value=6750
↳flags=slider
    pilot_tone_frequency (int) : min=0 max=19000 step=1 default=19000 value=19000
↳flags=slider
    pre_emphasis_settings (menu) : min=0 max=2 default=1 value=1
    tune_power_level (int) : min=0 max=120 step=1 default=88 value=88 flags=slider
    tune_antenna_capacitor (int) : min=0 max=191 step=1 default=0 value=110
↳flags=slider

```

Here is a summary of them:

- Pilot is an audible tone sent by the device.
- `pilot_frequency` - Configures the frequency of the stereo pilot tone.
- `pilot_deviation` - Configures pilot tone frequency deviation level.
- `pilot_enabled` - Enables or disables the pilot tone feature.
- The si4713 device is capable of applying audio compression to the transmitted signal.
- `acomp_enabled` - Enables or disables the audio dynamic range control feature.
- `acomp_gain` - Sets the gain for audio dynamic range control.
- `acomp_threshold` - Sets the threshold level for audio dynamic range control.
- `acomp_attack_time` - Sets the attack time for audio dynamic range control.
- `acomp_release_time` - Sets the release time for audio dynamic range control.
- Limiter setups audio deviation limiter feature. Once a over deviation occurs, it is possible to adjust the front-end gain of the audio input and always prevent over deviation.
- `limiter_enabled` - Enables or disables the limiter feature.
- `limiter_deviation` - Configures audio frequency deviation level.
- `limiter_release_time` - Sets the limiter release time.
- Tuning power
- `power_level` - Sets the output power level for signal transmission. `antenna_capacitor` - This selects the value of antenna tuning capacitor manually or automatically if set to zero.
- RDS related
- `rds_ps_name` - Sets the RDS ps name field for transmission.
- `rds_radio_text` - Sets the RDS radio text for transmission.

- `rds_pi` - Sets the RDS PI field for transmission.
- `rds_pty` - Sets the RDS PTY field for transmission.
- Region related
- `preemphasis` - sets the preemphasis to be applied for transmission.

RNL

This device also has an interface to measure received noise level. To do that, you should `ioctl` the device node. Here is an code of example:

```
int main (int argc, char *argv[])
{
    struct si4713_rnl rnl;
    int fd = open("/dev/radio0", O_RDWR);
    int rval;

    if (argc < 2)
        return -EINVAL;

    if (fd < 0)
        return fd;

    sscanf(argv[1], "%d", &rnl.frequency);

    rval = ioctl(fd, SI4713_IOC_MEASURE_RNL, &rnl);
    if (rval < 0)
        return rval;

    printf("received noise level: %d\n", rnl.rnl);

    close(fd);
}
```

The `struct si4713_rnl` and `SI4713_IOC_MEASURE_RNL` are defined under `include/linux/platform_data/media/si4713.h`.

Stereo/Mono and RDS subchannels

The device can also be configured using the available sub channels for transmission. To do that use S/G_MODULATOR ioctl and configure txsubchans properly. Refer to the V4L2 API specification for proper use of this ioctl.

Testing

Testing is usually done with v4l2-ctl utility for managing FM tuner cards. The tool can be found in v4l-dvb repository under v4l2-apps/util directory.

Example for setting rds ps name:

```
# v4l2-ctl -d /dev/radio0 --set-ctrl=rds_ps_name="Dummy"
```

1.1.6.21 The SI476x Driver

Copyright © 2013 Andrey Smirnov <andrew.smirnov@gmail.com>

TODO for the driver

- According to the SiLabs' datasheet it is possible to update the firmware of the radio chip in the run-time, thus bringing it to the most recent version. Unfortunately I couldn't find any mentioning of the said firmware update for the old chips that I tested the driver against, so for chips like that the driver only exposes the old functionality.

Parameters exposed over debugfs

SI476x allow user to get multiple characteristics that can be very useful for EoL testing/RF performance estimation, parameters that have very little to do with V4L2 subsystem. Such parameters are exposed via debugfs and can be accessed via regular file I/O operations.

The drivers exposes following files:

- /sys/kernel/debug/<device-name>/acf This file contains ACF(Automatically Controlled Features) status information. The contents of the file is binary data of the following layout:

Offset	Name	Description
0x00	blend_int	Flag, set when stereo separation has crossed below the blend threshold
0x01	hblend_int	Flag, set when HiBlend cutoff frequency is lower than threshold
0x02	hicut_int	Flag, set when HiCut cutoff frequency is lower than threshold
0x03	chbw_int	Flag, set when channel filter bandwidth is less than threshold
0x04	softmute_int	Flag indicating that softmute attenuation has increased above softmute threshold
0x05	smute	0 - Audio is not soft muted 1 - Audio is soft muted
0x06	smattn	Soft mute attenuation level in dB
0x07	chbw	Channel filter bandwidth in kHz
0x08	hicut	HiCut cutoff frequency in units of 100Hz
0x09	hiblend	HiBlend cutoff frequency in units of 100 Hz
0x10	pilot	0 - Stereo pilot is not present 1 - Stereo pilot is present
0x11	stblend	Stereo blend in %

- /sys/kernel/debug/<device-name>/rds_blkcnt This file contains statistics about RDS receptions. It's binary data has the following layout:

Offset	Name	Description
0x00	expected	Number of expected RDS blocks
0x02	received	Number of received RDS blocks
0x04	uncorrectable	Number of uncorrectable RDS blocks

- /sys/kernel/debug/<device-name>/agc This file contains information about parameters pertaining to AGC(Automatic Gain Control)

The layout is:

Offset	Name	Description
0x00	mxhi	0 - FM Mixer PD high threshold is not tripped 1 - FM Mixer PD high threshold is tripped
0x01	mxlo	ditto for FM Mixer PD low
0x02	lnahi	ditto for FM LNA PD high
0x03	lnalo	ditto for FM LNA PD low
0x04	fmagc1	FMAGC1 attenuator resistance (see datasheet for more detail)
0x05	fmagc2	ditto for FMAGC2
0x06	pgagain	PGA gain in dB
0x07	fmwblang	FM/WB LNA Gain in dB

- /sys/kernel/debug/<device-name>/rsq This file contains information about parameters pertaining to RSQ(Received Signal Quality)

The layout is:

Offset	Name	Description
0x00	multhint	0 - multipath value has not crossed the Multipath high threshold 1 - multipath value has crossed the Multipath high threshold
0x01	multlint	ditto for Multipath low threshold
0x02	snrhint	0 - received signal's SNR has not crossed high threshold 1 - received signal's SNR has crossed high threshold
0x03	snrlint	ditto for low threshold
0x04	rssihint	ditto for RSSI high threshold
0x05	rssilint	ditto for RSSI low threshold
0x06	bltf	Flag indicating if seek command reached/wrapped seek band limit
0x07	snr_ready	Indicates that SNR metrics is ready
0x08	rssiready	ditto for RSSI metrics
0x09	injside	0 - Low-side injection is being used 1 - High-side injection is used
0x10	afcr1	Flag indicating if AFC rails
0x11	valid	Flag indicating if channel is valid
0x12	readfreq	Current tuned frequency
0x14	freqoff	Signed frequency offset in units of 2ppm
0x15	rssi	Signed value of RSSI in dBuV
0x16	snr	Signed RF SNR in dB
0x17	issi	Signed Image Strength Signal indicator
0x18	lassi	Signed Low side adjacent Channel Strength indicator
0x19	hassi	ditto for High side
0x20	mult	Multipath indicator
0x21	dev	Frequency deviation
0x24	assi	Adjacent channel SSI
0x25	usn	Ultrasonic noise indicator
0x26	pilotdev	Pilot deviation in units of 100 Hz
0x27	rdsdev	ditto for RDS
0x28	assidev	ditto for ASSI
0x29	strongdev	Frequency deviation
0x30	rdspi	RDS PI code

- `/sys/kernel/debug/<device-name>/rsq_primary` This file contains information about parameters pertaining to RSQ(Received Signal Quality) for primary tuner only. Layout is as the one above.

1.1.6.22 The Virtual Media Controller Driver (vimc)

The vimc driver emulates complex video hardware using the V4L2 API and the Media API. It has a capture device and three subdevices: sensor, debayer and scaler.

Topology

The topology is hardcoded, although you could modify it in vimc-core and recompile the driver to achieve your own topology. This is the default topology:

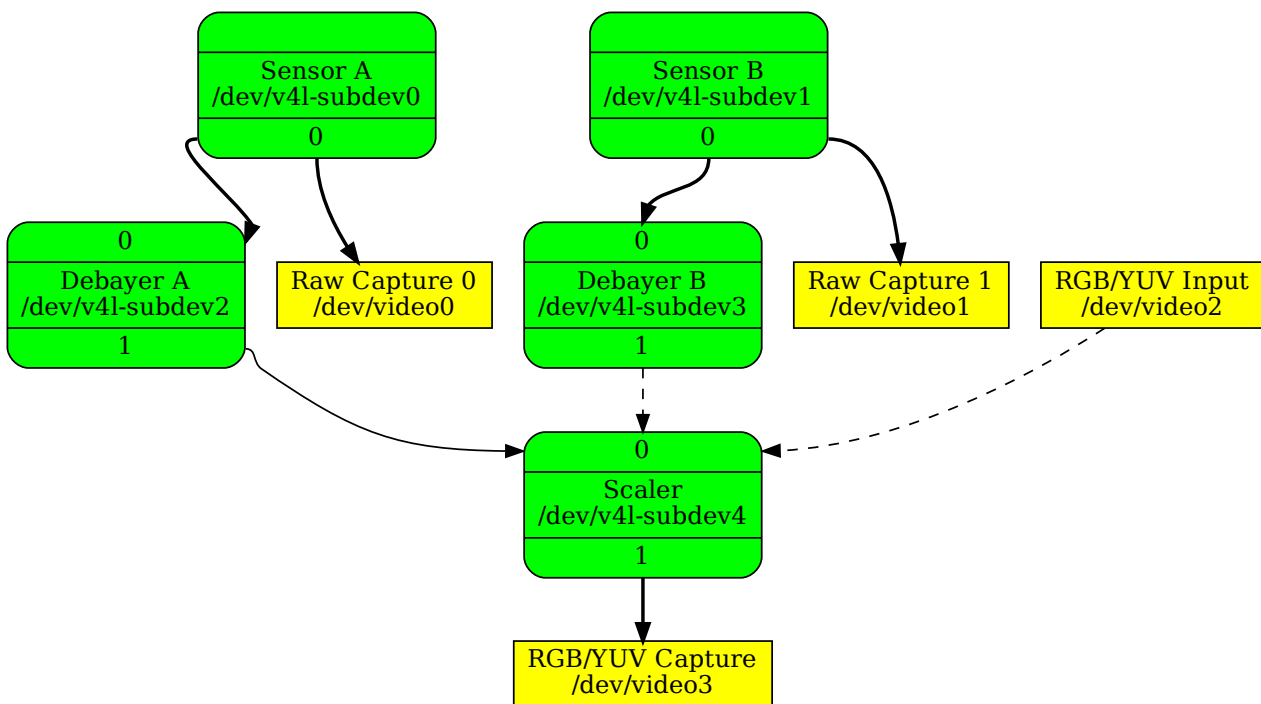


Fig. 7: Media pipeline graph on vimc

Configuring the topology

Each subdevice will come with its default configuration (pixelformat, height, width, ...). One needs to configure the topology in order to match the configuration on each linked subdevice to stream frames through the pipeline. If the configuration doesn't match, the stream will fail. The v4l-utils package is a bundle of user-space applications, that comes with media-ctl and v4l2-ctl that can be used to configure the vimc configuration. This sequence of commands fits for the default topology:

```

media-ctl -d platform:vimc -V '"Sensor A":0[fmt:SBGGR8_1X8/640x480]'
media-ctl -d platform:vimc -V '"Debayer A":0[fmt:SBGGR8_1X8/640x480]'
media-ctl -d platform:vimc -V '"Sensor B":0[fmt:SBGGR8_1X8/640x480]'
media-ctl -d platform:vimc -V '"Debayer B":0[fmt:SBGGR8_1X8/640x480]'
v4l2-ctl -z platform:vimc -d "RGB/YUV Capture" -v width=1920,height=1440
  
```

```
v4l2-ctl -z platform:vimc -d "Raw Capture 0" -v pixelformat=BA81
v4l2-ctl -z platform:vimc -d "Raw Capture 1" -v pixelformat=BA81
```

Subdevices

Subdevices define the behavior of an entity in the topology. Depending on the subdevice, the entity can have multiple pads of type source or sink.

vimc-sensor: Generates images in several formats using video test pattern generator. Exposes:

- 1 Pad source

vimc-debayer: Transforms images in bayer format into a non-bayer format. Exposes:

- 1 Pad sink
- 1 Pad source

vimc-scaler: Re-size the image to meet the source pad resolution. E.g.: if the sync pad is configured to 360x480 and the source to 1280x720, the image will be stretched to fit the source resolution. Works for any resolution within the vimc limitations (even shrinking the image if necessary). Exposes:

- 1 Pad sink
- 1 Pad source

vimc-capture: Exposes node /dev/videoX to allow userspace to capture the stream. Exposes:

- 1 Pad sink
- 1 Pad source

1.1.6.23 The Virtual Video Test Driver (vivid)

This driver emulates video4linux hardware of various types: video capture, video output, vbi capture and output, metadata capture and output, radio receivers and transmitters, touch capture and a software defined radio receiver. In addition a simple framebuffer device is available for testing capture and output overlays.

Up to 64 vivid instances can be created, each with up to 16 inputs and 16 outputs.

Each input can be a webcam, TV capture device, S-Video capture device or an HDMI capture device. Each output can be an S-Video output device or an HDMI output device.

These inputs and outputs act exactly as a real hardware device would behave. This allows you to use this driver as a test input for application development, since you can test the various features without requiring special hardware.

This document describes the features implemented by this driver:

- Support for read()/write(), MMAP, USERPTR and DMABUF streaming I/O.
- A large list of test patterns and variations thereof
- Working brightness, contrast, saturation and hue controls

- Support for the alpha color component
- Full colorspace support, including limited/full RGB range
- All possible control types are present
- Support for various pixel aspect ratios and video aspect ratios
- Error injection to test what happens if errors occur
- Supports crop/compose/scale in any combination for both input and output
- Can emulate up to 4K resolutions
- All Field settings are supported for testing interlaced capturing
- Supports all standard YUV and RGB formats, including two multiplanar YUV formats
- Raw and Sliced VBI capture and output support
- Radio receiver and transmitter support, including RDS support
- Software defined radio (SDR) support
- Capture and output overlay support
- Metadata capture and output support
- Touch capture support

These features will be described in more detail below.

Configuring the driver

By default the driver will create a single instance that has a video capture device with webcam, TV, S-Video and HDMI inputs, a video output device with S-Video and HDMI outputs, one vbi capture device, one vbi output device, one radio receiver device, one radio transmitter device and one SDR device.

The number of instances, devices, video inputs and outputs and their types are all configurable using the following module options:

- `n_devs`:
 - number of driver instances to create. By default set to 1. Up to 64 instances can be created.
- `node_types`:
 - which devices should each driver instance create. An array of hexadecimal values, one for each instance. The default is 0x1d3d. Each value is a bitmask with the following meaning:
 - bit 0: Video Capture node
 - bit 2-3: VBI Capture node: 0 = none, 1 = raw vbi, 2 = sliced vbi, 3 = both
 - bit 4: Radio Receiver node
 - bit 5: Software Defined Radio Receiver node
 - bit 8: Video Output node

- bit 10-11: VBI Output node: 0 = none, 1 = raw vbi, 2 = sliced vbi, 3 = both
- bit 12: Radio Transmitter node
- bit 16: Framebuffer for testing overlays
- bit 17: Metadata Capture node
- bit 18: Metadata Output node
- bit 19: Touch Capture node

So to create four instances, the first two with just one video capture device, the second two with just one video output device you would pass these module options to vivid:

```
n_devs=4 node_types=0x1,0x1,0x100,0x100
```

- **num_inputs:**

the number of inputs, one for each instance. By default 4 inputs are created for each video capture device. At most 16 inputs can be created, and there must be at least one.

- **input_types:**

the input types for each instance, the default is 0xe4. This defines what the type of each input is when the inputs are created for each driver instance. This is a hexadecimal value with up to 16 pairs of bits, each pair gives the type and bits 0-1 map to input 0, bits 2-3 map to input 1, 30-31 map to input 15. Each pair of bits has the following meaning:

- 00: this is a webcam input
- 01: this is a TV tuner input
- 10: this is an S-Video input
- 11: this is an HDMI input

So to create a video capture device with 8 inputs where input 0 is a TV tuner, inputs 1-3 are S-Video inputs and inputs 4-7 are HDMI inputs you would use the following module options:

```
num_inputs=8 input_types=0xffa9
```

- **num_outputs:**

the number of outputs, one for each instance. By default 2 outputs are created for each video output device. At most 16 outputs can be created, and there must be at least one.

- **output_types:**

the output types for each instance, the default is 0x02. This defines what the type of each output is when the outputs are created for each driver instance. This is a hexadecimal value with up to 16 bits, each bit gives the type and bit 0 maps to output 0, bit 1 maps to output 1, bit 15 maps to output 15. The meaning of each bit is as follows:

- 0: this is an S-Video output

- 1: this is an HDMI output

So to create a video output device with 8 outputs where outputs 0-3 are S-Video outputs and outputs 4-7 are HDMI outputs you would use the following module options:

```
num_outputs=8 output_types=0xf0
```

- `vid_cap_nr`:

give the desired videoX start number for each video capture device. The default is -1 which will just take the first free number. This allows you to map capture video nodes to specific videoX device nodes. Example:

```
n_devs=4 vid_cap_nr=2,4,6,8
```

This will attempt to assign `/dev/video2` for the video capture device of the first vivid instance, `video4` for the next up to `video8` for the last instance. If it can't succeed, then it will just take the next free number.

- `vid_out_nr`:

give the desired videoX start number for each video output device. The default is -1 which will just take the first free number.

- `vbi_cap_nr`:

give the desired vbiX start number for each vbi capture device. The default is -1 which will just take the first free number.

- `vbi_out_nr`:

give the desired vbiX start number for each vbi output device. The default is -1 which will just take the first free number.

- `radio_rx_nr`:

give the desired radioX start number for each radio receiver device. The default is -1 which will just take the first free number.

- `radio_tx_nr`:

give the desired radioX start number for each radio transmitter device. The default is -1 which will just take the first free number.

- `sdr_cap_nr`:

give the desired swradioX start number for each SDR capture device. The default is -1 which will just take the first free number.

- `meta_cap_nr`:

give the desired videoX start number for each metadata capture device. The default is -1 which will just take the first free number.

- `meta_out_nr`:

give the desired videoX start number for each metadata output device. The default is -1 which will just take the first free number.

- `touch_cap_nr`:

give the desired v4l-touchX start number for each touch capture device. The default is -1 which will just take the first free number.

- `ccs_cap_mode`:

specify the allowed video capture crop/compose/scaling combination for each driver instance. Video capture devices can have any combination of cropping, composing and scaling capabilities and this will tell the vivid driver which of those is should emulate. By default the user can select this through controls.

The value is either -1 (controlled by the user) or a set of three bits, each enabling (1) or disabling (0) one of the features:

- bit 0:

Enable crop support. Cropping will take only part of the incoming picture.

- bit 1:

Enable compose support. Composing will copy the incoming picture into a larger buffer.

- bit 2:

Enable scaling support. Scaling can scale the incoming picture. The scaler of the vivid driver can enlarge up or down to four times the original size. The scaler is very simple and low-quality. Simplicity and speed were key, not quality.

Note that this value is ignored by webcam inputs: those enumerate discrete framesizes and that is incompatible with cropping, composing or scaling.

- `ccs_out_mode`:

specify the allowed video output crop/compose/scaling combination for each driver instance. Video output devices can have any combination of cropping, composing and scaling capabilities and this will tell the vivid driver which of those is should emulate. By default the user can select this through controls.

The value is either -1 (controlled by the user) or a set of three bits, each enabling (1) or disabling (0) one of the features:

- bit 0:

Enable crop support. Cropping will take only part of the outgoing buffer.

- bit 1:

Enable compose support. Composing will copy the incoming buffer into a larger picture frame.

- bit 2:

Enable scaling support. Scaling can scale the incoming buffer. The scaler of the vivid driver can enlarge up or down to four times the original size. The scaler is very simple and low-quality. Simplicity and speed were key, not quality.

- `multiplanar`:

select whether each device instance supports multi-planar formats, and thus the V4L2 multi-planar API. By default device instances are single-planar.

This module option can override that for each instance. Values are:

- 1: this is a single-planar instance.
- 2: this is a multi-planar instance.

- `vivid_debug`:

enable driver debugging info

- `no_error_inj`:

if set disable the error injecting controls. This option is needed in order to run a tool like `v4l2-compliance`. Tools like that exercise all controls including a control like 'Disconnect' which emulates a USB disconnect, making the device inaccessible and so all tests that `v4l2-compliance` is doing will fail afterwards.

There may be other situations as well where you want to disable the error injection support of `vivid`. When this option is set, then the controls that select crop, compose and scale behavior are also removed. Unless overridden by `ccs_cap_mode` and/or `ccs_out_mode` they will default to enabling crop, compose and scaling.

- `allocators`:

memory allocator selection, default is 0. It specifies the way buffers will be allocated.

- 0: `vmalloc`
- 1: `dma-contig`

- `cache_hints`:

specifies if the device should set queues' user-space cache and memory consistency hint capability (`V4L2_BUF_CAP_SUPPORTS_MMAP_CACHE_HINTS`). The hints are valid only when using `MMAP` streaming I/O. Default is 0.

- 0: forbid hints
- 1: allow hints

Taken together, all these module options allow you to precisely customize the driver behavior and test your application with all sorts of permutations. It is also very suitable to emulate hardware that is not yet available, e.g. when developing software for a new upcoming device.

Video Capture

This is probably the most frequently used feature. The video capture device can be configured by using the module options `num_inputs`, `input_types` and `ccs_cap_mode` (see section 1 for more detailed information), but by default four inputs are configured: a webcam, a TV tuner, an S-Video and an HDMI input, one input for each input type. Those are described in more detail below.

Special attention has been given to the rate at which new frames become available. The jitter will be around 1 jiffie (that depends on the HZ configuration of your kernel, so usually 1/100, 1/250 or 1/1000 of a second), but the long-term behavior is exactly following the framerate. So

a framerate of 59.94 Hz is really different from 60 Hz. If the framerate exceeds your kernel's HZ value, then you will get dropped frames, but the frame/field sequence counting will keep track of that so the sequence count will skip whenever frames are dropped.

Webcam Input

The webcam input supports three framesizes: 320x180, 640x360 and 1280x720. It supports frames per second settings of 10, 15, 25, 30, 50 and 60 fps. Which ones are available depends on the chosen framesize: the larger the framesize, the lower the maximum frames per second.

The initially selected colorspace when you switch to the webcam input will be sRGB.

TV and S-Video Inputs

The only difference between the TV and S-Video input is that the TV has a tuner. Otherwise they behave identically.

These inputs support audio inputs as well: one TV and one Line-In. They both support all TV standards. If the standard is queried, then the Vivid controls 'Standard Signal Mode' and 'Standard' determine what the result will be.

These inputs support all combinations of the field setting. Special care has been taken to faithfully reproduce how fields are handled for the different TV standards. This is particularly noticeable when generating a horizontally moving image so the temporal effect of using interlaced formats becomes clearly visible. For 50 Hz standards the top field is the oldest and the bottom field is the newest in time. For 60 Hz standards that is reversed: the bottom field is the oldest and the top field is the newest in time.

When you start capturing in V4L2_FIELD_ALTERNATE mode the first buffer will contain the top field for 50 Hz standards and the bottom field for 60 Hz standards. This is what capture hardware does as well.

Finally, for PAL/SECAM standards the first half of the top line contains noise. This simulates the Wide Screen Signal that is commonly placed there.

The initially selected colorspace when you switch to the TV or S-Video input will be SMPTE-170M.

The pixel aspect ratio will depend on the TV standard. The video aspect ratio can be selected through the 'Standard Aspect Ratio' Vivid control. Choices are '4x3', '16x9' which will give letterboxed widescreen video and '16x9 Anamorphic' which will give full screen squashed anamorphic widescreen video that will need to be scaled accordingly.

The TV 'tuner' supports a frequency range of 44-958 MHz. Channels are available every 6 MHz, starting from 49.25 MHz. For each channel the generated image will be in color for the +/- 0.25 MHz around it, and in grayscale for +/- 1 MHz around the channel. Beyond that it is just noise. The VIDIOC_G_TUNER ioctl will return 100% signal strength for +/- 0.25 MHz and 50% for +/- 1 MHz. It will also return correct afc values to show whether the frequency is too low or too high.

The audio subchannels that are returned are MONO for the +/- 1 MHz range around a valid channel frequency. When the frequency is within +/- 0.25 MHz of the channel it will return either MONO, STEREO, either MONO | SAP (for NTSC) or LANG1 | LANG2 (for others), or STEREO | SAP.

Which one is returned depends on the chosen channel, each next valid channel will cycle through the possible audio subchannel combinations. This allows you to test the various combinations by just switching channels..

Finally, for these inputs the `v4l2_timecode` struct is filled in in the dequeued `v4l2_buffer` struct.

HDMI Input

The HDMI inputs supports all CEA-861 and DMT timings, both progressive and interlaced, for pixelclock frequencies between 25 and 600 MHz. The field mode for interlaced formats is always `V4L2_FIELD_ALTERNATE`. For HDMI the field order is always top field first, and when you start capturing an interlaced format you will receive the top field first.

The initially selected colorspace when you switch to the HDMI input or select an HDMI timing is based on the format resolution: for resolutions less than or equal to 720x576 the colorspace is set to SMPTE-170M, for others it is set to REC-709 (CEA-861 timings) or sRGB (VESA DMT timings).

The pixel aspect ratio will depend on the HDMI timing: for 720x480 is it set as for the NTSC TV standard, for 720x576 it is set as for the PAL TV standard, and for all others a 1:1 pixel aspect ratio is returned.

The video aspect ratio can be selected through the 'DV Timings Aspect Ratio' Vivid control. Choices are 'Source Width x Height' (just use the same ratio as the chosen format), '4x3' or '16x9', either of which can result in pillarboxed or letterboxed video.

For HDMI inputs it is possible to set the EDID. By default a simple EDID is provided. You can only set the EDID for HDMI inputs. Internally, however, the EDID is shared between all HDMI inputs.

No interpretation is done of the EDID data with the exception of the physical address. See the CEC section for more details.

There is a maximum of 15 HDMI inputs (if there are more, then they will be reduced to 15) since that's the limitation of the EDID physical address.

Video Output

The video output device can be configured by using the module options `num_outputs`, `output_types` and `ccs_out_mode` (see section 1 for more detailed information), but by default two outputs are configured: an S-Video and an HDMI input, one output for each output type. Those are described in more detail below.

Like with video capture the framerate is also exact in the long term.

S-Video Output

This output supports audio outputs as well: “Line-Out 1” and “Line-Out 2”. The S-Video output supports all TV standards.

This output supports all combinations of the field setting.

The initially selected colorspace when you switch to the TV or S-Video input will be SMPTE-170M.

HDMI Output

The HDMI output supports all CEA-861 and DMT timings, both progressive and interlaced, for pixelclock frequencies between 25 and 600 MHz. The field mode for interlaced formats is always `V4L2_FIELD_ALTERNATE`.

The initially selected colorspace when you switch to the HDMI output or select an HDMI timing is based on the format resolution: for resolutions less than or equal to 720x576 the colorspace is set to SMPTE-170M, for others it is set to REC-709 (CEA-861 timings) or sRGB (VESA DMT timings).

The pixel aspect ratio will depend on the HDMI timing: for 720x480 it is set as for the NTSC TV standard, for 720x576 it is set as for the PAL TV standard, and for all others a 1:1 pixel aspect ratio is returned.

An HDMI output has a valid EDID which can be obtained through `VIDIOC_G_EDID`.

There is a maximum of 15 HDMI outputs (if there are more, then they will be reduced to 15) since that's the limitation of the EDID physical address. See also the CEC section for more details.

VBI Capture

There are three types of VBI capture devices: those that only support raw (undecoded) VBI, those that only support sliced (decoded) VBI and those that support both. This is determined by the `node_types` module option. In all cases the driver will generate valid VBI data: for 60 Hz standards it will generate Closed Caption and XDS data. The closed caption stream will alternate between “Hello world!” and “Closed captions test” every second. The XDS stream will give the current time once a minute. For 50 Hz standards it will generate the Wide Screen Signal which is based on the actual Video Aspect Ratio control setting and teletext pages 100-159, one page per frame.

The VBI device will only work for the S-Video and TV inputs, it will give back an error if the current input is a webcam or HDMI.

VBI Output

There are three types of VBI output devices: those that only support raw (undecoded) VBI, those that only support sliced (decoded) VBI and those that support both. This is determined by the `node_types` module option.

The sliced VBI output supports the Wide Screen Signal and the teletext signal for 50 Hz standards and Closed Captioning + XDS for 60 Hz standards.

The VBI device will only work for the S-Video output, it will give back an error if the current output is HDMI.

Radio Receiver

The radio receiver emulates an FM/AM/SW receiver. The FM band also supports RDS. The frequency ranges are:

- FM: 64 MHz - 108 MHz
- AM: 520 kHz - 1710 kHz
- SW: 2300 kHz - 26.1 MHz

Valid channels are emulated every 1 MHz for FM and every 100 kHz for AM and SW. The signal strength decreases the further the frequency is from the valid frequency until it becomes 0% at +/- 50 kHz (FM) or 5 kHz (AM/SW) from the ideal frequency. The initial frequency when the driver is loaded is set to 95 MHz.

The FM receiver supports RDS as well, both using 'Block I/O' and 'Controls' modes. In the 'Controls' mode the RDS information is stored in read-only controls. These controls are updated every time the frequency is changed, or when the tuner status is requested. The Block I/O method uses the `read()` interface to pass the RDS blocks on to the application for decoding.

The RDS signal is 'detected' for +/- 12.5 kHz around the channel frequency, and the further the frequency is away from the valid frequency the more RDS errors are randomly introduced into the block I/O stream, up to 50% of all blocks if you are +/- 12.5 kHz from the channel frequency. All four errors can occur in equal proportions: blocks marked 'CORRECTED', blocks marked 'ERROR', blocks marked 'INVALID' and dropped blocks.

The generated RDS stream contains all the standard fields contained in a 0B group, and also radio text and the current time.

The receiver supports HW frequency seek, either in Bounded mode, Wrap Around mode or both, which is configurable with the "Radio HW Seek Mode" control.

Radio Transmitter

The radio transmitter emulates an FM/AM/SW transmitter. The FM band also supports RDS. The frequency ranges are:

- FM: 64 MHz - 108 MHz
- AM: 520 kHz - 1710 kHz
- SW: 2300 kHz - 26.1 MHz

The initial frequency when the driver is loaded is 95.5 MHz.

The FM transmitter supports RDS as well, both using 'Block I/O' and 'Controls' modes. In the 'Controls' mode the transmitted RDS information is configured using controls, and in 'Block I/O' mode the blocks are passed to the driver using write().

Software Defined Radio Receiver

The SDR receiver has three frequency bands for the ADC tuner:

- 300 kHz
- 900 kHz - 2800 kHz
- 3200 kHz

The RF tuner supports 50 MHz - 2000 MHz.

The generated data contains the In-phase and Quadrature components of a 1 kHz tone that has an amplitude of $\sqrt{2}$.

Metadata Capture

The Metadata capture generates UVC format metadata. The PTS and SCR are transmitted based on the values set in vivid controls.

The Metadata device will only work for the Webcam input, it will give back an error for all other inputs.

Metadata Output

The Metadata output can be used to set brightness, contrast, saturation and hue.

The Metadata device will only work for the Webcam output, it will give back an error for all other outputs.

Touch Capture

The Touch capture generates touch patterns simulating single tap, double tap, triple tap, move from left to right, zoom in, zoom out, palm press (simulating a large area being pressed on a touchpad), and simulating 16 simultaneous touch points.

Controls

Different devices support different controls. The sections below will describe each control and which devices support them.

User Controls - Test Controls

The Button, Boolean, Integer 32 Bits, Integer 64 Bits, Menu, String, Bitmask and Integer Menu are controls that represent all possible control types. The Menu control and the Integer Menu control both have 'holes' in their menu list, meaning that one or more menu items return EINVAL when VIDIOC_QUERYMENU is called. Both menu controls also have a non-zero minimum control value. These features allow you to check if your application can handle such things correctly. These controls are supported for every device type.

User Controls - Video Capture

The following controls are specific to video capture.

The Brightness, Contrast, Saturation and Hue controls actually work and are standard. There is one special feature with the Brightness control: each video input has its own brightness value, so changing input will restore the brightness for that input. In addition, each video input uses a different brightness range (minimum and maximum control values). Switching inputs will cause a control event to be sent with the V4L2_EVENT_CTRL_CH_RANGE flag set. This allows you to test controls that can change their range.

The 'Gain, Automatic' and Gain controls can be used to test volatile controls: if 'Gain, Automatic' is set, then the Gain control is volatile and changes constantly. If 'Gain, Automatic' is cleared, then the Gain control is a normal control.

The 'Horizontal Flip' and 'Vertical Flip' controls can be used to flip the image. These combine with the 'Sensor Flipped Horizontally/Vertically' Vivid controls.

The 'Alpha Component' control can be used to set the alpha component for formats containing an alpha channel.

User Controls - Audio

The following controls are specific to video capture and output and radio receivers and transmitters.

The 'Volume' and 'Mute' audio controls are typical for such devices to control the volume and mute the audio. They don't actually do anything in the vivid driver.

Vivid Controls

These vivid custom controls control the image generation, error injection, etc.

Test Pattern Controls

The Test Pattern Controls are all specific to video capture.

- **Test Pattern:**

selects which test pattern to use. Use the CSC Colorbar for testing colorspace conversions: the colors used in that test pattern map to valid colors in all colorspace. The colorspace conversion is disabled for the other test patterns.
- **OSD Text Mode:**

selects whether the text superimposed on the test pattern should be shown, and if so, whether only counters should be displayed or the full text.
- **Horizontal Movement:**

selects whether the test pattern should move to the left or right and at what speed.
- **Vertical Movement:**

does the same for the vertical direction.
- **Show Border:**

show a two-pixel wide border at the edge of the actual image, excluding letter or pillarboxing.
- **Show Square:**

show a square in the middle of the image. If the image is displayed with the correct pixel and image aspect ratio corrections, then the width and height of the square on the monitor should be the same.
- **Insert SAV Code in Image:**

adds a SAV (Start of Active Video) code to the image. This can be used to check if such codes in the image are inadvertently interpreted instead of being ignored.
- **Insert EAV Code in Image:**

does the same for the EAV (End of Active Video) code.

Capture Feature Selection Controls

These controls are all specific to video capture.

- **Sensor Flipped Horizontally:**

the image is flipped horizontally and the V4L2_IN_ST_HFLIP input status flag is set. This emulates the case where a sensor is for example mounted upside down.
- **Sensor Flipped Vertically:**

the image is flipped vertically and the `V4L2_IN_ST_VFLIP` input status flag is set. This emulates the case where a sensor is for example mounted upside down.

- **Standard Aspect Ratio:**

selects if the image aspect ratio as used for the TV or S-Video input should be 4x3, 16x9 or anamorphic widescreen. This may introduce letterboxing.

- **DV Timings Aspect Ratio:**

selects if the image aspect ratio as used for the HDMI input should be the same as the source width and height ratio, or if it should be 4x3 or 16x9. This may introduce letter or pillarboxing.

- **Timestamp Source:**

selects when the timestamp for each buffer is taken.

- **Colorspace:**

selects which colorspace should be used when generating the image. This only applies if the CSC Colorbar test pattern is selected, otherwise the test pattern will go through unconverted. This behavior is also what you want, since a 75% Colorbar should really have 75% signal intensity and should not be affected by colorspace conversions.

Changing the colorspace will result in the `V4L2_EVENT_SOURCE_CHANGE` to be sent since it emulates a detected colorspace change.

- **Transfer Function:**

selects which colorspace transfer function should be used when generating an image. This only applies if the CSC Colorbar test pattern is selected, otherwise the test pattern will go through unconverted. This behavior is also what you want, since a 75% Colorbar should really have 75% signal intensity and should not be affected by colorspace conversions.

Changing the transfer function will result in the `V4L2_EVENT_SOURCE_CHANGE` to be sent since it emulates a detected colorspace change.

- **Y'CbCr Encoding:**

selects which Y'CbCr encoding should be used when generating a Y'CbCr image. This only applies if the format is set to a Y'CbCr format as opposed to an RGB format.

Changing the Y'CbCr encoding will result in the `V4L2_EVENT_SOURCE_CHANGE` to be sent since it emulates a detected colorspace change.

- **Quantization:**

selects which quantization should be used for the RGB or Y'CbCr encoding when generating the test pattern.

Changing the quantization will result in the `V4L2_EVENT_SOURCE_CHANGE` to be sent since it emulates a detected colorspace change.

- **Limited RGB Range (16-235):**

selects if the RGB range of the HDMI source should be limited or full range. This combines with the Digital Video 'Rx RGB Quantization Range' control and can be used to test what happens if a source provides you with the wrong quantization range information. See the description of that control for more details.

- **Apply Alpha To Red Only:**

apply the alpha channel as set by the 'Alpha Component' user control to the red color of the test pattern only.

- **Enable Capture Cropping:**

enables crop support. This control is only present if the `ccs_cap_mode` module option is set to the default value of -1 and if the `no_error_inj` module option is set to 0 (the default).

- **Enable Capture Composing:**

enables composing support. This control is only present if the `ccs_cap_mode` module option is set to the default value of -1 and if the `no_error_inj` module option is set to 0 (the default).

- **Enable Capture Scaler:**

enables support for a scaler (maximum 4 times upscaling and downscaling). This control is only present if the `ccs_cap_mode` module option is set to the default value of -1 and if the `no_error_inj` module option is set to 0 (the default).

- **Maximum EDID Blocks:**

determines how many EDID blocks the driver supports. Note that the vivid driver does not actually interpret new EDID data, it just stores it. It allows for up to 256 EDID blocks which is the maximum supported by the standard.

- **Fill Percentage of Frame:**

can be used to draw only the top X percent of the image. Since each frame has to be drawn by the driver, this demands a lot of the CPU. For large resolutions this becomes problematic. By drawing only part of the image this CPU load can be reduced.

Output Feature Selection Controls

These controls are all specific to video output.

- **Enable Output Cropping:**

enables crop support. This control is only present if the `ccs_out_mode` module option is set to the default value of -1 and if the `no_error_inj` module option is set to 0 (the default).

- **Enable Output Composing:**

enables composing support. This control is only present if the `ccs_out_mode` module option is set to the default value of -1 and if the `no_error_inj` module option is set to 0 (the default).

- **Enable Output Scaler:**

enables support for a scaler (maximum 4 times upscaling and downscaling). This control is only present if the `ccs_out_mode` module option is set to the default value of -1 and if the `no_error_inj` module option is set to 0 (the default).

Error Injection Controls

The following two controls are only valid for video and vbi capture.

- Standard Signal Mode:

selects the behavior of `VIDIOC_QUERYSTD`: what should it return?

Changing this control will result in the `V4L2_EVENT_SOURCE_CHANGE` to be sent since it emulates a changed input condition (e.g. a cable was plugged in or out).

- Standard:

selects the standard that `VIDIOC_QUERYSTD` should return if the previous control is set to "Selected Standard".

Changing this control will result in the `V4L2_EVENT_SOURCE_CHANGE` to be sent since it emulates a changed input standard.

The following two controls are only valid for video capture.

- DV Timings Signal Mode:

selects the behavior of `VIDIOC_QUERY_DV_TIMINGS`: what should it return?

Changing this control will result in the `V4L2_EVENT_SOURCE_CHANGE` to be sent since it emulates a changed input condition (e.g. a cable was plugged in or out).

- DV Timings:

selects the timings the `VIDIOC_QUERY_DV_TIMINGS` should return if the previous control is set to "Selected DV Timings".

Changing this control will result in the `V4L2_EVENT_SOURCE_CHANGE` to be sent since it emulates changed input timings.

The following controls are only present if the `no_error_inj` module option is set to 0 (the default). These controls are valid for video and vbi capture and output streams and for the SDR capture device except for the Disconnect control which is valid for all devices.

- Wrap Sequence Number:

test what happens when you wrap the sequence number in struct `v4l2_buffer` around.

- Wrap Timestamp:

test what happens when you wrap the timestamp in struct `v4l2_buffer` around.

- Percentage of Dropped Buffers:

sets the percentage of buffers that are never returned by the driver (i.e., they are dropped).

- Disconnect:

emulates a USB disconnect. The device will act as if it has been disconnected. Only after all open filehandles to the device node have been closed will the device become 'connected' again.

- Inject V4L2_BUF_FLAG_ERROR:
when pressed, the next frame returned by the driver will have the error flag set (i.e. the frame is marked corrupt).
- Inject VIDIOC_REQBUFS Error:
when pressed, the next REQBUFS or CREATE_BUFS ioctl call will fail with an error. To be precise: the videobuf2 `queue_setup()` op will return `-EINVAL`.
- Inject VIDIOC_QBUF Error:
when pressed, the next VIDIOC_QBUF or VIDIOC_PREPARE_BUFFER ioctl call will fail with an error. To be precise: the videobuf2 `buf_prepare()` op will return `-EINVAL`.
- Inject VIDIOC_STREAMON Error:
when pressed, the next VIDIOC_STREAMON ioctl call will fail with an error. To be precise: the videobuf2 `start_streaming()` op will return `-EINVAL`.
- Inject Fatal Streaming Error:
when pressed, the streaming core will be marked as having suffered a fatal error, the only way to recover from that is to stop streaming. To be precise: the videobuf2 `vb2_queue_error()` function is called.

VBI Raw Capture Controls

- Interlaced VBI Format:
if set, then the raw VBI data will be interlaced instead of providing it grouped by field.

Digital Video Controls

- Rx RGB Quantization Range:
sets the RGB quantization detection of the HDMI input. This combines with the Vivid 'Limited RGB Range (16-235)' control and can be used to test what happens if a source provides you with the wrong quantization range information. This can be tested by selecting an HDMI input, setting this control to Full or Limited range and selecting the opposite in the 'Limited RGB Range (16-235)' control. The effect is easy to see if the 'Gray Ramp' test pattern is selected.
- Tx RGB Quantization Range:
sets the RGB quantization detection of the HDMI output. It is currently not used for anything in vivid, but most HDMI transmitters would typically have this control.
- Transmit Mode:

sets the transmit mode of the HDMI output to HDMI or DVI-D. This affects the reported colorspace since DVI_D outputs will always use sRGB.

- Display Present:

sets the presence of a “display” on the HDMI output. This affects the `tx_edid_present`, `tx_hotplug` and `tx_rxsense` controls.

FM Radio Receiver Controls

- RDS Reception:

set if the RDS receiver should be enabled.

- RDS Program Type:
- RDS PS Name:
- RDS Radio Text:
- RDS Traffic Announcement:
- RDS Traffic Program:
- RDS Music:

these are all read-only controls. If RDS Rx I/O Mode is set to “Block I/O”, then they are inactive as well. If RDS Rx I/O Mode is set to “Controls”, then these controls report the received RDS data.

Note: The vivid implementation of this is pretty basic: they are only updated when you set a new frequency or when you get the tuner status (`VIDIOC_G_TUNER`).

- Radio HW Seek Mode:

can be one of “Bounded”, “Wrap Around” or “Both”. This determines if `VIDIOC_S_HW_FREQ_SEEK` will be bounded by the frequency range or wrap-around or if it is selectable by the user.

- Radio Programmable HW Seek:

if set, then the user can provide the lower and upper bound of the HW Seek. Otherwise the frequency range boundaries will be used.

- Generate RBDS Instead of RDS:

if set, then generate RBDS (the US variant of RDS) data instead of RDS (European-style RDS). This affects only the PICODE and PTY codes.

- RDS Rx I/O Mode:

this can be “Block I/O” where the RDS blocks have to be read() by the application, or “Controls” where the RDS data is provided by the RDS controls mentioned above.

FM Radio Modulator Controls

- RDS Program ID:
- RDS Program Type:
- RDS PS Name:
- RDS Radio Text:
- RDS Stereo:
- RDS Artificial Head:
- RDS Compressed:
- RDS Dynamic PTY:
- RDS Traffic Announcement:
- RDS Traffic Program:
- RDS Music:

these are all controls that set the RDS data that is transmitted by the FM modulator.

- RDS Tx I/O Mode:

this can be “Block I/O” where the application has to use write() to pass the RDS blocks to the driver, or “Controls” where the RDS data is Provided by the RDS controls mentioned above.

Metadata Capture Controls

- Generate PTS

if set, then the generated metadata stream contains Presentation timestamp.

- Generate SCR

if set, then the generated metadata stream contains Source Clock information.

Video, VBI and RDS Looping

The vivid driver supports looping of video output to video input, VBI output to VBI input and RDS output to RDS input. For video/VBI looping this emulates as if a cable was hooked up between the output and input connector. So video and VBI looping is only supported between S-Video and HDMI inputs and outputs. VBI is only valid for S-Video as it makes no sense for HDMI.

Since radio is wireless this looping always happens if the radio receiver frequency is close to the radio transmitter frequency. In that case the radio transmitter will ‘override’ the emulated radio stations.

Looping is currently supported only between devices created by the same vivid driver instance.

Video and Sliced VBI looping

The way to enable video/VBI looping is currently fairly crude. A 'Loop Video' control is available in the "Vivid" control class of the video capture and VBI capture devices. When checked the video looping will be enabled. Once enabled any video S-Video or HDMI input will show a static test pattern until the video output has started. At that time the video output will be looped to the video input provided that:

- the input type matches the output type. So the HDMI input cannot receive video from the S-Video output.
- the video resolution of the video input must match that of the video output. So it is not possible to loop a 50 Hz (720x576) S-Video output to a 60 Hz (720x480) S-Video input, or a 720p60 HDMI output to a 1080p30 input.
- the pixel formats must be identical on both sides. Otherwise the driver would have to do pixel format conversion as well, and that's taking things too far.
- the field settings must be identical on both sides. Same reason as above: requiring the driver to convert from one field format to another complicated matters too much. This also prohibits capturing with 'Field Top' or 'Field Bottom' when the output video is set to 'Field Alternate'. This combination, while legal, became too complicated to support. Both sides have to be 'Field Alternate' for this to work. Also note that for this specific case the sequence and field counting in struct `v4l2_buffer` on the capture side may not be 100% accurate.
- field settings `V4L2_FIELD_SEQ_TB/BT` are not supported. While it is possible to implement this, it would mean a lot of work to get this right. Since these field values are rarely used the decision was made not to implement this for now.
- on the input side the "Standard Signal Mode" for the S-Video input or the "DV Timings Signal Mode" for the HDMI input should be configured so that a valid signal is passed to the video input.

The framerates do not have to match, although this might change in the future.

By default you will see the OSD text superimposed on top of the looped video. This can be turned off by changing the "OSD Text Mode" control of the video capture device.

For VBI looping to work all of the above must be valid and in addition the vbi output must be configured for sliced VBI. The VBI capture side can be configured for either raw or sliced VBI. Note that at the moment only CC/XDS (60 Hz formats) and WSS (50 Hz formats) VBI data is looped. Teletext VBI data is not looped.

Radio & RDS Looping

As mentioned in section 6 the radio receiver emulates stations are regular frequency intervals. Depending on the frequency of the radio receiver a signal strength value is calculated (this is returned by `VIDIOC_G_TUNER`). However, it will also look at the frequency set by the radio transmitter and if that results in a higher signal strength than the settings of the radio transmitter will be used as if it was a valid station. This also includes the RDS data (if any) that the transmitter 'transmits'. This is received faithfully on the receiver side. Note that when the driver is loaded the frequencies of the radio receiver and transmitter are not identical, so initially no looping takes place.

Cropping, Composing, Scaling

This driver supports cropping, composing and scaling in any combination. Normally which features are supported can be selected through the Vivid controls, but it is also possible to hardcode it when the module is loaded through the `ccs_cap_mode` and `ccs_out_mode` module options. See section 1 on the details of these module options.

This allows you to test your application for all these variations.

Note that the webcam input never supports cropping, composing or scaling. That only applies to the TV/S-Video/HDMI inputs and outputs. The reason is that webcams, including this virtual implementation, normally use `VIDIOC_ENUM_FRAMESIZES` to list a set of discrete framesizes that it supports. And that does not combine with cropping, composing or scaling. This is primarily a limitation of the V4L2 API which is carefully reproduced here.

The minimum and maximum resolutions that the scaler can achieve are 16x16 and (4096 * 4) x (2160 x 4), but it can only scale up or down by a factor of 4 or less. So for a source resolution of 1280x720 the minimum the scaler can do is 320x180 and the maximum is 5120x2880. You can play around with this using the `qv4l2` test tool and you will see these dependencies.

This driver also supports larger 'bytesperline' settings, something that `VIDIOC_S_FMT` allows but that few drivers implement.

The scaler is a simple scaler that uses the Coarse Bresenham algorithm. It's designed for speed and simplicity, not quality.

If the combination of crop, compose and scaling allows it, then it is possible to change crop and compose rectangles on the fly.

Formats

The driver supports all the regular packed and planar 4:4:4, 4:2:2 and 4:2:0 YUYV formats, 8, 16, 24 and 32 RGB packed formats and various multiplanar formats.

The alpha component can be set through the 'Alpha Component' User control for those formats that support it. If the 'Apply Alpha To Red Only' control is set, then the alpha component is only used for the color red and set to 0 otherwise.

The driver has to be configured to support the multiplanar formats. By default the driver instances are single-planar. This can be changed by setting the multiplanar module option, see section 1 for more details on that option.

If the driver instance is using the multiplanar formats/API, then the first single planar format (YUYV) and the multiplanar NV16M and NV61M formats will have a plane that has a non-zero `data_offset` of 128 bytes. It is rare for `data_offset` to be non-zero, so this is a useful feature for testing applications.

Video output will also honor any `data_offset` that the application set.

Capture Overlay

Note: capture overlay support is implemented primarily to test the existing V4L2 capture overlay API. In practice few if any GPUs support such overlays anymore, and neither are they generally needed anymore since modern hardware is so much more capable. By setting flag 0x10000 in the `node_types` module option the `vivid` driver will create a simple framebuffer device that can be used for testing this API. Whether this API should be used for new drivers is questionable.

This driver has support for a destructive capture overlay with bitmap clipping and list clipping (up to 16 rectangles) capabilities. Overlays are not supported for multiplanar formats. It also honors the `struct v4l2_window` field setting: if it is set to `FIELD_TOP` or `FIELD_BOTTOM` and the capture setting is `FIELD_ALTERNATE`, then only the top or bottom fields will be copied to the overlay.

The overlay only works if you are also capturing at that same time. This is a vivid limitation since it copies from a buffer to the overlay instead of filling the overlay directly. And if you are not capturing, then no buffers are available to fill.

In addition, the `pixelformat` of the capture format and that of the framebuffer must be the same for the overlay to work. Otherwise `VIDIOC_OVERLAY` will return an error.

In order to really see what it going on you will need to create two vivid instances: the first with a framebuffer enabled. You configure the capture overlay of the second instance to use the framebuffer of the first, then you start capturing in the second instance. For the first instance you setup the output overlay for the video output, turn on video looping and capture to see the blended framebuffer overlay that's being written to by the second instance. This setup would require the following commands:

```
$ sudo modprobe vivid n_devs=2 node_types=0x10101,0x1
$ v4l2-ctl -d1 --find-fb
/dev/fb1 is the framebuffer associated with base address 0x12800000
$ sudo v4l2-ctl -d2 --set-fbuf fb=1
$ v4l2-ctl -d1 --set-fbuf fb=1
$ v4l2-ctl -d0 --set-fmt-video=pixelformat='AR15'
$ v4l2-ctl -d1 --set-fmt-video-out=pixelformat='AR15'
$ v4l2-ctl -d2 --set-fmt-video=pixelformat='AR15'
$ v4l2-ctl -d0 -i2
$ v4l2-ctl -d2 -i2
$ v4l2-ctl -d2 -c horizontal_movement=4
$ v4l2-ctl -d1 --overlay=1
$ v4l2-ctl -d1 -c loop_video=1
$ v4l2-ctl -d2 --stream-mmap --overlay=1
```

And from another console:

```
$ v4l2-ctl -d1 --stream-out-mmap
```

And yet another console:

```
$ qv4l2
```

and start streaming.

As you can see, this is not for the faint of heart...

Output Overlay

Note: output overlays are primarily implemented in order to test the existing V4L2 output overlay API. Whether this API should be used for new drivers is questionable.

This driver has support for an output overlay and is capable of:

- bitmap clipping,
- list clipping (up to 16 rectangles)
- chromakey
- source chromakey
- global alpha
- local alpha
- local inverse alpha

Output overlays are not supported for multiplanar formats. In addition, the pixel format of the capture format and that of the framebuffer must be the same for the overlay to work. Otherwise `VIDIOC_OVERLAY` will return an error.

Output overlays only work if the driver has been configured to create a framebuffer by setting flag `0x10000` in the `node_types` module option. The created framebuffer has a size of `720x576` and supports `ARGB 1:5:5:5` and `RGB 5:6:5`.

In order to see the effects of the various clipping, chromakeying or alpha processing capabilities you need to turn on video looping and see the results on the capture side. The use of the clipping, chromakeying or alpha processing capabilities will slow down the video loop considerably as a lot of checks have to be done per pixel.

CEC (Consumer Electronics Control)

If there are HDMI inputs then a CEC adapter will be created that has the same number of input ports. This is the equivalent of e.g. a TV that has that number of inputs. Each HDMI output will also create a CEC adapter that is hooked up to the corresponding input port, or (if there are more outputs than inputs) is not hooked up at all. In other words, this is the equivalent of hooking up each output device to an input port of the TV. Any remaining output devices remain unconnected.

The EDID that each output reads reports a unique CEC physical address that is based on the physical address of the EDID of the input. So if the EDID of the receiver has physical address `A.B.0.0`, then each output will see an EDID containing physical address `A.B.C.0` where `C` is 1 to the number of inputs. If there are more outputs than inputs then the remaining outputs have a CEC adapter that is disabled and reports an invalid physical address.

Some Future Improvements

Just as a reminder and in no particular order:

- Add a virtual alsa driver to test audio
- Add virtual sub-devices and media controller support
- Some support for testing compressed video
- Add support to loop raw VBI output to raw VBI input
- Add support to loop teletext sliced VBI output to VBI input
- Fix sequence/field numbering when looping of video with alternate fields
- Add support for V4L2_CID_BG_COLOR for video outputs
- Add ARGB888 overlay support: better testing of the alpha channel
- Improve pixel aspect support in the tpg code by passing a real v4l2_fract
- Use per-queue locks and/or per-device locks to improve throughput
- Add support to loop from a specific output to a specific input across vivid instances
- The SDR radio should use the same ‘frequencies’ for stations as the normal radio receiver, and give back noise if the frequency doesn’t match up with a station frequency
- Make a thread for the RDS generation, that would help in particular for the “Controls” RDS Rx I/O Mode as the read-only RDS controls could be updated in real-time.
- Changing the EDID should cause hotplug detect emulation to happen.

1.1.7 Digital TV driver-specific documentation

1.1.7.1 Avermedia DVB-T on BT878 Release Notes

February 14th 2006

Note: Several other Avermedia devices are supported. For a more broader and updated content about that, please check:

<https://linuxtv.org/wiki/index.php/AVerMedia>

The Avermedia DVB-T

The Avermedia DVB-T is a budget PCI DVB card. It has 3 inputs:

- RF Tuner Input
- Composite Video Input (RCA Jack)
- SVIDEO Input (Mini-DIN)

The RF Tuner Input is the input to the tuner module of the card. The Tuner is otherwise known as the “Frontend” . The Frontend of the Avermedia DVB-T is a Microtune 7202D. A timely post to the linux-dvb mailing list ascertained that the Microtune 7202D is supported by the sp887x driver which is found in the dvb-hw CVS module.

The DVB-T card is based around the BT878 chip which is a very common multimedia bridge and often found on Analogue TV cards. There is no on-board MPEG2 decoder, which means that all MPEG2 decoding must be done in software, or if you have one, on an MPEG2 hardware decoding card or chipset.

Getting the card going

At this stage, it has not been able to ascertain the functionality of the remaining device nodes in respect of the Avermedia DVBT. However, full functionality in respect of tuning, receiving and supplying the MPEG2 data stream is possible with the currently available versions of the driver. It may be possible that additional functionality is available from the card (i.e. viewing the additional analogue inputs that the card presents), but this has not been tested yet. If I get around to this, I’ll update the document with whatever I find.

To power up the card, load the following modules in the following order:

- modprobe btvt (normally loaded automatically)
- modprobe dvb-bt8xx (or place dvb-bt8xx in /etc/modules)

Insertion of these modules into the running kernel will activate the appropriate DVB device nodes. It is then possible to start accessing the card with utilities such as scan, tzap, dvbstream etc.

The frontend module sp887x.o, requires an external firmware. Please use the command “get_dvb_firmware sp887x” to download it. Then copy it to /usr/lib/hotplug/firmware or /lib/firmware/ (depending on configuration of firmware hotplug).

Known Limitations

At present I can say with confidence that the frontend tunes via /dev/dvb/adapter{x}/frontend0 and supplies an MPEG2 stream via /dev/dvb/adapter{x}/dvr0. I have not tested the functionality of any other part of the card yet. I will do so over time and update this document.

There are some limitations in the i2c layer due to a returned error message inconsistency. Although this generates errors in dmesg and the system logs, it does not appear to affect the ability of the frontend to function correctly.

Further Update

dvbstream and VideoLAN Client on windows works a treat with DVB, in fact this is currently serving as my main way of viewing DVB-T at the moment. Additionally, VLC is happily decoding HDTV signals, although the PC is dropping the odd frame here and there - I assume due to processing capability - as all the decoding is being done under windows in software.

Many thanks to Nigel Pearson for the updates to this document since the recent revision of the driver.

1.1.7.2 How to get the bt8xx cards working

Authors: Richard Walker, Jamie Honan, Michael Hunold, Manu Abraham, Uwe Bugla, Michael Krufky

General information

This class of cards has a bt878a as the PCI interface, and require the bttv driver for accessing the i2c bus and the gpio pins of the bt8xx chipset.

Please see [Documentation/admin-guide/media/bttv-cardlist.rst](#) for a complete list of Cards based on the Conexant Bt8xx PCI bridge supported by the Linux Kernel.

In order to be able to compile the kernel, some config options should be enabled:

```
./scripts/config -e PCI
./scripts/config -e INPUT
./scripts/config -m I2C
./scripts/config -m MEDIA_SUPPORT
./scripts/config -e MEDIA_PCI_SUPPORT
./scripts/config -e MEDIA_ANALOG_TV_SUPPORT
./scripts/config -e MEDIA_DIGITAL_TV_SUPPORT
./scripts/config -e MEDIA_RADIO_SUPPORT
./scripts/config -e RC_CORE
./scripts/config -m VIDEO_BT848
./scripts/config -m DVB_BT8XX
```

If you want to automatically support all possible variants of the Bt8xx cards, you should also do:

```
./scripts/config -e MEDIA_SUBDRV_AUTOSELECT
```

Note: Please use the following options with care as deselection of drivers which are in fact necessary may result in DVB devices that cannot be tuned due to lack of driver support.

If your goal is to just support an specific board, you may, instead, disable `MEDIA_SUBDRV_AUTOSELECT` and manually select the frontend drivers required by your board. With that, you can save some RAM.

You can do that by calling `make xconfig/qconfig/menuconfig` and look at the options on those menu options (only enabled if `Autoselect ancillary drivers` is disabled:

- 1) Device drivers => Multimedia support => Customize TV tuners
- 2) Device drivers => Multimedia support => Customize DVB frontends

Then, on each of the above menu, please select your card-specific frontend and tuner modules.

Loading Modules

Regular case: If the bttv driver detects a bt8xx-based DVB card, all frontend and backend modules will be loaded automatically.

Exceptions are:

- Old TV cards without EEPROMs, sharing a common PCI subsystem ID;
- Old TwinHan DST cards or clones with or without CA slot and not containing an Eeprom.

In the following cases overriding the PCI type detection for bttv and for dvb-bt8xx drivers by passing modprobe parameters may be necessary.

Running TwinHan and Clones

As shown at Documentation/admin-guide/media/bttv-cardlist.rst, TwinHan and clones use card=113 modprobe parameter. So, in order to properly detect it for devices without EEPROM, you should use:

```
$ modprobe bttv card=113
$ modprobe dst
```

Useful parameters for verbosity level and debugging the dst module:

```
verbose=0:      messages are disabled
    1:          only error messages are displayed
    2:          notifications are displayed
    3:          other useful messages are displayed
    4:          debug setting
dst_addons=0:   card is a free to air (FTA) card only
    0x20:      card has a conditional access slot for scrambled channels
dst_algo=0:     (default) Software tuning algorithm
    1:          Hardware tuning algorithm
```

The autodetected values are determined by the cards' "response string".

In your logs see f. ex.: dst_get_device_id: Recognize [DSTMCI].

For bug reports please send in a complete log with verbose=4 activated. Please also see Documentation/admin-guide/media/ci.rst.

Running multiple cards

See Documentation/admin-guide/media/bttv-cardlist.rst for a complete list of Card ID. Some examples:

Brand name	ID
Pinnacle PCTV Sat	94
Nebula Electronics Digi TV	104
pcHDTV HD-2000 TV	112
Twinhan DST and clones	113
Avermedia AverTV DVB-T 77:	123
Avermedia AverTV DVB-T 761	124
DViCO FusionHDTV DVB-T Lite	128
DViCO FusionHDTV 5 Lite	135

Note: When you have multiple cards, the order of the card ID should match the order where they're detected by the system. Please notice that removing/inserting other PCI cards may change the detection order.

Example:

```
$ modprobe bttv card=113 card=135
```

In case of further problems please subscribe and send questions to the mailing list: linux-media@vger.kernel.org.

Probing the cards with broken PCI subsystem ID

There are some TwinHan cards whose EEPROM has become corrupted for some reason. The cards do not have a correct PCI subsystem ID. Still, it is possible to force probing the cards with:

```
$ echo 109e 0878 $subvendor $subdevice > \  
/sys/bus/pci/drivers/bt878/new_id
```

The two numbers there are:

```
109e: PCI_VENDOR_ID_BROOKTREE  
0878: PCI_DEVICE_ID_BROOKTREE_878
```

1.1.7.3 Firmware files for lmedm04 cards

To extract firmware for the DM04/QQBOX you need to copy the following file(s) to this directory.

For DM04+/QQBOX LME2510C (Sharp 7395 Tuner)

The Sharp 7395 driver can be found in windows/system32/drivers

US2A0D.sys (dated 17 Mar 2009)

and run:

```
scripts/get_dvb_firmware lme2510c_s7395
```

will produce dvb-usb-lme2510c-s7395.fw

An alternative but older firmware can be found on the driver disk DVB-S_EN_3.5A in BDADriver/driver

LMEBDA_DVBS7395C.sys (dated 18 Jan 2008)

and run:

```
./get_dvb_firmware lme2510c_s7395_old
```

will produce dvb-usb-lme2510c-s7395.fw

The LG firmware can be found on the driver disk DM04+_5.1A[LG] in BDADriver/driver

For DM04 LME2510 (LG Tuner)

LMEBDA_DVBS.sys (dated 13 Nov 2007)

and run:

```
./get_dvb_firmware lme2510_lg
```

will produce dvb-usb-lme2510-lg.fw

Other LG firmware can be extracted manually from US280D.sys only found in windows/system32/drivers

```
dd if=US280D.sys ibs=1 skip=42360 count=3924 of=dvb-usb-lme2510-lg.fw
```

For DM04 LME2510C (LG Tuner)

```
dd if=US280D.sys ibs=1 skip=35200 count=3850 of=dvb-usb-lme2510c-lg.fw
```

The Sharp 0194 tuner driver can be found in windows/system32/drivers

US290D.sys (dated 09 Apr 2009)

For LME2510

```
dd if=US290D.sys ibs=1 skip=36856 count=3976 of=dvb-usb-lme2510-s0194.fw
```

For LME2510C

```
dd if=US290D.sys ibs=1 skip=33152 count=3697 of=dvb-usb-lme2510c-s0194.fw
```

The m88rs2000 tuner driver can be found in windows/system32/drivers

US2B0D.sys (dated 29 Jun 2010)

```
dd if=US2B0D.sys ibs=1 skip=34432 count=3871 of=dvb-usb-lme2510c-rs2000.fw
```

We need to modify id of rs2000 firmware or it will warm boot id 3344:1120.

```
echo -ne \\xF0\\x22 | dd conv=notrunc bs=1 count=2 seek=266 of=dvb-usb-lme2510c-rs2000.  
↪fw
```

Copy the firmware file(s) to /lib/firmware

1.1.7.4 Opera firmware

Author: Marco Gittler <g.marco@freenet.de>

To extract the firmware for the Opera DVB-S1 USB-Box you need to copy the files:

2830SCap2.sys 2830SLoad2.sys

from the windriver disk into this directory.

Then run:

```
scripts/get_dvb_firmware opera1
```

and after that you have 2 files:

dvb-usb-opera-01.fw dvb-usb-opera1-fpga-01.fw

in here.

Copy them into /lib/firmware/ .

After that the driver can load the firmware (if you have enabled firmware loading in kernel config and have hotplug running).

1.1.7.5 How to set up the Technisat/B2C2 Flexcop devices

Note: This documentation is outdated.

Author: Uwe Bugla <uwe.bugla@gmx.de> August 2009

Find out what device you have

Important Notice: The driver does NOT support Technisat USB 2 devices!

First start your linux box with a shipped kernel:

```
lspci -vvv for a PCI device (lsusb -vvv for an USB device) will show you for example:
02:0b.0 Network controller: Techsan Electronics Co Ltd B2C2 FlexCopII DVB chip /
Technisat SkyStar2 DVB card (rev 02)
```

```
dmesg | grep frontend may show you for example:
DVB: registering frontend 0 (Conexant CX24123/CX24109)...
```

Kernel compilation:

If the Flexcop / Technisat is the only DVB / TV / Radio device in your box get rid of unnecessary modules and check this one:

Multimedia support => Customise analog and hybrid tuner modules to build

In this directory uncheck every driver which is activated there (except Simple tuner support for ATSC 3rd generation only -> see case 9 please).

Then please activate:

- Main module part:

Multimedia support => DVB/ATSC adapters => Technisat/B2C2 FlexcopII(b) and FlexCopIII adapters

- 1) => Technisat/B2C2 Air/Sky/Cable2PC PCI (PCI card) or
- 2) => Technisat/B2C2 Air/Sky/Cable2PC USB (USB 1.1 adapter) and for troubleshooting purposes:
- 3) => Enable debug for the B2C2 FlexCop drivers

- Frontend / Tuner / Demodulator module part:

Multimedia support => DVB/ATSC adapters => Customise the frontend modules to build Customise DVB frontends =>

- SkyStar DVB-S Revision 2.3:
 - 1) => Zarlink VP310/MT312/ZL10313 based
 - 2) => Generic I2C PLL based tuners
- SkyStar DVB-S Revision 2.6:

- 1) => ST STV0299 based
- 2) => Generic I2C PLL based tuners
- SkyStar DVB-S Revision 2.7:
 - 1) => Samsung S5H1420 based
 - 2) => Integrant ITD1000 Zero IF tuner for DVB-S/DSS
 - 3) => ISL6421 SEC controller
- SkyStar DVB-S Revision 2.8:
 - 1) => Conexant CX24123 based
 - 2) => Conexant CX24113/CX24128 tuner for DVB-S/DSS
 - 3) => ISL6421 SEC controller
- AirStar DVB-T card:
 - 1) => Zarlink MT352 based
 - 2) => Generic I2C PLL based tuners
- CableStar DVB-C card:
 - 1) => ST STV0297 based
 - 2) => Generic I2C PLL based tuners
- AirStar ATSC card 1st generation:
 - 1) => Broadcom BCM3510
- AirStar ATSC card 2nd generation:
 - 1) => NxtWave Communications NXT2002/NXT2004 based
 - 2) => Generic I2C PLL based tuners
- AirStar ATSC card 3rd generation:
 - 1) => LG Electronics LGDT3302/LGDT3303 based
 - 2) Multimedia support => Customise analog and hybrid tuner modules to build => Simple tuner support

1.1.7.6 TechnoTrend/Hauppauge DEC USB Driver

Driver Status

Supported:

- DEC2000-t
- DEC2450-t
- DEC3000-s
- Video Streaming
- Audio Streaming

- Section Filters
- Channel Zapping
- Hotplug firmware loader

To Do:

- Tuner status information
- DVB network interface
- Streaming video PC->DEC
- Conax support for 2450-t

Getting the Firmware

To download the firmware, use the following commands:

```
scripts/get_dvb_firmware dec2000t
scripts/get_dvb_firmware dec2540t
scripts/get_dvb_firmware dec3000s
```

Hotplug Firmware Loading

Since 2.6 kernels, the firmware is loaded at the point that the driver module is loaded.

Copy the three files downloaded above into the `/usr/lib/hotplug/firmware` or `/lib/firmware` directory (depending on configuration of firmware hotplug).

1.1.7.7 Zoran 364xx based USB webcam module

site: <http://royale.zerezo.com/zr364xx/>

mail: royale@zerezo.com

Introduction

This brings support under Linux for the Aiptek PocketDV 3300 and similar devices in webcam mode. If you just want to get on your PC the pictures and movies on the camera, you should use the `usb-storage` module instead.

The driver works with several other cameras in webcam mode (see the list below).

Possible chipsets are : ZR36430 (ZR36430BGC) and maybe ZR36431, ZR36440, ZR36442...

You can try the experience changing the vendor/product ID values (look at the source code).

You can get these values by looking at `/var/log/messages` when you plug your camera, or by typing : `cat /sys/kernel/debug/usb/devices`.

Install

In order to use this driver, you must compile it with your kernel, with the following config options:

```
./scripts/config -e USB
./scripts/config -m MEDIA_SUPPORT
./scripts/config -e MEDIA_USB_SUPPORT
./scripts/config -e MEDIA_CAMERA_SUPPORT
./scripts/config -m USB_ZR364XX
```

Usage

`modprobe zr364xx debug=X mode=Y`

- `debug` : set to 1 to enable verbose debug messages
- `mode` : 0 = 320x240, 1 = 160x120, 2 = 640x480

You can then use the camera with V4L2 compatible applications, for example Ekiga.

To capture a single image, try this: `dd if=/dev/video0 of=test.jpg bs=1M count=1`

links

<http://mxhaard.free.fr/> (support for many others cams including some Aiptek PocketDV) <http://www.harmwal.nl/pccam880/> (this project also supports cameras based on this chipset)

Supported devices

Vendor	Product	Distributor	Model
0x08ca	0x0109	Aiptek	PocketDV 3300
0x08ca	0x0109	Maxell	Maxcam PRO DV3
0x041e	0x4024	Creative	PC-CAM 880
0x0d64	0x0108	Aiptek	Fidelity 3200
0x0d64	0x0108	Praktica	DCZ 1.3 S
0x0d64	0x0108	Genius	Digital Camera (?)
0x0d64	0x0108	DXG Technology	Fashion Cam
0x0546	0x3187	Polaroid	iON 230
0x0d64	0x3108	Praktica	Exakta DC 2200
0x0d64	0x3108	Genius	G-Shot D211
0x0595	0x4343	Concord	Eye-Q Duo 1300
0x0595	0x4343	Concord	Eye-Q Duo 2000
0x0595	0x4343	Fujifilm	EX-10
0x0595	0x4343	Ricoh	RDC-6000
0x0595	0x4343	Digitrex	DSC 1300
0x0595	0x4343	Firstline	FDC 2000
0x0bb0	0x500d	Concord	EyeQ Go Wireless

Continued on next page

Table 20 – continued from previous page

Vendor	Product	Distributor	Model
0x0feb	0x2004	CRS Electronic	3.3 Digital Camera
0x0feb	0x2004	Packard Bell	DSC-300
0x055f	0xb500	Mustek	MDC 3000
0x08ca	0x2062	Aiptek	PocketDV 5700
0x052b	0x1a18	Chiphead	Megapix V12
0x04c8	0x0729	Konica	Revio 2
0x04f2	0xa208	Creative	PC-CAM 850
0x0784	0x0040	Traveler	Slimline X5
0x06d6	0x0034	Trust	Powerc@m 750
0x0a17	0x0062	Pentax	Optio 50L
0x06d6	0x003b	Trust	Powerc@m 970Z
0x0a17	0x004e	Pentax	Optio 50
0x041e	0x405d	Creative	DiVi CAM 516
0x08ca	0x2102	Aiptek	DV T300
0x06d6	0x003d	Trust	Powerc@m 910Z

1.1.8 CEC driver-specific documentation

1.1.8.1 Pulse-Eight CEC Adapter driver

The pulse8-cec driver implements the following module option:

persistent_config

By default this is off, but when set to 1 the driver will store the current settings to the device's internal eeprom and restore it the next time the device is connected to the USB port.

Copyright © 1999-2020 : LinuxTV Developers

This documentation is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

For more details see the file COPYING in the source distribution of Linux.

MEDIA SUBSYSTEM KERNEL INTERNAL API

This section contains usage information about media subsystem and its supported drivers.

Please see:

Documentation/admin-guide/media/index.rst

- for usage information about media subsystem and supported drivers;

Documentation/userspace-api/media/index.rst

- for the userspace APIs used on media devices.

2.1 Media Subsystem Profile

2.1.1 Overview

The media subsystem covers support for a variety of devices: stream capture, analog and digital TV streams, cameras, remote controllers, HDMI CEC and media pipeline control.

It covers, mainly, the contents of those directories:

- drivers/media
- drivers/staging/media
- Documentation/admin-guide/media
- Documentation/driver-api/media
- Documentation/userspace-api/media
- Documentation/devicetree/bindings/media/¹
- include/media

Both media userspace and Kernel APIs are documented and the documentation must be kept in sync with the API changes. It means that all patches that add new features to the subsystem must also bring changes to the corresponding API files.

Due to the size and wide scope of the media subsystem, media's maintainership model is to have sub-maintainers that have a broad knowledge of a specific aspect of the subsystem. It is

¹ Device tree bindings are maintained by the OPEN FIRMWARE AND FLATTENED DEVICE TREE BINDINGS maintainers (see the MAINTAINERS file). So, changes there must be reviewed by them before being merged via the media subsystem's development tree.

the sub-maintainers' task to review the patches, providing feedback to users if the patches are following the subsystem rules and are properly using the media kernel and userspace APIs.

Patches for the media subsystem must be sent to the media mailing list at linux-media@vger.kernel.org as plain text only e-mail. Emails with HTML will be automatically rejected by the mail server. It could be wise to also copy the sub-maintainer(s).

Media's workflow is heavily based on Patchwork, meaning that, once a patch is submitted, the e-mail will first be accepted by the mailing list server, and, after a while, it should appear at:

- <https://patchwork.linuxtv.org/project/linux-media/list/>

If it doesn't automatically appear there after a few minutes, then probably something went wrong on your submission. Please check if the email is in plain text² only and if your emailer is not mangling whitespaces before complaining or submitting them again.

You can check if the mailing list server accepted your patch, by looking at:

- <https://lore.kernel.org/linux-media/>

2.1.1.1 Media maintainers

At the media subsystem, we have a group of senior developers that are responsible for doing the code reviews at the drivers (also known as sub-maintainers), and another senior developer responsible for the subsystem as a whole. For core changes, whenever possible, multiple media maintainers do the review.

The media maintainers that work on specific areas of the subsystem are:

- **Remote Controllers (infrared):** Sean Young <sean@mess.org>
- **HDMI CEC:** Hans Verkuil <hverkuil@xs4all.nl>
- **Media controller drivers:** Laurent Pinchart <laurent.pinchart@ideasonboard.com>
- **ISP, v4l2-async, v4l2-fwnode, v4l2-flash-led-class and Sensor drivers:** Sakari Ailus <sakari.ailus@linux.intel.com>
- **V4L2 drivers and core V4L2 frameworks:** Hans Verkuil <hverkuil@xs4all.nl>

The subsystem maintainer is: Mauro Carvalho Chehab <mchehab@kernel.org>

Media maintainers may delegate a patch to other media maintainers as needed. On such case, checkpatch's `delegate` field indicates who's currently responsible for reviewing a patch.

2.1.2 Submit Checklist Addendum

Patches that change the Open Firmware/Device Tree bindings must be reviewed by the Device Tree maintainers. So, DT maintainers should be Cc:ed when those are submitted via device-tree@vger.kernel.org mailing list.

There is a set of compliance tools at <https://git.linuxtv.org/v4l-utils.git/> that should be used in order to check if the drivers are properly implementing the media APIs:

² If your email contains HTML, the mailing list server will simply drop it, without any further notice.

Type	Tool
V4L2 drivers ³	v4l2-compliance
V4L2 virtual drivers	contrib/test/test-media
CEC drivers	cec-compliance

Other compliance tools are under development to check other parts of the subsystem.

Those tests need to pass before the patches go upstream.

Also, please notice that we build the Kernel with:

```
make CF=-D__CHECK_ENDIAN__ CONFIG_DEBUG_SECTION_MISMATCH=y C=1 W=1 CHECK=check_script
```

Where the check script is:

```
#!/bin/bash
/devel/smatch/smatch -p=kernel $@ >&2
/devel/sparse/sparse $@ >&2
```

Be sure to not introduce new warnings on your patches without a very good reason.

2.1.2.1 Style Cleanup Patches

Style cleanups are welcome when they come together with other changes at the files where the style changes will affect.

We may accept pure standalone style cleanups, but they should ideally be one patch for the whole subsystem (if the cleanup is low volume), or at least be grouped per directory. So, for example, if you're doing a big cleanup change set at drivers under drivers/media, please send a single patch for all drivers under drivers/media/pci, another one for drivers/media/usb and so on.

2.1.2.2 Coding Style Addendum

Media development uses `checkpatch.pl` on strict mode to verify the code style, e.g.:

```
$ ./scripts/checkpatch.pl --strict --max-line-length=80
```

In principle, patches should follow the coding style rules, but exceptions are allowed if there are good reasons. On such case, maintainers and reviewers may question about the rationale for not addressing the `checkpatch.pl`.

Please notice that the goal here is to improve code readability. On a few cases, `checkpatch.pl` may actually point to something that would look worse. So, you should use good sense.

Note that addressing one `checkpatch.pl` issue (of any kind) alone may lead to having longer lines than 80 characters per line. While this is not strictly prohibited, efforts should be made towards staying within 80 characters per line. This could include using re-factoring code that leads to less indentation, shorter variable or function names and last but not least, simply wrapping the lines.

In particular, we accept lines with more than 80 columns:

³ The v4l2-compliance also covers the media controller usage inside V4L2 drivers.

- on strings, as they shouldn't be broken due to line length limits;
- when a function or variable name need to have a big identifier name, which keeps hard to honor the 80 columns limit;
- on arithmetic expressions, when breaking lines makes them harder to read;
- when they avoid a line to end with an open parenthesis or an open bracket.

2.1.3 Key Cycle Dates

New submissions can be sent at any time, but if they intend to hit the next merge window they should be sent before -rc5, and ideally stabilized in the linux-media branch by -rc6.

2.1.4 Review Cadence

Provided that your patch is at <https://patchwork.linuxtv.org>, it should be sooner or later handled, so you don't need to re-submit a patch.

Except for bug fixes, we don't usually add new patches to the development tree between -rc6 and the next -rc1.

Please notice that the media subsystem is a high traffic one, so it could take a while for us to be able to review your patches. Feel free to ping if you don't get a feedback in a couple of weeks or to ask other developers to publicly add Reviewed-by and, more importantly, Tested-by: tags.

Please note that we expect a detailed description for Tested-by:, identifying what boards were used at the test and what it was tested.

2.2 Video4Linux devices

2.2.1 Introduction

The V4L2 drivers tend to be very complex due to the complexity of the hardware: most devices have multiple ICs, export multiple device nodes in /dev, and create also non-V4L2 devices such as DVB, ALSA, FB, I2C and input (IR) devices.

Especially the fact that V4L2 drivers have to setup supporting ICs to do audio/video muxing/encoding/decoding makes it more complex than most. Usually these ICs are connected to the main bridge driver through one or more I2C buses, but other buses can also be used. Such devices are called 'sub-devices'.

For a long time the framework was limited to the video_device struct for creating V4L device nodes and video_buf for handling the video buffers (note that this document does not discuss the video_buf framework).

This meant that all drivers had to do the setup of device instances and connecting to sub-devices themselves. Some of this is quite complicated to do right and many drivers never did do it correctly.

There is also a lot of common code that could never be refactored due to the lack of a framework.

So this framework sets up the basic building blocks that all drivers need and this same framework should make it much easier to refactor common code into utility functions shared by all drivers.

A good example to look at as a reference is the `v4l2-pci-skeleton.c` source that is available in `samples/v4l/`. It is a skeleton driver for a PCI capture card, and demonstrates how to use the V4L2 driver framework. It can be used as a template for real PCI video capture driver.

2.2.2 Structure of a V4L driver

All drivers have the following structure:

- 1) A struct for each device instance containing the device state.
- 2) A way of initializing and commanding sub-devices (if any).
- 3) Creating V4L2 device nodes (`/dev/videoX`, `/dev/vbiX` and `/dev/radioX`) and keeping track of device-node specific data.
- 4) Filehandle-specific structs containing per-filehandle data;
- 5) video buffer handling.

This is a rough schematic of how it all relates:

```

device instances
 |
+-sub-device instances
 |
 \-V4L2 device nodes
     |
     \-filehandle instances
  
```

2.2.3 Structure of the V4L2 framework

The framework closely resembles the driver structure: it has a `v4l2_device` struct for the device instance data, a `v4l2_subdev` struct to refer to sub-device instances, the `video_device` struct stores V4L2 device node data and the `v4l2_fh` struct keeps track of filehandle instances.

The V4L2 framework also optionally integrates with the media framework. If a driver sets the *struct* `v4l2_device` `mdev` field, sub-devices and video nodes will automatically appear in the media framework as entities.

2.2.4 Video device' s internal representation

The actual device nodes in the `/dev` directory are created using the *video_device* struct (`v4l2-dev.h`). This struct can either be allocated dynamically or embedded in a larger struct.

To allocate it dynamically use *video_device_alloc()*:

```

struct video_device *vdev = video_device_alloc();

if (vdev == NULL)
    return -ENOMEM;
  
```

```
vdev->release = video_device_release;
```

If you embed it in a larger struct, then you must set the `release()` callback to your own function:

```
struct video_device *vdev = &my_vdev->vdev;

vdev->release = my_vdev_release;
```

The `release()` callback must be set and it is called when the last user of the video device exits. The default `video_device_release()` callback currently just calls `kfree` to free the allocated memory.

There is also a `video_device_release_empty()` function that does nothing (is empty) and should be used if the struct is embedded and there is nothing to do when it is released.

You should also set these fields of `video_device`:

- `video_device->v4l2_dev`: must be set to the `v4l2_device` parent device.
- `video_device->name`: set to something descriptive and unique.
- `video_device->vfl_dir`: set this to `VFL_DIR_RX` for capture devices (`VFL_DIR_RX` has value 0, so this is normally already the default), set to `VFL_DIR_TX` for output devices and `VFL_DIR_M2M` for mem2mem (codec) devices.
- `video_device->fops`: set to the `v4l2_file_operations` struct.
- `video_device->ioctl_ops`: if you use the `v4l2_ioctl_ops` to simplify ioctl maintenance (highly recommended to use this and it might become compulsory in the future!), then set this to your `v4l2_ioctl_ops` struct. The `video_device->vfl_type` and `video_device->vfl_dir` fields are used to disable ops that do not match the type/dir combination. E.g. VBI ops are disabled for non-VBI nodes, and output ops are disabled for a capture device. This makes it possible to provide just one `v4l2_ioctl_ops` struct for both vbi and video nodes.
- `video_device->lock`: leave to NULL if you want to do all the locking in the driver. Otherwise you give it a pointer to a struct `mutex_lock` and before the `video_device->unlocked_ioctl` file operation is called this lock will be taken by the core and released afterwards. See the next section for more details.
- `video_device->queue`: a pointer to the `struct vb2_queue` associated with this device node. If queue is not NULL, and `queue->lock` is not NULL, then `queue->lock` is used for the queuing ioctls (`VIDIOC_REQBUFS`, `CREATE_BUFS`, `QBUF`, `DQBUF`, `QUERYBUF`, `PREPARE_BUF`, `STREAMON` and `STREAMOFF`) instead of the lock above. That way the `vb2` queuing framework does not have to wait for other ioctls. This queue pointer is also used by the `vb2` helper functions to check for queuing ownership (i.e. is the filehandle calling it allowed to do the operation).
- `video_device->prio`: keeps track of the priorities. Used to implement `VIDIOC_G_PRIORITY` and `VIDIOC_S_PRIORITY`. If left to NULL, then it will use the `struct v4l2_prio_state` in `v4l2_device`. If you want to have a separate priority state per (group of) device node(s), then you can point it to your own struct `v4l2_prio_state`.
- `video_device->dev_parent`: you only set this if `v4l2_device` was registered with NULL as the parent device struct. This only happens in cases where one hardware device has multiple PCI devices that all share the same `v4l2_device` core.

The cx88 driver is an example of this: one core `v4l2_device` struct, but it is used by both a raw video PCI device (cx8800) and a MPEG PCI device (cx8802). Since the `v4l2_device` cannot be associated with two PCI devices at the same time it is setup without a parent device. But when the `struct video_device` is initialized you **do** know which parent PCI device to use and so you set `dev_device` to the correct PCI device.

If you use `v4l2_ioctl_ops`, then you should set `video_device->unlocked_ioctl` to `video_ioctl2()` in your `v4l2_file_operations` struct.

In some cases you want to tell the core that a function you had specified in your `v4l2_ioctl_ops` should be ignored. You can mark such ioctls by calling this function before `video_register_device()` is called:

```
v4l2_disable_ioctl (vdev, cmd).
```

This tends to be needed if based on external factors (e.g. which card is being used) you want to turn off certain features in `v4l2_ioctl_ops` without having to make a new struct.

The `v4l2_file_operations` struct is a subset of `file_operations`. The main difference is that the `inode` argument is omitted since it is never used.

If integration with the media framework is needed, you must initialize the `media_entity` struct embedded in the `video_device` struct (`entity` field) by calling `media_entity_pads_init()`:

```
struct media_pad *pad = &my_vdev->pad;
int err;

err = media_entity_pads_init(&vdev->entity, 1, pad);
```

The `pads` array must have been previously initialized. There is no need to manually set the `struct media_entity` type and name fields.

A reference to the entity will be automatically acquired/released when the video device is opened/closed.

2.2.4.1 ioctls and locking

The V4L core provides optional locking services. The main service is the `lock` field in `struct video_device`, which is a pointer to a mutex. If you set this pointer, then that will be used by `unlocked_ioctl` to serialize all ioctls.

If you are using the `videobuf2 framework`, then there is a second lock that you can set: `video_device->queue->lock`. If set, then this lock will be used instead of `video_device->lock` to serialize all queuing ioctls (see the previous section for the full list of those ioctls).

The advantage of using a different lock for the queuing ioctls is that for some drivers (particularly USB drivers) certain commands such as setting controls can take a long time, so you want to use a separate lock for the buffer queuing ioctls. That way your `VIDIOC_DQBUF` doesn't stall because the driver is busy changing the e.g. exposure of the webcam.

Of course, you can always do all the locking yourself by leaving both lock pointers at `NULL`.

If you use the old `videobuf framework` then you must pass the `video_device->lock` to the `videobuf` queue initialize function: if `videobuf` has to wait for a frame to arrive, then it will temporarily unlock the lock and relock it afterwards. If your driver also waits in the code, then you should do the same to allow other processes to access the device node while the first process is waiting for something.

In the case of *videobuf2* you will need to implement the `wait_prepare()` and `wait_finish()` callbacks to unlock/lock if applicable. If you use the `queue->lock` pointer, then you can use the helper functions `vb2_ops_wait_prepare()` and `vb2_ops_wait_finish()`.

The implementation of a hotplug disconnect should also take the lock from *video_device* before calling `v4l2_device_disconnect`. If you are also using `video_device->queue->lock`, then you have to first lock `video_device->queue->lock` followed by `video_device->lock`. That way you can be sure no ioctl is running when you call `v4l2_device_disconnect()`.

2.2.4.2 Video device registration

Next you register the video device with `video_register_device()`. This will create the character device for you.

```
err = video_register_device(vdev, VFL_TYPE_VIDEO, -1);
if (err) {
    video_device_release(vdev); /* or kfree(my_vdev); */
    return err;
}
```

If the *v4l2_device* parent device has a not NULL `mdev` field, the video device entity will be automatically registered with the media device.

Which device is registered depends on the type argument. The following types exist:

<i>vfl_devnode_type</i>	Device name	Usage
VFL_TYPE_VIDEO	/dev/videoX	for video input/output devices
VFL_TYPE_VBI	/dev/vbiX	for vertical blank data (i.e. closed captions, teletext)
VFL_TYPE_RADIO	/dev/radioX	for radio tuners
VFL_TYPE_SUBDEV	/dev/v4l-subdevX	for V4L2 subdevices
VFL_TYPE_SDR	/dev/swradioX	for Software Defined Radio (SDR) tuners
VFL_TYPE_TOUCH	/dev/v4l-touchX	for touch sensors

The last argument gives you a certain amount of control over the device node number used (i.e. the X in `videoX`). Normally you will pass -1 to let the v4l2 framework pick the first free number. But sometimes users want to select a specific node number. It is common that drivers allow the user to select a specific device node number through a driver module option. That number is then passed to this function and `video_register_device` will attempt to select that device node number. If that number was already in use, then the next free device node number will be selected and it will send a warning to the kernel log.

Another use-case is if a driver creates many devices. In that case it can be useful to place different video devices in separate ranges. For example, video capture devices start at 0, video output devices start at 16. So you can use the last argument to specify a minimum device node number and the v4l2 framework will try to pick the first free number that is equal or higher to what you passed. If that fails, then it will just pick the first free number.

Since in this case you do not care about a warning about not being able to select the specified device node number, you can call the function `video_register_device_no_warn()` instead.

Whenever a device node is created some attributes are also created for you. If you look in `/sys/class/video4linux` you see the devices. Go into e.g. `video0` and you will see 'name',

'dev_debug' and 'index' attributes. The 'name' attribute is the 'name' field of the video_device struct. The 'dev_debug' attribute can be used to enable core debugging. See the next section for more detailed information on this.

The 'index' attribute is the index of the device node: for each call to `video_register_device()` the index is just increased by 1. The first video device node you register always starts with index 0.

Users can setup udev rules that utilize the index attribute to make fancy device names (e.g. 'mpegX' for MPEG video capture device nodes).

After the device was successfully registered, then you can use these fields:

- `video_device->vfl_type`: the device type passed to `video_register_device()`.
- `video_device->minor`: the assigned device minor number.
- `video_device->num`: the device node number (i.e. the X in videoX).
- `video_device->index`: the device index number.

If the registration failed, then you need to call `video_device_release()` to free the allocated `video_device` struct, or free your own struct if the `video_device` was embedded in it. The `vdev->release()` callback will never be called if the registration failed, nor should you ever attempt to unregister the device if the registration failed.

2.2.4.3 video device debugging

The 'dev_debug' attribute that is created for each video, vbi, radio or swradio device in `/sys/class/video4linux/<devX>/` allows you to enable logging of file operations.

It is a bitmask and the following bits can be set:

Mask	Description
0x01	Log the ioctl name and error code. VIDIOC_(D)QBUF ioctls are only logged if bit 0x08 is also set.
0x02	Log the ioctl name arguments and error code. VIDIOC_(D)QBUF ioctls are only logged if bit 0x08 is also set.
0x04	Log the file operations open, release, read, write, mmap and get_unmapped_area. The read and write operations are only logged if bit 0x08 is also set.
0x08	Log the read and write file operations and the VIDIOC_QBUF and VIDIOC_DQBUF ioctls.
0x10	Log the poll file operation.
0x20	Log error and messages in the control operations.

2.2.4.4 Video device cleanup

When the video device nodes have to be removed, either during the unload of the driver or because the USB device was disconnected, then you should unregister them with:

```
video_unregister_device() (vdev);
```

This will remove the device nodes from sysfs (causing udev to remove them from /dev).

After *video_unregister_device()* returns no new opens can be done. However, in the case of USB devices some application might still have one of these device nodes open. So after the unregister all file operations (except release, of course) will return an error as well.

When the last user of the video device node exits, then the *vdev->release()* callback is called and you can do the final cleanup there.

Don't forget to cleanup the media entity associated with the video device if it has been initialized:

```
media_entity_cleanup (&vdev->entity);
```

This can be done from the release callback.

2.2.4.5 helper functions

There are a few useful helper functions:

- file and *video_device* private data

You can set/get driver private data in the *video_device* struct using:

```
video_get_drvdata (vdev);
```

```
video_set_drvdata (vdev);
```

Note that you can safely call *video_set_drvdata()* before calling *video_register_device()*.

And this function:

```
video_devdata (struct file *file);
```

returns the *video_device* belonging to the file struct.

The *video_devdata()* function combines *video_get_drvdata()* with *video_devdata()*:

```
video_drvdata (struct file *file);
```

You can go from a *video_device* struct to the *v4l2_device* struct using:

```
struct v4l2_device *v4l2_dev = vdev->v4l2_dev;
```

- Device node name

The *video_device* node kernel name can be retrieved using:

```
video_device_node_name (vdev);
```

The name is used as a hint by userspace tools such as udev. The function should be used where possible instead of accessing the *video_device::num* and *video_device::minor* fields.

2.2.4.6 video_device functions and data structures

enum **vfl_devnode_type**
 type of V4L2 device node

Constants

VFL_TYPE_VIDEO for video input/output devices

VFL_TYPE_VBI for vertical blank data (i.e. closed captions, teletext)

VFL_TYPE_RADIO for radio tuners

VFL_TYPE_SUBDEV for V4L2 subdevices

VFL_TYPE_SDR for Software Defined Radio tuners

VFL_TYPE_TOUCH for touch sensors

VFL_TYPE_MAX number of VFL types, must always be last in the enum

enum **vfl_devnode_direction**

Identifies if a *struct video_device* corresponds to a receiver, a transmitter or a mem-to-mem device.

Constants

VFL_DIR_RX device is a receiver.

VFL_DIR_TX device is a transmitter.

VFL_DIR_M2M device is a memory to memory device.

Note

Ignored if *enum vfl_devnode_type* is **VFL_TYPE_SUBDEV**.

enum **v4l2_video_device_flags**

Flags used by *struct video_device*

Constants

V4L2_FL_REGISTERED

indicates that a *struct video_device* is registered. Drivers can clear this flag if they want to block all future device access. It is cleared by `video_unregister_device`.

V4L2_FL_USES_V4L2_FH

indicates that `file->private_data` points to *struct v4l2_fh*. This flag is set by the core when `v4l2_fh_init()` is called. All new drivers should use it.

V4L2_FL_QUIRK_INVERTED_CROP

some old M2M drivers use `g/s_crop/cropcap` incorrectly: `crop` and `compose` are swapped. If this flag is set, then the selection targets are swapped in the `g/s_crop/cropcap` functions in `v4l2-ioctl.c`. This allows those drivers to correctly implement the selection API, but the old `crop` API will still work as expected in order to preserve backwards compatibility. Never set this flag for new drivers.

V4L2_FL_SUBDEV_RO_DEVNODE

indicates that the video device node is registered in read-only mode. The flag only applies to device nodes registered for sub-devices, it is set by the core when the sub-devices device nodes are registered with `v4l2_device_register_ro_subdev_nodes()` and used by the sub-device ioctl handler to restrict access to some ioctl calls.

struct **v4l2_prio_state**
stores the priority states

Definition

```
struct v4l2_prio_state {
    atomic_t prios[4];
};
```

Members

prios array with elements to store the array priorities

Description

Note: The size of **prios** array matches the number of priority types defined by enum `v4l2_priority`.

void **v4l2_prio_init**(struct *v4l2_prio_state* **global*)
initializes a *struct v4l2_prio_state*

Parameters

struct v4l2_prio_state **global* pointer to *struct v4l2_prio_state*

int **v4l2_prio_change**(struct *v4l2_prio_state* **global*, enum *v4l2_priority* **local*, enum *v4l2_priority* *new*)
changes the v4l2 file handler priority

Parameters

struct v4l2_prio_state **global* pointer to the *struct v4l2_prio_state* of the device node.

enum v4l2_priority **local* pointer to the desired priority, as defined by enum `v4l2_priority`

enum v4l2_priority *new* Priority type requested, as defined by enum `v4l2_priority`.

Description

Note: This function should be used only by the V4L2 core.

void **v4l2_prio_open**(struct *v4l2_prio_state* **global*, enum *v4l2_priority* **local*)
Implements the priority logic for a file handler open

Parameters

struct v4l2_prio_state **global* pointer to the *struct v4l2_prio_state* of the device node.

enum v4l2_priority *local* pointer to the desired priority, as defined by enum v4l2_priority

Description

Note: This function should be used only by the V4L2 core.

void **v4l2_prio_close**(struct *v4l2_prio_state* **global*, enum v4l2_priority *local*)
Implements the priority logic for a file handler close

Parameters

struct v4l2_prio_state **global* pointer to the *struct v4l2_prio_state* of the device node.

enum v4l2_priority *local* priority to be released, as defined by enum v4l2_priority

Description

Note: This function should be used only by the V4L2 core.

enum v4l2_priority **v4l2_prio_max**(struct *v4l2_prio_state* **global*)
Return the maximum priority, as stored at the **global** array.

Parameters

struct v4l2_prio_state **global* pointer to the *struct v4l2_prio_state* of the device node.

Description

Note: This function should be used only by the V4L2 core.

int **v4l2_prio_check**(struct *v4l2_prio_state* **global*, enum v4l2_priority *local*)
Implements the priority logic for a file handler close

Parameters

struct v4l2_prio_state **global* pointer to the *struct v4l2_prio_state* of the device node.

enum v4l2_priority *local* desired priority, as defined by enum v4l2_priority *local*

Description

Note: This function should be used only by the V4L2 core.

struct **v4l2_file_operations**
fs operations used by a V4L2 device

Definition

```
struct v4l2_file_operations {
    struct module *owner;
    ssize_t (*read) (struct file *, char __user *, size_t, loff_t *);
    ssize_t (*write) (struct file *, const char __user *, size_t, loff_t *);
    __poll_t (*poll) (struct file *, struct poll_table_struct *);
    long (*unlocked_ioctl) (struct file *, unsigned int, unsigned long);
#ifdef CONFIG_COMPAT;
    long (*compat_ioctl32) (struct file *, unsigned int, unsigned long);
#endif;
    unsigned long (*get_unmapped_area) (struct file *, unsigned long, unsigned long,
    ↪ unsigned long, unsigned long);
    int (*mmap) (struct file *, struct vm_area_struct *);
    int (*open) (struct file *);
    int (*release) (struct file *);
};
```

Members

owner pointer to struct module

read operations needed to implement the read() syscall

write operations needed to implement the write() syscall

poll operations needed to implement the poll() syscall

unlocked_ioctl operations needed to implement the ioctl() syscall

compat_ioctl32 operations needed to implement the ioctl() syscall for the special case where the Kernel uses 64 bits instructions, but the userspace uses 32 bits.

get_unmapped_area called by the mmap() syscall, used when %!CONFIG_MMU

mmap operations needed to implement the mmap() syscall

open operations needed to implement the open() syscall

release operations needed to implement the release() syscall

Description

Note: Those operations are used to implemente the fs struct file_operations at the V4L2 drivers. The V4L2 core overrides the fs ops with some extra logic needed by the subsystem.

struct video_device

Structure used to create and manage the V4L2 device nodes.

Definition

```
struct video_device {
#ifdef CONFIG_MEDIA_CONTROLLER;
    struct media_entity entity;
    struct media_intf_devnode *intf_devnode;
    struct media_pipeline pipe;
#endif;
    const struct v4l2_file_operations *fops;
    u32 device_caps;
    struct device dev;
    struct cdev *cdev;
};
```



```

struct v4l2_device *v4l2_dev;
struct device *dev_parent;
struct v4l2_ctrl_handler *ctrl_handler;
struct vb2_queue *queue;
struct v4l2_prio_state *prio;
char name[32];
enum vfl_devnode_type vfl_type;
enum vfl_devnode_direction vfl_dir;
int minor;
u16 num;
unsigned long flags;
int index;
spinlock_t fh_lock;
struct list_head fh_list;
int dev_debug;
v4l2_std_id tvnorms;
void (*release)(struct video_device *vdev);
const struct v4l2_ioctl_ops *ioctl_ops;
unsigned long valid_ioctls[BITS_TO_LONGS(BASE_VIDIIOC_PRIVATE)];
struct mutex *lock;
};

```

Members

entity *struct media_entity*

intf_devnode pointer to *struct media_intf_devnode*

pipe *struct media_pipeline*

fops pointer to *struct v4l2_file_operations* for the video device

device_caps device capabilities as used in *v4l2_capabilities*

dev *struct device* for the video device

cdev character device

v4l2_dev pointer to *struct v4l2_device* parent

dev_parent pointer to *struct device* parent

ctrl_handler Control handler associated with this device node. May be NULL.

queue *struct vb2_queue* associated with this device node. May be NULL.

prio pointer to *struct v4l2_prio_state* with device's Priority state. If NULL, then *v4l2_dev->prio* will be used.

name video device name

vfl_type V4L device type, as defined by *enum vfl_devnode_type*

vfl_dir V4L receiver, transmitter or m2m

minor device node 'minor'. It is set to -1 if the registration failed

num number of the video device node

flags video device flags. Use bitops to set/clear/test flags. Contains a set of *enum v4l2_video_device_flags*.

index attribute to differentiate multiple indices on one physical device

fh_lock Lock for all v4l2_fhs

fh_list List of *struct v4l2_fh*

dev_debug Internal device debug flags, not for use by drivers

tvnorms Supported tv norms

release video device release() callback

ioctl_ops pointer to *struct v4l2_ioctl_ops* with ioctl callbacks

valid_ioctls bitmap with the valid ioctls for this device

lock pointer to struct mutex serialization lock

Description

Note: Only set **dev_parent** if that can't be deduced from **v4l2_dev**.

media_entity_to_video_device(__entity)

Returns a *struct video_device* from the *struct media_entity* embedded on it.

Parameters

__entity pointer to *struct media_entity*

to_video_device(cd)

Returns a *struct video_device* from the struct device embedded on it.

Parameters

cd pointer to struct device

int __video_register_device(struct video_device *vdev, enum vfl_devnode_type type, int nr, int warn_if_nr_in_use, struct module *owner)
register video4linux devices

Parameters

struct video_device *vdev *struct video_device* to register

enum vfl_devnode_type type type of device to register, as defined by *enum vfl_devnode_type*

int nr which device node number is desired: (0 == /dev/video0, 1 == /dev/video1, ..., -1 == first free)

int warn_if_nr_in_use warn if the desired device node number was already in use and another number was chosen instead.

struct module *owner module that owns the video device node

Description

The registration code assigns minor numbers and device node numbers based on the requested type and registers the new device node with the kernel.

This function assumes that *struct video_device* was zeroed when it was allocated and does not contain any stale data.

An error is returned if no free minor or device node number could be found, or if the registration of the device node failed.

Returns 0 on success.

Note: This function is meant to be used only inside the V4L2 core. Drivers should use `video_register_device()` or `video_register_device_no_warn()`.

```
int video_register_device(struct video_device *vdev, enum vfl_devnode_type type,
                        int nr)
    register video4linux devices
```

Parameters

struct video_device *vdev `struct video_device` to register

enum vfl_devnode_type type type of device to register, as defined by `enum vfl_devnode_type`

int nr which device node number is desired: (0 == /dev/video0, 1 == /dev/video1, ..., -1 == first free)

Description

Internally, it calls `__video_register_device()`. Please see its documentation for more details.

Note: if `video_register_device` fails, the `release()` callback of `struct video_device` structure is *not* called, so the caller is responsible for freeing any data. Usually that means that you `video_device_release()` should be called on failure.

```
int video_register_device_no_warn(struct video_device *vdev, enum
                                vfl_devnode_type type, int nr)
    register video4linux devices
```

Parameters

struct video_device *vdev `struct video_device` to register

enum vfl_devnode_type type type of device to register, as defined by `enum vfl_devnode_type`

int nr which device node number is desired: (0 == /dev/video0, 1 == /dev/video1, ..., -1 == first free)

Description

This function is identical to `video_register_device()` except that no warning is issued if the desired device node number was already in use.

Internally, it calls `__video_register_device()`. Please see its documentation for more details.

Note: if `video_register_device` fails, the `release()` callback of `struct video_device` structure is *not* called, so the caller is responsible for freeing any data. Usually that means that you `video_device_release()` should be called on failure.

void **video_unregister_device**(struct *video_device* *vdev)
Unregister video devices.

Parameters

struct video_device *vdev *struct video_device* to register

Description

Does nothing if vdev == NULL or if *video_is_registered()* returns false.

struct *video_device* * **video_device_alloc**(void)
helper function to alloc *struct video_device*

Parameters

void no arguments

Description

Returns NULL if -ENOMEM or a *struct video_device* on success.

void **video_device_release**(struct *video_device* *vdev)
helper function to release *struct video_device*

Parameters

struct video_device *vdev pointer to *struct video_device*

Description

Can also be used for video_device->release().

void **video_device_release_empty**(struct *video_device* *vdev)
helper function to implement the video_device->release() callback.

Parameters

struct video_device *vdev pointer to *struct video_device*

Description

This release function does nothing.

It should be used when the video_device is a static global struct.

Note: Having a static video_device is a dubious construction at best.

void **v4l2_disable_ioctl**(struct *video_device* *vdev, unsigned int cmd)
mark that a given command isn't implemented. shouldn't use core locking

Parameters

struct video_device *vdev pointer to *struct video_device*

unsigned int cmd ioctl command

Description

This function allows drivers to provide just one v4l2_ioctl_ops struct, but disable ioctls based on the specific card that is actually found.

Note: This must be called before `video_register_device`. See also the comments for `determine_valid_ioctls()`.

`void *video_get_drvdata(struct video_device *vdev)`
gets private data from *struct video_device*.

Parameters

`struct video_device *vdev` pointer to *struct video_device*

Description

returns a pointer to the private data

`void video_set_drvdata(struct video_device *vdev, void *data)`
sets private data from *struct video_device*.

Parameters

`struct video_device *vdev` pointer to *struct video_device*

`void *data` private data pointer

`struct video_device *video_devdata(struct file *file)`
gets *struct video_device* from struct file.

Parameters

`struct file *file` pointer to struct file

`void *video_drvdata(struct file *file)`
gets private data from *struct video_device* using the struct file.

Parameters

`struct file *file` pointer to struct file

Description

This is function combines both `video_get_drvdata()` and `video_devdata()` as this is used very often.

`const char *video_device_node_name(struct video_device *vdev)`
returns the video device name

Parameters

`struct video_device *vdev` pointer to *struct video_device*

Description

Returns the device name string

`int video_is_registered(struct video_device *vdev)`
returns true if the *struct video_device* is registered.

Parameters

`struct video_device *vdev` pointer to *struct video_device*

Description

2.2.5 V4L2 device instance

Each device instance is represented by a *struct v4l2_device*. Very simple devices can just allocate this struct, but most of the time you would embed this struct inside a larger struct.

You must register the device instance by calling:

```
v4l2_device_register (dev, v4l2_dev).
```

Registration will initialize the *v4l2_device* struct. If the *dev->driver_data* field is NULL, it will be linked to *v4l2_dev* argument.

Drivers that want integration with the media device framework need to set *dev->driver_data* manually to point to the driver-specific device structure that embed the *struct v4l2_device* instance. This is achieved by a *dev_set_drvdata()* call before registering the V4L2 device instance. They must also set the *struct v4l2_device* *mdev* field to point to a properly initialized and registered *media_device* instance.

If *v4l2_dev->name* is empty then it will be set to a value derived from *dev* (driver name followed by the *bus_id*, to be precise). If you set it up before calling *v4l2_device_register()* then it will be untouched. If *dev* is NULL, then you **must** setup *v4l2_dev->name* before calling *v4l2_device_register()*.

You can use *v4l2_device_set_name()* to set the name based on a driver name and a driver-global *atomic_t* instance. This will generate names like *ivtv0*, *ivtv1*, etc. If the name ends with a digit, then it will insert a dash: *cx18-0*, *cx18-1*, etc. This function returns the instance number.

The first *dev* argument is normally the *struct device* pointer of a *pci_dev*, *usb_interface* or *platform_device*. It is rare for *dev* to be NULL, but it happens with ISA devices or when one device creates multiple PCI devices, thus making it impossible to associate *v4l2_dev* with a particular parent.

You can also supply a *notify()* callback that can be called by sub-devices to notify you of events. Whether you need to set this depends on the sub-device. Any notifications a sub-device supports must be defined in a header in *include/media/subdevice.h*.

V4L2 devices are unregistered by calling:

```
v4l2_device_unregister (v4l2_dev).
```

If the *dev->driver_data* field points to *v4l2_dev*, it will be reset to NULL. Unregistering will also automatically unregister all subdevs from the device.

If you have a hotpluggable device (e.g. a USB device), then when a disconnect happens the parent device becomes invalid. Since *v4l2_device* has a pointer to that parent device it has to be cleared as well to mark that the parent is gone. To do this call:

```
v4l2_device_disconnect (v4l2_dev).
```

This does *not* unregister the subdevs, so you still need to call the *v4l2_device_unregister()* function for that. If your driver is not hotpluggable, then there is no need to call *v4l2_device_disconnect()*.

Sometimes you need to iterate over all devices registered by a specific driver. This is usually the case if multiple device drivers use the same hardware. E.g. the *ivtvfb* driver is a framebuffer driver that uses the *ivtv* hardware. The same is true for *alsa* drivers for example.

You can iterate over all registered devices as follows:

```

static int callback(struct device *dev, void *p)
{
    struct v4l2_device *v4l2_dev = dev_get_drvdata(dev);

    /* test if this device was inited */
    if (v4l2_dev == NULL)
        return 0;

    ...
    return 0;
}

int iterate(void *p)
{
    struct device_driver *drv;
    int err;

    /* Find driver 'ivtv' on the PCI bus.
    pci_bus_type is a global. For USB buses use usb_bus_type. */
    drv = driver_find("ivtv", &pci_bus_type);
    /* iterate over all ivtv device instances */
    err = driver_for_each_device(drv, NULL, p, callback);
    put_driver(drv);
    return err;
}

```

Sometimes you need to keep a running counter of the device instance. This is commonly used to map a device instance to an index of a module option array.

The recommended approach is as follows:

```

static atomic_t drv_instance = ATOMIC_INIT(0);

static int drv_probe(struct pci_dev *pdev, const struct pci_device_id *pci_id)
{
    ...
    state->instance = atomic_inc_return(&drv_instance) - 1;
}

```

If you have multiple device nodes then it can be difficult to know when it is safe to unregister `v4l2_device` for hotpluggable devices. For this purpose `v4l2_device` has refcounting support. The refcount is increased whenever `video_register_device()` is called and it is decreased whenever that device node is released. When the refcount reaches zero, then the `v4l2_device release()` callback is called. You can do your final cleanup there.

If other device nodes (e.g. ALSA) are created, then you can increase and decrease the refcount manually as well by calling:

```
v4l2_device_get() (v4l2_dev).
```

or:

```
v4l2_device_put() (v4l2_dev).
```

Since the initial refcount is 1 you also need to call `v4l2_device_put()` in the `disconnect()` callback (for USB devices) or in the `remove()` callback (for e.g. PCI devices), otherwise the refcount will never reach 0.

2.2.5.1 v4l2_device functions and data structures

struct **v4l2_device**

main struct to for V4L2 device drivers

Definition

```
struct v4l2_device {
    struct device *dev;
    struct media_device *mdev;
    struct list_head subdevs;
    spinlock_t lock;
    char name[V4L2_DEVICE_NAME_SIZE];
    void (*notify)(struct v4l2_subdev *sd, unsigned int notification, void *arg);
    struct v4l2_ctrl_handler *ctrl_handler;
    struct v4l2_prio_state prio;
    struct kref ref;
    void (*release)(struct v4l2_device *v4l2_dev);
};
```

Members

dev pointer to struct device.

mdev pointer to *struct media_device*, may be NULL.

subdevs used to keep track of the registered subdevs

lock lock this struct; can be used by the driver as well if this struct is embedded into a larger struct.

name unique device name, by default the driver name + bus ID

notify notify operation called by some sub-devices.

ctrl_handler The control handler. May be NULL.

prio Device's priority state

ref Keep track of the references to this struct.

release Release function that is called when the ref count goes to 0.

Description

Each instance of a V4L2 device should create the `v4l2_device` struct, either stand-alone or embedded in a larger struct.

It allows easy access to sub-devices (see `v4l2-subdev.h`) and provides basic V4L2 device-level support.

Note:

- 1) **dev->driver_data** points to this struct.
 - 2) **dev** might be NULL if there is no parent device
-

void **v4l2_device_get**(struct *v4l2_device* **v4l2_dev*)
gets a V4L2 device reference

Parameters

struct v4l2_device *v4l2_dev pointer to struct *v4l2_device*

Description

This is an ancillary routine meant to increment the usage for the struct *v4l2_device* pointed by **v4l2_dev**.

```
int v4l2_device_put(struct v4l2_device *v4l2_dev)
    puts a V4L2 device reference
```

Parameters

struct v4l2_device *v4l2_dev pointer to struct *v4l2_device*

Description

This is an ancillary routine meant to decrement the usage for the struct *v4l2_device* pointed by **v4l2_dev**.

```
int v4l2_device_register(struct device *dev, struct v4l2_device *v4l2_dev)
    Initialize v4l2_dev and make dev->driver_data point to v4l2_dev.
```

Parameters

struct device *dev pointer to struct device

struct v4l2_device *v4l2_dev pointer to struct *v4l2_device*

Description

Note: **dev** may be NULL in rare cases (ISA devices). In such case the caller must fill in the **v4l2_dev->name** field before calling this function.

```
int v4l2_device_set_name(struct v4l2_device *v4l2_dev, const char *basename,
    atomic_t *instance)
    Optional function to initialize the name field of struct v4l2_device
```

Parameters

struct v4l2_device *v4l2_dev pointer to struct *v4l2_device*

const char *basename base name for the device name

atomic_t *instance pointer to a static `atomic_t` var with the instance usage for the device driver.

Description

v4l2_device_set_name() initializes the name field of struct *v4l2_device* using the driver name and a driver-global `atomic_t` instance.

This function will increment the instance counter and returns the instance value used in the name.

The first time this is called the name field will be set to `foo0` and this function returns 0. If the name ends with a digit (e.g. `cx18`), then the name will be set to `cx18-0` since `cx180` would look really odd.

Example

```
static atomic_t drv_instance = ATOMIC_INIT(0);
```

```
...
```

```
instance = v4l2_device_set_name(&v4l2_dev, "foo", &drv_instance);
```

```
void v4l2_device_disconnect(struct v4l2_device *v4l2_dev)
```

Change V4L2 device state to disconnected.

Parameters

struct v4l2_device *v4l2_dev pointer to *struct v4l2_device*

Description

Should be called when the USB parent disconnects. Since the parent disappears, this ensures that **v4l2_dev** doesn't have an invalid parent pointer.

Note: This function sets **v4l2_dev->dev** to NULL.

```
void v4l2_device_unregister(struct v4l2_device *v4l2_dev)
```

Unregister all sub-devices and any other resources related to **v4l2_dev**.

Parameters

struct v4l2_device *v4l2_dev pointer to *struct v4l2_device*

```
int v4l2_device_register_subdev(struct v4l2_device *v4l2_dev, struct  
v4l2_subdev *sd)
```

Registers a subdev with a v4l2 device.

Parameters

struct v4l2_device *v4l2_dev pointer to *struct v4l2_device*

struct v4l2_subdev *sd pointer to *struct v4l2_subdev*

Description

While registered, the subdev module is marked as in-use.

An error is returned if the module is no longer loaded on any attempts to register it.

```
void v4l2_device_unregister_subdev(struct v4l2_subdev *sd)
```

Unregisters a subdev with a v4l2 device.

Parameters

struct v4l2_subdev *sd pointer to *struct v4l2_subdev*

Description

Note: Can also be called if the subdev wasn't registered. In such case, it will do nothing.

```
int __v4l2_device_register_subdev_nodes(struct v4l2_device *v4l2_dev,  
bool read_only)
```

Registers device nodes for all subdevs of the v4l2 device that are marked with the V4L2_SUBDEV_FL_HAS_DEVNODE flag.

Parameters

struct v4l2_device *v4l2_dev pointer to *struct v4l2_device*

bool read_only subdevices read-only flag. True to register the subdevices device nodes in read-only mode, false to allow full access to the subdevice userspace API.

int **v4l2_device_register_subdev_nodes**(struct *v4l2_device* *v4l2_dev)

Registers subdevices device nodes with unrestricted access to the subdevice userspace operations

Parameters

struct v4l2_device *v4l2_dev pointer to *struct v4l2_device*

Description

Internally calls `__v4l2_device_register_subdev_nodes()`. See its documentation for more details.

int **v4l2_device_register_ro_subdev_nodes**(struct *v4l2_device* *v4l2_dev)

Registers subdevices device nodes in read-only mode

Parameters

struct v4l2_device *v4l2_dev pointer to *struct v4l2_device*

Description

Internally calls `__v4l2_device_register_subdev_nodes()`. See its documentation for more details.

void **v4l2_subdev_notify**(struct *v4l2_subdev* *sd, unsigned int *notification*, void *arg)

Sends a notification to v4l2_device.

Parameters

struct v4l2_subdev *sd pointer to *struct v4l2_subdev*

unsigned int notification type of notification. Please notice that the notification type is driver-specific.

void *arg arguments for the notification. Those are specific to each notification type.

bool **v4l2_device_supports_requests**(struct *v4l2_device* *v4l2_dev)

Test if requests are supported.

Parameters

struct v4l2_device *v4l2_dev pointer to *struct v4l2_device*

v4l2_device_for_each_subdev(sd, v4l2_dev)

Helper macro that iterates over all sub-devices of a given *v4l2_device*.

Parameters

sd pointer that will be filled by the macro with all *struct v4l2_subdev* pointer used as an iterator by the loop.

v4l2_dev *struct v4l2_device* owning the sub-devices to iterate over.

Description

This macro iterates over all sub-devices owned by the **v4l2_dev** device. It acts as a for loop iterator and executes the next statement with the **sd** variable pointing to each sub-device in turn.

`__v4l2_device_call_subdevs_p(v4l2_dev, sd, cond, o, f, args...)`

Calls the specified operation for all subdevs matching the condition.

Parameters

v4l2_dev *struct v4l2_device* owning the sub-devices to iterate over.

sd pointer that will be filled by the macro with all *struct v4l2_subdev* pointer used as an iterator by the loop.

cond condition to be match

o name of the element at *struct v4l2_subdev_ops* that contains **f**. Each element there groups a set of operations functions.

f operation function that will be called if **cond** matches. The operation functions are defined in groups, according to each element at *struct v4l2_subdev_ops*.

args... arguments for **f**.

Description

Ignore any errors.

Note

subdevs cannot be added or deleted while walking the subdevs list.

`__v4l2_device_call_subdevs(v4l2_dev, cond, o, f, args...)`

Calls the specified operation for all subdevs matching the condition.

Parameters

v4l2_dev *struct v4l2_device* owning the sub-devices to iterate over.

cond condition to be match

o name of the element at *struct v4l2_subdev_ops* that contains **f**. Each element there groups a set of operations functions.

f operation function that will be called if **cond** matches. The operation functions are defined in groups, according to each element at *struct v4l2_subdev_ops*.

args... arguments for **f**.

Description

Ignore any errors.

Note

subdevs cannot be added or deleted while walking the subdevs list.

`__v4l2_device_call_subdevs_until_err_p(v4l2_dev, sd, cond, o, f, args...)`

Calls the specified operation for all subdevs matching the condition.

Parameters

v4l2_dev *struct v4l2_device* owning the sub-devices to iterate over.

sd pointer that will be filled by the macro with all *struct v4l2_subdev* sub-devices associated with **v4l2_dev**.

cond condition to be match

o name of the element at *struct v4l2_subdev_ops* that contains **f**. Each element there groups a set of operations functions.

f operation function that will be called if **cond** matches. The operation functions are defined in groups, according to each element at *struct v4l2_subdev_ops*.

args... arguments for **f**.

Return

Description

If the operation returns an error other than 0 or -ENOIOCTLCMD for any subdevice, then abort and return with that error code, zero otherwise.

Note

subdevs cannot be added or deleted while walking the subdevs list.

__v4l2_device_call_subdevs_until_err(*v4l2_dev, cond, o, f, args...*)
Calls the specified operation for all subdevs matching the condition.

Parameters

v4l2_dev *struct v4l2_device* owning the sub-devices to iterate over.

cond condition to be match

o name of the element at *struct v4l2_subdev_ops* that contains **f**. Each element there groups a set of operations functions.

f operation function that will be called if **cond** matches. The operation functions are defined in groups, according to each element at *struct v4l2_subdev_ops*.

args... arguments for **f**.

Return

Description

If the operation returns an error other than 0 or -ENOIOCTLCMD for any subdevice, then abort and return with that error code, zero otherwise.

Note

subdevs cannot be added or deleted while walking the subdevs list.

v4l2_device_call_all(*v4l2_dev, grp_id, o, f, args...*)
Calls the specified operation for all subdevs matching the *v4l2_subdev.grp_id*, as assigned by the bridge driver.

Parameters

v4l2_dev *struct v4l2_device* owning the sub-devices to iterate over.

grp_id *struct v4l2_subdev->grp_id* group ID to match. Use 0 to match them all.

o name of the element at *struct v4l2_subdev_ops* that contains **f**. Each element there groups a set of operations functions.

f operation function that will be called if **cond** matches. The operation functions are defined in groups, according to each element at *struct v4l2_subdev_ops*.

args... arguments for **f**.

Description

Ignore any errors.

Note

subdevs cannot be added or deleted while walking the subdevs list.

v4l2_device_call_until_err(*v4l2_dev, grp_id, o, f, args...*)

Calls the specified operation for all subdevs matching the *v4l2_subdev.grp_id*, as assigned by the bridge driver, until an error occurs.

Parameters

v4l2_dev *struct v4l2_device* owning the sub-devices to iterate over.

grp_id *struct v4l2_subdev->grp_id* group ID to match. Use 0 to match them all.

o name of the element at *struct v4l2_subdev_ops* that contains **f**. Each element there groups a set of operations functions.

f operation function that will be called if **cond** matches. The operation functions are defined in groups, according to each element at *struct v4l2_subdev_ops*.

args... arguments for **f**.

Return

Description

If the operation returns an error other than 0 or -ENOIOCTLCMD for any subdevice, then abort and return with that error code, zero otherwise.

Note

subdevs cannot be added or deleted while walking the subdevs list.

v4l2_device_mask_call_all(*v4l2_dev, grpmsk, o, f, args...*)

Calls the specified operation for all subdevices where a group ID matches a specified bitmask.

Parameters

v4l2_dev *struct v4l2_device* owning the sub-devices to iterate over.

grpmsk bitmask to be checked against *struct v4l2_subdev->grp_id* group ID to be matched. Use 0 to match them all.

o name of the element at *struct v4l2_subdev_ops* that contains **f**. Each element there groups a set of operations functions.

f operation function that will be called if **cond** matches. The operation functions are defined in groups, according to each element at *struct v4l2_subdev_ops*.

args... arguments for **f**.

Description

Ignore any errors.

Note

subdevs cannot be added or deleted while walking the subdevs list.

v4l2_device_mask_call_until_err(*v4l2_dev*, *grpmsk*, *o*, *f*, *args...*)

Calls the specified operation for all subdevices where a group ID matches a specified bit-mask.

Parameters

v4l2_dev *struct v4l2_device* owning the sub-devices to iterate over.

grpmsk bitmask to be checked against *struct v4l2_subdev->grp_id* group ID to be matched. Use 0 to match them all.

o name of the element at *struct v4l2_subdev_ops* that contains **f**. Each element there groups a set of operations functions.

f operation function that will be called if **cond** matches. The operation functions are defined in groups, according to each element at *struct v4l2_subdev_ops*.

args... arguments for **f**.

Return

Description

If the operation returns an error other than 0 or -ENOIOCTLCMD for any subdevice, then abort and return with that error code, zero otherwise.

Note

subdevs cannot be added or deleted while walking the subdevs list.

v4l2_device_has_op(*v4l2_dev*, *grp_id*, *o*, *f*)

checks if any subdev with matching *grp_id* has a given *ops*.

Parameters

v4l2_dev *struct v4l2_device* owning the sub-devices to iterate over.

grp_id *struct v4l2_subdev->grp_id* group ID to match. Use 0 to match them all.

o name of the element at *struct v4l2_subdev_ops* that contains **f**. Each element there groups a set of operations functions.

f operation function that will be called if **cond** matches. The operation functions are defined in groups, according to each element at *struct v4l2_subdev_ops*.

v4l2_device_mask_has_op(*v4l2_dev*, *grpmsk*, *o*, *f*)

checks if any subdev with matching group mask has a given *ops*.

Parameters

v4l2_dev *struct v4l2_device* owning the sub-devices to iterate over.

grpmsk bitmask to be checked against *struct v4l2_subdev->grp_id* group ID to be matched. Use 0 to match them all.

o name of the element at *struct v4l2_subdev_ops* that contains **f**. Each element there groups a set of operations functions.

f operation function that will be called if **cond** matches. The operation functions are defined in groups, according to each element at *struct v4l2_subdev_ops*.

2.2.6 V4L2 File handlers

`struct v4l2_fh` provides a way to easily keep file handle specific data that is used by the V4L2 framework.

Attention: New drivers must use `struct v4l2_fh` since it is also used to implement priority handling (`ioctl VIDIOC_G_PRIORITY, VIDIOC_S_PRIORITY`).

The users of `v4l2_fh` (in the V4L2 framework, not the driver) know whether a driver uses `v4l2_fh` as its `file->private_data` pointer by testing the `V4L2_FL_USES_V4L2_FH` bit in `video_device->flags`. This bit is set whenever `v4l2_fh_init()` is called.

`struct v4l2_fh` is allocated as a part of the driver's own file handle structure and `file->private_data` is set to it in the driver's `open()` function by the driver.

In many cases the `struct v4l2_fh` will be embedded in a larger structure. In that case you should call:

- 1) `v4l2_fh_init()` and `v4l2_fh_add()` in `open()`
- 2) `v4l2_fh_del()` and `v4l2_fh_exit()` in `release()`

Drivers can extract their own file handle structure by using the `container_of` macro.

Example:

```
struct my_fh {
    int blah;
    struct v4l2_fh fh;
};

...

int my_open(struct file *file)
{
    struct my_fh *my_fh;
    struct video_device *vfd;
    int ret;

    ...

    my_fh = kzalloc(sizeof(*my_fh), GFP_KERNEL);

    ...

    v4l2_fh_init(&my_fh->fh, vfd);

    ...

    file->private_data = &my_fh->fh;
    v4l2_fh_add(&my_fh->fh);
    return 0;
}

int my_release(struct file *file)
{
```



```

struct v4l2_fh *fh = file->private_data;
struct my_fh *my_fh = container_of(fh, struct my_fh, fh);

...
v4l2_fh_del(&my_fh->fh);
v4l2_fh_exit(&my_fh->fh);
kfree(my_fh);
return 0;
}

```

Below is a short description of the *v4l2_fh* functions used:

v4l2_fh_init (*fh*, *vdev*)

- Initialise the file handle. This **MUST** be performed in the driver's *v4l2_file_operations->open()* handler.

v4l2_fh_add (*fh*)

- Add a *v4l2_fh* to *video_device* file handle list. Must be called once the file handle is completely initialized.

v4l2_fh_del (*fh*)

- Unassociate the file handle from *video_device*. The file handle exit function may now be called.

v4l2_fh_exit (*fh*)

- Uninitialise the file handle. After uninitialisation the *v4l2_fh* memory can be freed.

If *struct v4l2_fh* is not embedded, then you can use these helper functions:

v4l2_fh_open (struct file *filp)

- This allocates a *struct v4l2_fh*, initializes it and adds it to the *struct video_device* associated with the file struct.

v4l2_fh_release (struct file *filp)

- This deletes it from the *struct video_device* associated with the file struct, uninitialised the *v4l2_fh* and frees it.

These two functions can be plugged into the *v4l2_file_operation*'s *open()* and *release()* ops.

Several drivers need to do something when the first file handle is opened and when the last file handle closes. Two helper functions were added to check whether the *v4l2_fh* struct is the only open filehandle of the associated device node:

v4l2_fh_is_singular (*fh*)

- Returns 1 if the file handle is the only open file handle, else 0.

v4l2_fh_is_singular_file (struct file *filp)

- Same, but it calls *v4l2_fh_is_singular* with *filp->private_data*.

2.2.6.1 V4L2 fh functions and data structures

struct **v4l2_fh**

Describes a V4L2 file handler

Definition

```
struct v4l2_fh {
    struct list_head      list;
    struct video_device   *vdev;
    struct v4l2_ctrl_handler *ctrl_handler;
    enum v4l2_priority    prio;
    wait_queue_head_t    wait;
    struct mutex          subscribe_lock;
    struct list_head      subscribed;
    struct list_head      available;
    unsigned int          navailable;
    u32 sequence;
    struct v4l2_m2m_ctx   *m2m_ctx;
};
```

Members

list list of file handlers

vdev pointer to *struct video_device*

ctrl_handler pointer to *struct v4l2_ctrl_handler*

prio priority of the file handler, as defined by enum *v4l2_priority*

wait event's wait queue

subscribe_lock serialise changes to the subscribed list; guarantee that the add and del event callbacks are orderly called

subscribed list of subscribed events

available list of events waiting to be dequeued

navailable number of available events at **available** list

sequence event sequence number

m2m_ctx pointer to *struct v4l2_m2m_ctx*

void **v4l2_fh_init**(struct *v4l2_fh* *fh, struct *video_device* *vdev)
Initialise the file handle.

Parameters

struct v4l2_fh *fh pointer to *struct v4l2_fh*

struct video_device *vdev pointer to *struct video_device*

Description

Parts of the V4L2 framework using the file handles should be initialised in this function. Must be called from driver's *v4l2_file_operations->open()* handler if the driver uses *struct v4l2_fh*.

void **v4l2_fh_add**(struct *v4l2_fh* *fh)

Add the fh to the list of file handles on a video_device.

Parameters

struct v4l2_fh *fh pointer to *struct v4l2_fh*

Description

Note: The **fh** file handle must be initialised first.

int **v4l2_fh_open**(struct file *filp)

Ancillary routine that can be used as the open() op of v4l2_file_operations.

Parameters

struct file *filp pointer to struct file

Description

It allocates a v4l2_fh and inits and adds it to the *struct video_device* associated with the file pointer.

void **v4l2_fh_del**(struct *v4l2_fh* *fh)

Remove file handle from the list of file handles.

Parameters

struct v4l2_fh *fh pointer to *struct v4l2_fh*

Description

On error filp->private_data will be NULL, otherwise it will point to the *struct v4l2_fh*.

Note: Must be called in v4l2_file_operations->release() handler if the driver uses *struct v4l2_fh*.

void **v4l2_fh_exit**(struct *v4l2_fh* *fh)

Release resources related to a file handle.

Parameters

struct v4l2_fh *fh pointer to *struct v4l2_fh*

Description

Parts of the V4L2 framework using the v4l2_fh must release their resources here, too.

Note: Must be called in v4l2_file_operations->release() handler if the driver uses *struct v4l2_fh*.

int **v4l2_fh_release**(struct file *filp)

Ancillary routine that can be used as the release() op of v4l2_file_operations.

Parameters

struct file *filp pointer to struct file

Description

It deletes and exits the `v4l2_fh` associated with the file pointer and frees it. It will do nothing if `filp->private_data` (the pointer to the `v4l2_fh` struct) is `NULL`.

This function always returns 0.

int `v4l2_fh_is_singular`(struct `v4l2_fh` *`fh`)

Returns 1 if this filehandle is the only filehandle opened for the associated `video_device`.

Parameters

struct `v4l2_fh` *`fh` pointer to `struct v4l2_fh`

Description

If `fh` is `NULL`, then it returns 0.

int `v4l2_fh_is_singular_file`(struct file *`filp`)

Returns 1 if this filehandle is the only filehandle opened for the associated `video_device`.

Parameters

struct file *`filp` pointer to struct file

Description

This is a helper function variant of `v4l2_fh_is_singular()` with uses struct file as argument.

If `filp->private_data` is `NULL`, then it will return 0.

2.2.7 V4L2 sub-devices

Many drivers need to communicate with sub-devices. These devices can do all sort of tasks, but most commonly they handle audio and/or video muxing, encoding or decoding. For webcams common sub-devices are sensors and camera controllers.

Usually these are I2C devices, but not necessarily. In order to provide the driver with a consistent interface to these sub-devices the `v4l2_subdev` struct (`v4l2-subdev.h`) was created.

Each sub-device driver must have a `v4l2_subdev` struct. This struct can be stand-alone for simple sub-devices or it might be embedded in a larger struct if more state information needs to be stored. Usually there is a low-level device struct (e.g. `i2c_client`) that contains the device data as setup by the kernel. It is recommended to store that pointer in the private data of `v4l2_subdev` using `v4l2_set_subdevdata()`. That makes it easy to go from a `v4l2_subdev` to the actual low-level bus-specific device data.

You also need a way to go from the low-level struct to `v4l2_subdev`. For the common `i2c_client` struct the `i2c_set_clientdata()` call is used to store a `v4l2_subdev` pointer, for other buses you may have to use other methods.

Bridges might also need to store per-subdev private data, such as a pointer to bridge-specific per-subdev private data. The `v4l2_subdev` structure provides host private data for that purpose that can be accessed with `v4l2_get_subdev_hostdata()` and `v4l2_set_subdev_hostdata()`.

From the bridge driver perspective, you load the sub-device module and somehow obtain the `v4l2_subdev` pointer. For i2c devices this is easy: you call `i2c_get_clientdata()`. For other buses something similar needs to be done. Helper functions exist for sub-devices on an I2C bus that do most of this tricky work for you.

Each `v4l2_subdev` contains function pointers that sub-device drivers can implement (or leave NULL if it is not applicable). Since sub-devices can do so many different things and you do not want to end up with a huge ops struct of which only a handful of ops are commonly implemented, the function pointers are sorted according to category and each category has its own ops struct.

The top-level ops struct contains pointers to the category ops structs, which may be NULL if the subdev driver does not support anything from that category.

It looks like this:

```

struct v4l2_subdev_core_ops {
    int (*log_status)(struct v4l2_subdev *sd);
    int (*init)(struct v4l2_subdev *sd, u32 val);
    ...
};

struct v4l2_subdev_tuner_ops {
    ...
};

struct v4l2_subdev_audio_ops {
    ...
};

struct v4l2_subdev_video_ops {
    ...
};

struct v4l2_subdev_pad_ops {
    ...
};

struct v4l2_subdev_ops {
    const struct v4l2_subdev_core_ops *core;
    const struct v4l2_subdev_tuner_ops *tuner;
    const struct v4l2_subdev_audio_ops *audio;
    const struct v4l2_subdev_video_ops *video;
    const struct v4l2_subdev_pad_ops *video;
};

```

The core ops are common to all subdevs, the other categories are implemented depending on the sub-device. E.g. a video device is unlikely to support the audio ops and vice versa.

This setup limits the number of function pointers while still making it easy to add new ops and categories.

A sub-device driver initializes the `v4l2_subdev` struct using:

```
v4l2_subdev_init(sd, &ops).
```

Afterwards you need to initialize `sd->name` with a unique name and set the module owner. This is done for you if you use the i2c helper functions.

If integration with the media framework is needed, you must initialize the `media_entity` struct embedded in the `v4l2_subdev` struct (entity field) by calling `media_entity_pads_init()`, if the entity has pads:

```

struct media_pad *pads = &my_sd->pads;
int err;

```

```
err = media_entity_pads_init(&sd->entity, npads, pads);
```

The pads array must have been previously initialized. There is no need to manually set the `struct media_entity` function and name fields, but the revision field must be initialized if needed.

A reference to the entity will be automatically acquired/released when the subdev device node (if any) is opened/closed.

Don't forget to cleanup the media entity before the sub-device is destroyed:

```
media_entity_cleanup(&sd->entity);
```

If a sub-device driver implements sink pads, the subdev driver may set the `link_validate` field in `v4l2_subdev_pad_ops` to provide its own link validation function. For every link in the pipeline, the `link_validate` pad operation of the sink end of the link is called. In both cases the driver is still responsible for validating the correctness of the format configuration between sub-devices and video nodes.

If `link_validate` op is not set, the default function `v4l2_subdev_link_validate_default()` is used instead. This function ensures that width, height and the media bus pixel code are equal on both source and sink of the link. Subdev drivers are also free to use this function to perform the checks mentioned above in addition to their own checks.

2.2.7.1 Subdev registration

There are currently two ways to register subdevices with the V4L2 core. The first (traditional) possibility is to have subdevices registered by bridge drivers. This can be done when the bridge driver has the complete information about subdevices connected to it and knows exactly when to register them. This is typically the case for internal subdevices, like video data processing units within SoCs or complex PCI(e) boards, camera sensors in USB cameras or connected to SoCs, which pass information about them to bridge drivers, usually in their platform data.

There are however also situations where subdevices have to be registered asynchronously to bridge devices. An example of such a configuration is a Device Tree based system where information about subdevices is made available to the system independently from the bridge devices, e.g. when subdevices are defined in DT as I2C device nodes. The API used in this second case is described further below.

Using one or the other registration method only affects the probing process, the run-time bridge-subdevice interaction is in both cases the same.

In the **synchronous** case a device (bridge) driver needs to register the `v4l2_subdev` with the `v4l2_device`:

```
v4l2_device_register_subdev(v4l2_dev, sd).
```

This can fail if the subdev module disappeared before it could be registered. After this function was called successfully the `subdev->dev` field points to the `v4l2_device`.

If the `v4l2_device` parent device has a non-NULL `mdev` field, the sub-device entity will be automatically registered with the media device.

You can unregister a sub-device using:

`v4l2_device_unregister_subdev(sd)`.

Afterwards the subdev module can be unloaded and `sd->dev == NULL`.

In the **asynchronous** case subdevice probing can be invoked independently of the bridge driver availability. The subdevice driver then has to verify whether all the requirements for a successful probing are satisfied. This can include a check for a master clock availability. If any of the conditions aren't satisfied the driver might decide to return `-EPROBE_DEFER` to request further reprobing attempts. Once all conditions are met the subdevice shall be registered using the `v4l2_async_register_subdev()` function. Unregistration is performed using the `v4l2_async_unregister_subdev()` call. Subdevices registered this way are stored in a global list of subdevices, ready to be picked up by bridge drivers.

Bridge drivers in turn have to register a notifier object. This is performed using the `v4l2_async_nf_register()` call. To unregister the notifier the driver has to call `v4l2_async_nf_unregister()`. The former of the two functions takes two arguments: a pointer to struct `v4l2_device` and a pointer to struct `v4l2_async_notifier`.

Before registering the notifier, bridge drivers must do two things: first, the notifier must be initialized using the `v4l2_async_nf_init()`. Second, bridge drivers can then begin to form a list of subdevice descriptors that the bridge device needs for its operation. Several functions are available to add subdevice descriptors to a notifier, depending on the type of device and the needs of the driver.

`v4l2_async_nf_add_fwnode_remote()` and `v4l2_async_nf_add_i2c()` are for bridge and ISP drivers for registering their async sub-devices with the notifier.

`v4l2_async_register_subdev_sensor()` is a helper function for sensor drivers registering their own async sub-device, but it also registers a notifier and further registers async sub-devices for lens and flash devices found in firmware. The notifier for the sub-device is unregistered with the async sub-device.

These functions allocate an async sub-device descriptor which is of type struct `v4l2_async_subdev` embedded in a driver-specific struct. The `&struct v4l2_async_subdev` shall be the first member of this struct:

```

struct my_async_subdev {
    struct v4l2_async_subdev asd;
    ...
};

struct my_async_subdev *my_asd;
struct fwnode_handle *ep;

...

my_asd = v4l2_async_nf_add_fwnode_remote(&notifier, ep,
                                       struct my_async_subdev);
fwnode_handle_put(ep);

if (IS_ERR(asd))
    return PTR_ERR(asd);

```

The V4L2 core will then use these descriptors to match asynchronously registered subdevices to them. If a match is detected the `.bound()` notifier callback is called. After all subdevices have been located the `.complete()` callback is called. When a subdevice is removed from the system the `.unbind()` method is called. All three callbacks are optional.

2.2.7.2 Calling subdev operations

The advantage of using `v4l2_subdev` is that it is a generic struct and does not contain any knowledge about the underlying hardware. So a driver might contain several subdevs that use an I2C bus, but also a subdev that is controlled through GPIO pins. This distinction is only relevant when setting up the device, but once the subdev is registered it is completely transparent.

Once the subdev has been registered you can call an ops function either directly:

```
err = sd->ops->core->g_std(sd, &norm);
```

but it is better and easier to use this macro:

```
err = v4l2_subdev_call(sd, core, g_std, &norm);
```

The macro will do the right NULL pointer checks and returns `-ENODEV` if `sd` is NULL, `-ENOIOCTLCMD` if either `sd->core` or `sd->core->g_std` is NULL, or the actual result of the `sd->ops->core->g_std` ops.

It is also possible to call all or a subset of the sub-devices:

```
v4l2_device_call_all(v4l2_dev, 0, core, g_std, &norm);
```

Any subdev that does not support this ops is skipped and error results are ignored. If you want to check for errors use this:

```
err = v4l2_device_call_until_err(v4l2_dev, 0, core, g_std, &norm);
```

Any error except `-ENOIOCTLCMD` will exit the loop with that error. If no errors (except `-ENOIOCTLCMD`) occurred, then 0 is returned.

The second argument to both calls is a group ID. If 0, then all subdevs are called. If non-zero, then only those whose group ID match that value will be called. Before a bridge driver registers a subdev it can set `sd->grp_id` to whatever value it wants (it's 0 by default). This value is owned by the bridge driver and the sub-device driver will never modify or use it.

The group ID gives the bridge driver more control how callbacks are called. For example, there may be multiple audio chips on a board, each capable of changing the volume. But usually only one will actually be used when the user want to change the volume. You can set the group ID for that subdev to e.g. `AUDIO_CONTROLLER` and specify that as the group ID value when calling `v4l2_device_call_all()`. That ensures that it will only go to the subdev that needs it.

If the sub-device needs to notify its `v4l2_device` parent of an event, then it can call `v4l2_subdev_notify(sd, notification, arg)`. This macro checks whether there is a `notify()` callback defined and returns `-ENODEV` if not. Otherwise the result of the `notify()` call is returned.

2.2.8 V4L2 sub-device userspace API

Bridge drivers traditionally expose one or multiple video nodes to userspace, and control sub-devices through the `v4l2_subdev_ops` operations in response to video node operations. This hides the complexity of the underlying hardware from applications. For complex devices, finer-grained control of the device than what the video nodes offer may be required. In those cases, bridge drivers that implement *the media controller API* may opt for making the subdevice operations directly accessible from userspace.

Device nodes named `v4l-subdevX` can be created in `/dev` to access sub-devices directly. If a sub-device supports direct userspace configuration it must set the `V4L2_SUBDEV_FL_HAS_DEVNODE` flag before being registered.

After registering sub-devices, the `v4l2_device` driver can create device nodes for all registered sub-devices marked with `V4L2_SUBDEV_FL_HAS_DEVNODE` by calling `v4l2_device_register_subdev_nodes()`. Those device nodes will be automatically removed when sub-devices are unregistered.

The device node handles a subset of the V4L2 API.

`VIDIOC_QUERYCTRL`, `VIDIOC_QUERYMENU`, `VIDIOC_G_CTRL`, `VIDIOC_S_CTRL`,
`VIDIOC_G_EXT_CTRL`, `VIDIOC_S_EXT_CTRL` and `VIDIOC_TRY_EXT_CTRL`:

The controls ioctls are identical to the ones defined in V4L2. They behave identically, with the only exception that they deal only with controls implemented in the sub-device. Depending on the driver, those controls can be also be accessed through one (or several) V4L2 device nodes.

`VIDIOC_DQEVENT`, `VIDIOC_SUBSCRIBE_EVENT` and `VIDIOC_UNSUBSCRIBE_EVENT`

The events ioctls are identical to the ones defined in V4L2. They behave identically, with the only exception that they deal only with events generated by the sub-device. Depending on the driver, those events can also be reported by one (or several) V4L2 device nodes.

Sub-device drivers that want to use events need to set the `V4L2_SUBDEV_FL_HAS_EVENTS` `v4l2_subdev.flags` before registering the sub-device. After registration events can be queued as usual on the `v4l2_subdev.devnode` device node.

To properly support events, the `poll()` file operation is also implemented.

Private ioctls

All ioctls not in the above list are passed directly to the sub-device driver through the `core::ioctl` operation.

2.2.9 Read-only sub-device userspace API

Bridge drivers that control their connected subdevices through direct calls to the kernel API realized by `v4l2_subdev_ops` structure do not usually want userspace to be able to change the same parameters through the subdevice device node and thus do not usually register any.

It is sometimes useful to report to userspace the current subdevice configuration through a read-only API, that does not permit applications to change to the device parameters but allows interfacing to the subdevice device node to inspect them.

For instance, to implement cameras based on computational photography, userspace needs to know the detailed camera sensor configuration (in terms of skipping, binning, cropping and scaling) for each supported output resolution. To support such use cases, bridge drivers may expose the subdevice operations to userspace through a read-only API.

To create a read-only device node for all the subdevices registered with the `V4L2_SUBDEV_FL_HAS_DEVNODE` set, the `v4l2_device` driver should call `v4l2_device_register_ro_subdev_nodes()`.

Access to the following ioctls for userspace applications is restricted on sub-device device nodes registered with `v4l2_device_register_ro_subdev_nodes()`.

`VIDIOC_SUBDEV_S_FMT`, `VIDIOC_SUBDEV_S_CROP`, `VIDIOC_SUBDEV_S_SELECTION`:

These ioctls are only allowed on a read-only subdevice device node for the `V4L2_SUBDEV_FORMAT_TRY` formats and selection rectangles.

`VIDIOC_SUBDEV_S_FRAME_INTERVAL`, `VIDIOC_SUBDEV_S_DV_TIMINGS`, `VIDIOC_SUBDEV_S_STD`:

These ioctls are not allowed on a read-only subdevice node.

In case the ioctl is not allowed, or the format to modify is set to `V4L2_SUBDEV_FORMAT_ACTIVE`, the core returns a negative error code and the `errno` variable is set to `-EPERM`.

2.2.10 I2C sub-device drivers

Since these drivers are so common, special helper functions are available to ease the use of these drivers (`v4l2-common.h`).

The recommended method of adding `v4l2_subdev` support to an I2C driver is to embed the `v4l2_subdev` struct into the state struct that is created for each I2C device instance. Very simple devices have no state struct and in that case you can just create a `v4l2_subdev` directly.

A typical state struct would look like this (where 'chipname' is replaced by the name of the chip):

```
struct chipname_state {
    struct v4l2_subdev sd;
    ... /* additional state fields */
};
```

Initialize the `v4l2_subdev` struct as follows:

```
v4l2_i2c_subdev_init(&state->sd, client, subdev_ops);
```

This function will fill in all the fields of `v4l2_subdev` ensure that the `v4l2_subdev` and `i2c_client` both point to one another.

You should also add a helper inline function to go from a `v4l2_subdev` pointer to a `chipname_state` struct:

```
static inline struct chipname_state *to_state(struct v4l2_subdev *sd)
{
    return container_of(sd, struct chipname_state, sd);
}
```

Use this to go from the `v4l2_subdev` struct to the `i2c_client` struct:

```
struct i2c_client *client = v4l2_get_subdevdata(sd);
```

And this to go from an `i2c_client` to a `v4l2_subdev` struct:

```
struct v4l2_subdev *sd = i2c_get_clientdata(client);
```

Make sure to call `v4l2_device_unregister_subdev(sd)` when the `remove()` callback is called. This will unregister the sub-device from the bridge driver. It is safe to call this even if the sub-device was never registered.

You need to do this because when the bridge driver destroys the i2c adapter the `remove()` callbacks are called of the i2c devices on that adapter. After that the corresponding `v4l2_subdev` structures are invalid, so they have to be unregistered first. Calling `v4l2_device_unregister_subdev(sd)` from the `remove()` callback ensures that this is always done correctly.

The bridge driver also has some helper functions it can use:

```
struct v4l2_subdev *sd = v4l2_i2c_new_subdev(v4l2_dev, adapter,
                                           "module_foo", "chipid", 0x36, NULL);
```

This loads the given module (can be `NULL` if no module needs to be loaded) and calls `i2c_new_client_device()` with the given `i2c_adapter` and `chip/address` arguments. If all goes well, then it registers the subdev with the `v4l2_device`.

You can also use the last argument of `v4l2_i2c_new_subdev()` to pass an array of possible I2C addresses that it should probe. These probe addresses are only used if the previous argument is 0. A non-zero argument means that you know the exact i2c address so in that case no probing will take place.

Both functions return `NULL` if something went wrong.

Note that the `chipid` you pass to `v4l2_i2c_new_subdev()` is usually the same as the module name. It allows you to specify a chip variant, e.g. "saa7114" or "saa7115". In general though the i2c driver autodetects this. The use of `chipid` is something that needs to be looked at more closely at a later date. It differs between i2c drivers and as such can be confusing. To see which chip variants are supported you can look in the i2c driver code for the `i2c_device_id` table. This lists all the possibilities.

There are one more helper function:

`v4l2_i2c_new_subdev_board()` uses an `i2c_board_info` struct which is passed to the i2c driver and replaces the `irq`, `platform_data` and `addr` arguments.

If the subdev supports the `s_config` core ops, then that op is called with the `irq` and `platform_data` arguments after the subdev was setup.

The `v4l2_i2c_new_subdev()` function will call `v4l2_i2c_new_subdev_board()`, internally filling a `i2c_board_info` structure using the `client_type` and the `addr` to fill it.

2.2.11 V4L2 sub-device functions and data structures

struct **v4l2_decode_vbi_line**
used to decode_vbi_line

Definition

```
struct v4l2_decode_vbi_line {
    u32 is_second_field;
    u8 *p;
    u32 line;
    u32 type;
};
```

Members

is_second_field Set to 0 for the first (odd) field; set to 1 for the second (even) field.

p Pointer to the sliced VBI data from the decoder. On exit, points to the start of the payload.

line Line number of the sliced VBI data (1-23)

type VBI service type (V4L2_SLICED_*). 0 if no service found

enum **v4l2_subdev_io_pin_bits**
Subdevice external IO pin configuration bits

Constants

V4L2_SUBDEV_IO_PIN_DISABLE disables a pin config. ENABLE assumed.

V4L2_SUBDEV_IO_PIN_OUTPUT set it if pin is an output.

V4L2_SUBDEV_IO_PIN_INPUT set it if pin is an input.

V4L2_SUBDEV_IO_PIN_SET_VALUE to set the output value via *struct v4l2_subdev_io_pin_config->value*.

V4L2_SUBDEV_IO_PIN_ACTIVE_LOW pin active is bit 0. Otherwise, ACTIVE HIGH is assumed.

struct **v4l2_subdev_io_pin_config**
Subdevice external IO pin configuration

Definition

```
struct v4l2_subdev_io_pin_config {
    u32 flags;
    u8 pin;
    u8 function;
    u8 value;
    u8 strength;
};
```

Members

flags bitmask with flags for this pin's config, whose bits are defined by *enum v4l2_subdev_io_pin_bits*.

pin Chip external IO pin to configure

function Internal signal pad/function to route to IO pin

value Initial value for pin - e.g. GPIO output value

strength Pin drive strength

struct **v4l2_subdev_core_ops**

Define core ops callbacks for subdevs

Definition

```

struct v4l2_subdev_core_ops {
    int (*log_status)(struct v4l2_subdev *sd);
    int (*s_io_pin_config)(struct v4l2_subdev *sd, size_t n, struct v4l2_subdev_io_pin_
↳config *pincfg);
    int (*init)(struct v4l2_subdev *sd, u32 val);
    int (*load_fw)(struct v4l2_subdev *sd);
    int (*reset)(struct v4l2_subdev *sd, u32 val);
    int (*s_gpio)(struct v4l2_subdev *sd, u32 val);
    long (*command)(struct v4l2_subdev *sd, unsigned int cmd, void *arg);
    long (*ioctl)(struct v4l2_subdev *sd, unsigned int cmd, void *arg);
#ifdef CONFIG_COMPAT;
    long (*compat_ioctl32)(struct v4l2_subdev *sd, unsigned int cmd, unsigned long arg);
#endif;
#ifdef CONFIG_VIDEO_ADV_DEBUG;
    int (*g_register)(struct v4l2_subdev *sd, struct v4l2_dbg_register *reg);
    int (*s_register)(struct v4l2_subdev *sd, const struct v4l2_dbg_register *reg);
#endif;
    int (*s_power)(struct v4l2_subdev *sd, int on);
    int (*interrupt_service_routine)(struct v4l2_subdev *sd, u32 status, bool *handled);
    int (*subscribe_event)(struct v4l2_subdev *sd, struct v4l2_fh *fh, struct v4l2_event_
↳subscription *sub);
    int (*unsubscribe_event)(struct v4l2_subdev *sd, struct v4l2_fh *fh, struct v4l2_
↳event_subscription *sub);
};

```

Members

log_status callback for VIDIOC_LOG_STATUS() ioctl handler code.

s_io_pin_config configure one or more chip I/O pins for chips that multiplex different internal signal pads out to IO pins. This function takes a pointer to an array of 'n' pin configuration entries, one for each pin being configured. This function could be called at times other than just subdevice initialization.

init initialize the sensor registers to some sort of reasonable default values. Do not use for new drivers and should be removed in existing drivers.

load_fw load firmware.

reset generic reset command. The argument selects which subsystems to reset. Passing 0 will always reset the whole chip. Do not use for new drivers without discussing this first on the linux-media mailinglist. There should be no reason normally to reset a device.

s_gpio set GPIO pins. Very simple right now, might need to be extended with a direction argument if needed.

command called by in-kernel drivers in order to call functions internal to subdev drivers driver that have a separate callback.

ioctl called at the end of ioctl() syscall handler at the V4L2 core. used to provide support for private ioctls used on the driver.

compat_ioctl32 called when a 32 bits application uses a 64 bits Kernel, in order to fix data passed from/to userspace.

g_register callback for VIDIOC_DBG_G_REGISTER() ioctl handler code.

s_register callback for VIDIOC_DBG_S_REGISTER() ioctl handler code.

s_power puts subdevice in power saving mode (on == 0) or normal operation mode (on == 1).

interrupt_service_routine Called by the bridge chip's interrupt service handler, when an interrupt status has be raised due to this subdev, so that this subdev can handle the details. It may schedule work to be performed later. It must not sleep. **Called from an IRQ context.**

subscribe_event used by the drivers to request the control framework that for it to be warned when the value of a control changes.

unsubscribe_event remove event subscription from the control framework.

struct **v4l2_subdev_tuner_ops**

Callbacks used when v4l device was opened in radio mode.

Definition

```
struct v4l2_subdev_tuner_ops {
    int (*standby)(struct v4l2_subdev *sd);
    int (*s_radio)(struct v4l2_subdev *sd);
    int (*s_frequency)(struct v4l2_subdev *sd, const struct v4l2_frequency *freq);
    int (*g_frequency)(struct v4l2_subdev *sd, struct v4l2_frequency *freq);
    int (*enum_freq_bands)(struct v4l2_subdev *sd, struct v4l2_frequency_band *band);
    int (*g_tuner)(struct v4l2_subdev *sd, struct v4l2_tuner *vt);
    int (*s_tuner)(struct v4l2_subdev *sd, const struct v4l2_tuner *vt);
    int (*g_modulator)(struct v4l2_subdev *sd, struct v4l2_modulator *vm);
    int (*s_modulator)(struct v4l2_subdev *sd, const struct v4l2_modulator *vm);
    int (*s_type_addr)(struct v4l2_subdev *sd, struct tuner_setup *type);
    int (*s_config)(struct v4l2_subdev *sd, const struct v4l2_priv_tun_config *config);
};
```

Members

standby puts the tuner in standby mode. It will be woken up automatically the next time it is used.

s_radio callback that switches the tuner to radio mode. drivers should explicitly call it when a tuner ops should operate on radio mode, before being able to handle it. Used on devices that have both AM/FM radio receiver and TV.

s_frequency callback for VIDIOC_S_FREQUENCY() ioctl handler code.

g_frequency callback for VIDIOC_G_FREQUENCY() ioctl handler code. freq->type must be filled in. Normally done by [video_ioctl2\(\)](#) or the bridge driver.

enum_freq_bands callback for VIDIOC_ENUM_FREQ_BANDS() ioctl handler code.

g_tuner callback for VIDIOC_G_TUNER() ioctl handler code.

s_tuner callback for VIDIOC_S_TUNER() ioctl handler code. **vt->type** must be filled in. Normally done by video_ioctl2 or the bridge driver.

g_modulator callback for VIDIOC_G_MODULATOR() ioctl handler code.

s_modulator callback for VIDIOC_S_MODULATOR() ioctl handler code.

s_type_addr sets tuner type and its I2C addr.

s_config sets tda9887 specific stuff, like port1, port2 and qss

Description

Note: On devices that have both AM/FM and TV, it is up to the driver to explicitly call `s_radio` when the tuner should be switched to radio mode, before handling other `struct v4l2_subdev_tuner_ops` that would require it. An example of such usage is:

```
static void s_frequency(void *priv, const struct v4l2_frequency *f)
{
    ...
    if (f.type == V4L2_TUNER_RADIO)
        v4l2_device_call_all(v4l2_dev, 0, tuner, s_radio);
    ...
    v4l2_device_call_all(v4l2_dev, 0, tuner, s_frequency);
}
```

struct v4l2_subdev_audio_ops

Callbacks used for audio-related settings

Definition

```
struct v4l2_subdev_audio_ops {
    int (*s_clock_freq)(struct v4l2_subdev *sd, u32 freq);
    int (*s_i2s_clock_freq)(struct v4l2_subdev *sd, u32 freq);
    int (*s_routing)(struct v4l2_subdev *sd, u32 input, u32 output, u32 config);
    int (*s_stream)(struct v4l2_subdev *sd, int enable);
};
```

Members

s_clock_freq set the frequency (in Hz) of the audio clock output. Used to slave an audio processor to the video decoder, ensuring that audio and video remain synchronized. Usual values for the frequency are 48000, 44100 or 32000 Hz. If the frequency is not supported, then -EINVAL is returned.

s_i2s_clock_freq sets I2S speed in bps. This is used to provide a standard way to select I2S clock used by driving digital audio streams at some board designs. Usual values for the frequency are 1024000 and 2048000. If the frequency is not supported, then -EINVAL is returned.

s_routing used to define the input and/or output pins of an audio chip, and any additional configuration data. Never attempt to use user-level input IDs (e.g. Composite, S-Video, Tuner) at this level. An i2c device shouldn't know about whether an input pin is connected to a Composite connector, become on another board or platform it might be connected to something else entirely. The calling driver is responsible for mapping a user-level input to the right pins on the i2c device.

s_stream used to notify the audio code that stream will start or has stopped.

enum **v4l2_mbus_frame_desc_flags**
media bus frame description flags

Constants

V4L2_MBUS_FRAME_DESC_FL_LEN_MAX

Indicates that *struct v4l2_mbus_frame_desc_entry*->length field specifies maximum data length.

V4L2_MBUS_FRAME_DESC_FL_BLOB

Indicates that the format does not have line offsets, i.e. the receiver should use 1D DMA.

struct **v4l2_mbus_frame_desc_entry**
media bus frame description structure

Definition

```
struct v4l2_mbus_frame_desc_entry {
    enum v4l2_mbus_frame_desc_flags flags;
    u32 pixelcode;
    u32 length;
};
```

Members

flags bitmask flags, as defined by *enum v4l2_mbus_frame_desc_flags*.

pixelcode media bus pixel code, valid if **flags** FRAME_DESC_FL_BLOB is not set.

length number of octets per frame, valid if **flags** V4L2_MBUS_FRAME_DESC_FL_LEN_MAX is set.

struct **v4l2_mbus_frame_desc**
media bus data frame description

Definition

```
struct v4l2_mbus_frame_desc {
    struct v4l2_mbus_frame_desc_entry entry[V4L2_FRAME_DESC_ENTRY_MAX];
    unsigned short num_entries;
};
```

Members

entry frame descriptors array

num_entries number of entries in **entry** array

enum **v4l2_subdev_pre_streamon_flags**
Flags for pre_streamon subdev core op

Constants

V4L2_SUBDEV_PRE_STREAMON_FL_MANUAL_LP Set the transmitter to either LP-11 or LP-111 mode before call to s_stream().

struct **v4l2_subdev_video_ops**
Callbacks used when v4l device was opened in video mode.

Definition

```

struct v4l2_subdev_video_ops {
    int (*s_routing)(struct v4l2_subdev *sd, u32 input, u32 output, u32 config);
    int (*s_crystal_freq)(struct v4l2_subdev *sd, u32 freq, u32 flags);
    int (*g_std)(struct v4l2_subdev *sd, v4l2_std_id *norm);
    int (*s_std)(struct v4l2_subdev *sd, v4l2_std_id norm);
    int (*s_std_output)(struct v4l2_subdev *sd, v4l2_std_id std);
    int (*g_std_output)(struct v4l2_subdev *sd, v4l2_std_id *std);
    int (*querystd)(struct v4l2_subdev *sd, v4l2_std_id *std);
    int (*g_tvnorms)(struct v4l2_subdev *sd, v4l2_std_id *std);
    int (*g_tvnorms_output)(struct v4l2_subdev *sd, v4l2_std_id *std);
    int (*g_input_status)(struct v4l2_subdev *sd, u32 *status);
    int (*s_stream)(struct v4l2_subdev *sd, int enable);
    int (*g_pixelaspect)(struct v4l2_subdev *sd, struct v4l2_fract *aspect);
    int (*g_frame_interval)(struct v4l2_subdev *sd, struct v4l2_subdev_frame_interval_
↪*interval);
    int (*s_frame_interval)(struct v4l2_subdev *sd, struct v4l2_subdev_frame_interval_
↪*interval);
    int (*s_dv_timings)(struct v4l2_subdev *sd, struct v4l2_dv_timings *timings);
    int (*g_dv_timings)(struct v4l2_subdev *sd, struct v4l2_dv_timings *timings);
    int (*query_dv_timings)(struct v4l2_subdev *sd, struct v4l2_dv_timings *timings);
    int (*s_rx_buffer)(struct v4l2_subdev *sd, void *buf, unsigned int *size);
    int (*pre_streamon)(struct v4l2_subdev *sd, u32 flags);
    int (*post_streamoff)(struct v4l2_subdev *sd);
};

```

Members

s_routing see s_routing in audio_ops, except this version is for video devices.

s_crystal_freq sets the frequency of the crystal used to generate the clocks in Hz. An extra flags field allows device specific configuration regarding clock frequency dividers, etc. If not used, then set flags to 0. If the frequency is not supported, then -EINVAL is returned.

g_std callback for VIDIOC_G_STD() ioctl handler code.

s_std callback for VIDIOC_S_STD() ioctl handler code.

s_std_output set v4l2_std_id for video OUTPUT devices. This is ignored by video input devices.

g_std_output get current standard for video OUTPUT devices. This is ignored by video input devices.

querystd callback for VIDIOC_QUERYSTD() ioctl handler code.

g_tvnorms get v4l2_std_id with all standards supported by the video CAPTURE device. This is ignored by video output devices.

g_tvnorms_output get v4l2_std_id with all standards supported by the video OUTPUT device. This is ignored by video capture devices.

g_input_status get input status. Same as the status field in the struct v4l2_input

s_stream used to notify the driver that a video stream will start or has stopped.

g_pixelaspect callback to return the pixelaspect ratio.

g_frame_interval callback for VIDIOC_SUBDEV_G_FRAME_INTERVAL() ioctl handler code.

s_frame_interval callback for VIDIOC_SUBDEV_S_FRAME_INTERVAL() ioctl handler code.

s_dv_timings Set custom dv timings in the sub device. This is used when sub device is capable of setting detailed timing information in the hardware to generate/detect the video signal.

g_dv_timings Get custom dv timings in the sub device.

query_dv_timings callback for VIDIOC_QUERY_DV_TIMINGS() ioctl handler code.

s_rx_buffer set a host allocated memory buffer for the subdev. The subdev can adjust **size** to a lower value and must not write more data to the buffer starting at **data** than the original value of **size**.

pre_streamon May be called before streaming is actually started, to help initialising the bus. Current usage is to set a CSI-2 transmitter to LP-11 or LP-111 mode before streaming. See [enum v4l2_subdev_pre_streamon_flags](#).

pre_streamon shall return error if it cannot perform the operation as indicated by the flags argument. In particular, -EACCES indicates lack of support for the operation. The caller shall call post_streamoff for each successful call of pre_streamon.

post_streamoff Called after streaming is stopped, but if and only if pre_streamon was called earlier.

struct **v4l2_subdev_vbi_ops**

Callbacks used when v4l device was opened in video mode via the vbi device node.

Definition

```
struct v4l2_subdev_vbi_ops {
    int (*decode_vbi_line)(struct v4l2_subdev *sd, struct v4l2_decode_vbi_line *vbi_
    ↪line);
    int (*s_vbi_data)(struct v4l2_subdev *sd, const struct v4l2_sliced_vbi_data *vbi_
    ↪data);
    int (*g_vbi_data)(struct v4l2_subdev *sd, struct v4l2_sliced_vbi_data *vbi_data);
    int (*g_sliced_vbi_cap)(struct v4l2_subdev *sd, struct v4l2_sliced_vbi_cap *cap);
    int (*s_raw_fmt)(struct v4l2_subdev *sd, struct v4l2_vbi_format *fmt);
    int (*g_sliced_fmt)(struct v4l2_subdev *sd, struct v4l2_sliced_vbi_format *fmt);
    int (*s_sliced_fmt)(struct v4l2_subdev *sd, struct v4l2_sliced_vbi_format *fmt);
};
```

Members

decode_vbi_line video decoders that support sliced VBI need to implement this ioctl. Field p of the [struct v4l2_decode_vbi_line](#) is set to the start of the VBI data that was generated by the decoder. The driver then parses the sliced VBI data and sets the other fields in the struct accordingly. The pointer p is updated to point to the start of the payload which can be copied verbatim into the data field of the [struct v4l2_sliced_vbi_data](#). If no valid VBI data was found, then the type field is set to 0 on return.

s_vbi_data used to generate VBI signals on a video signal. [struct v4l2_sliced_vbi_data](#) is filled with the data packets that should be output. Note that if you set the line field to 0, then that VBI signal is disabled. If no valid VBI data was found, then the type field is set to 0 on return.

g_vbi_data used to obtain the sliced VBI packet from a readback register. Not all video decoders support this. If no data is available because the readback register contains invalid or erroneous data -EIO is returned. Note that you must fill in the 'id' member and the 'field' member (to determine whether CC data from the first or second field should be obtained).

g_sliced_vbi_cap callback for VIDIOC_G_SLICED_VBI_CAP() ioctl handler code.

s_raw_fmt setup the video encoder/decoder for raw VBI.

g_sliced_fmt retrieve the current sliced VBI settings.

s_sliced_fmt setup the sliced VBI settings.

struct **v4l2_subdev_sensor_ops**
v4l2-subdev sensor operations

Definition

```
struct v4l2_subdev_sensor_ops {
    int (*g_skip_top_lines)(struct v4l2_subdev *sd, u32 *lines);
    int (*g_skip_frames)(struct v4l2_subdev *sd, u32 *frames);
};
```

Members

g_skip_top_lines number of lines at the top of the image to be skipped. This is needed for some sensors, which always corrupt several top lines of the output image, or which send their metadata in them.

g_skip_frames number of frames to skip at stream start. This is needed for buggy sensors that generate faulty frames when they are turned on.

enum **v4l2_subdev_ir_mode**
describes the type of IR supported

Constants

V4L2_SUBDEV_IR_MODE_PULSE_WIDTH IR uses struct `ir_raw_event` records

struct **v4l2_subdev_ir_parameters**
Parameters for IR TX or TX

Definition

```
struct v4l2_subdev_ir_parameters {
    unsigned int bytes_per_data_element;
    enum v4l2_subdev_ir_mode mode;
    bool enable;
    bool interrupt_enable;
    bool shutdown;
    bool modulation;
    u32 max_pulse_width;
    unsigned int carrier_freq;
    unsigned int duty_cycle;
    bool invert_level;
    bool invert_carrier_sense;
    u32 noise_filter_min_width;
    unsigned int carrier_range_lower;
    unsigned int carrier_range_upper;
    u32 resolution;
};
```

Members

bytes_per_data_element bytes per data element of data in read or write call.

mode IR mode as defined by *enum v4l2_subdev_ir_mode*.

enable device is active if true

interrupt_enable IR interrupts are enabled if true

shutdown if true: set hardware to low/no power, false: normal mode

modulation if true, it uses carrier, if false: baseband

max_pulse_width maximum pulse width in ns, valid only for baseband signal

carrier_freq carrier frequency in Hz, valid only for modulated signal

duty_cycle duty cycle percentage, valid only for modulated signal

invert_level invert signal level

invert_carrier_sense Send 0/space as a carrier burst. used only in TX.

noise_filter_min_width min time of a valid pulse, in ns. Used only for RX.

carrier_range_lower Lower carrier range, in Hz, valid only for modulated signal. Used only for RX.

carrier_range_upper Upper carrier range, in Hz, valid only for modulated signal. Used only for RX.

resolution The receive resolution, in ns . Used only for RX.

struct **v4l2_subdev_ir_ops**
operations for IR subdevices

Definition

```
struct v4l2_subdev_ir_ops {
    int (*rx_read)(struct v4l2_subdev *sd, u8 *buf, size_t count, ssize_t *num);
    int (*rx_g_parameters)(struct v4l2_subdev *sd, struct v4l2_subdev_ir_parameters
↪ *params);
    int (*rx_s_parameters)(struct v4l2_subdev *sd, struct v4l2_subdev_ir_parameters
↪ *params);
    int (*tx_write)(struct v4l2_subdev *sd, u8 *buf, size_t count, ssize_t *num);
    int (*tx_g_parameters)(struct v4l2_subdev *sd, struct v4l2_subdev_ir_parameters
↪ *params);
    int (*tx_s_parameters)(struct v4l2_subdev *sd, struct v4l2_subdev_ir_parameters
↪ *params);
};
```

Members

rx_read Reads received codes or pulse width data. The semantics are similar to a non-blocking read() call.

rx_g_parameters Get the current operating parameters and state of the IR receiver.

rx_s_parameters Set the current operating parameters and state of the IR receiver. It is recommended to call [rt]x_g_parameters first to fill out the current state, and only change the fields that need to be changed. Upon return, the actual device operating parameters and state will be returned. Note that hardware limitations may prevent the actual settings from matching the requested settings - e.g. an actual carrier setting of 35,904 Hz when 36,000 Hz was requested. An exception is when the shutdown parameter is true. The last used operational parameters will be returned, but the actual state of the hardware be different to minimize power consumption and processing when shutdown is true.

tx_write Writes codes or pulse width data for transmission. The semantics are similar to a non-blocking write() call.

tx_g_parameters Get the current operating parameters and state of the IR transmitter.

tx_s_parameters Set the current operating parameters and state of the IR transmitter. It is recommended to call [rt]x_g_parameters first to fill out the current state, and only change the fields that need to be changed. Upon return, the actual device operating parameters and state will be returned. Note that hardware limitations may prevent the actual settings from matching the requested settings - e.g. an actual carrier setting of 35,904 Hz when 36,000 Hz was requested. An exception is when the shutdown parameter is true. The last used operational parameters will be returned, but the actual state of the hardware be different to minimize power consumption and processing when shutdown is true.

struct **v4l2_subdev_pad_config**

Used for storing subdev pad information.

Definition

```
struct v4l2_subdev_pad_config {
    struct v4l2_mbus_framefmt try_fmt;
    struct v4l2_rect try_crop;
    struct v4l2_rect try_compose;
};
```

Members

try_fmt struct v4l2_mbus_framefmt

try_crop struct v4l2_rect to be used for crop

try_compose struct v4l2_rect to be used for compose

Description

This structure only needs to be passed to the pad op if the 'which' field of the main argument is set to V4L2_SUBDEV_FORMAT_TRY. For V4L2_SUBDEV_FORMAT_ACTIVE it is safe to pass NULL.

struct **v4l2_subdev_state**

Used for storing subdev state information.

Definition

```
struct v4l2_subdev_state {
    struct v4l2_subdev_pad_config *pads;
};
```

Members

pads *struct v4l2_subdev_pad_config* array

Description

This structure only needs to be passed to the pad op if the 'which' field of the main argument is set to V4L2_SUBDEV_FORMAT_TRY. For V4L2_SUBDEV_FORMAT_ACTIVE it is safe to pass NULL.

struct **v4l2_subdev_pad_ops**

v4l2-subdev pad level operations

Definition

```

struct v4l2_subdev_pad_ops {
    int (*init_cfg)(struct v4l2_subdev *sd, struct v4l2_subdev_state *state);
    int (*enum_mbus_code)(struct v4l2_subdev *sd, struct v4l2_subdev_state *state, struct
↳ v4l2_subdev_mbus_code_enum *code);
    int (*enum_frame_size)(struct v4l2_subdev *sd, struct v4l2_subdev_state *state,
↳ struct v4l2_subdev_frame_size_enum *fse);
    int (*enum_frame_interval)(struct v4l2_subdev *sd, struct v4l2_subdev_state *state,
↳ struct v4l2_subdev_frame_interval_enum *fie);
    int (*get_fmt)(struct v4l2_subdev *sd, struct v4l2_subdev_state *state, struct v4l2_
↳ subdev_format *format);
    int (*set_fmt)(struct v4l2_subdev *sd, struct v4l2_subdev_state *state, struct v4l2_
↳ subdev_format *format);
    int (*get_selection)(struct v4l2_subdev *sd, struct v4l2_subdev_state *state, struct
↳ v4l2_subdev_selection *sel);
    int (*set_selection)(struct v4l2_subdev *sd, struct v4l2_subdev_state *state, struct
↳ v4l2_subdev_selection *sel);
    int (*get_edid)(struct v4l2_subdev *sd, struct v4l2_edid *edid);
    int (*set_edid)(struct v4l2_subdev *sd, struct v4l2_edid *edid);
    int (*dv_timings_cap)(struct v4l2_subdev *sd, struct v4l2_dv_timings_cap *cap);
    int (*enum_dv_timings)(struct v4l2_subdev *sd, struct v4l2_enum_dv_timings *timings);
#ifdef CONFIG_MEDIA_CONTROLLER;
    int (*link_validate)(struct v4l2_subdev *sd, struct media_link *link, struct v4l2_
↳ subdev_format *source_fmt, struct v4l2_subdev_format *sink_fmt);
#endif ;
    int (*get_frame_desc)(struct v4l2_subdev *sd, unsigned int pad, struct v4l2_mbus_
↳ frame_desc *fd);
    int (*set_frame_desc)(struct v4l2_subdev *sd, unsigned int pad, struct v4l2_mbus_
↳ frame_desc *fd);
    int (*get_mbus_config)(struct v4l2_subdev *sd, unsigned int pad, struct v4l2_mbus_
↳ config *config);
    int (*set_mbus_config)(struct v4l2_subdev *sd, unsigned int pad, struct v4l2_mbus_
↳ config *config);
};

```

Members

init_cfg initialize the pad config to default values

enum_mbus_code callback for VIDIOC_SUBDEV_ENUM_MBUS_CODE() ioctl handler code.

enum_frame_size callback for VIDIOC_SUBDEV_ENUM_FRAME_SIZE() ioctl handler code.

enum_frame_interval callback for VIDIOC_SUBDEV_ENUM_FRAME_INTERVAL() ioctl handler code.

get_fmt callback for VIDIOC_SUBDEV_G_FMT() ioctl handler code.

set_fmt callback for VIDIOC_SUBDEV_S_FMT() ioctl handler code.

get_selection callback for VIDIOC_SUBDEV_G_SELECTION() ioctl handler code.

set_selection callback for VIDIOC_SUBDEV_S_SELECTION() ioctl handler code.

get_edid callback for VIDIOC_SUBDEV_G_EDID() ioctl handler code.

set_edid callback for VIDIOC_SUBDEV_S_EDID() ioctl handler code.

dv_timings_cap callback for VIDIOC_SUBDEV_DV_TIMINGS_CAP() ioctl handler code.

enum_dv_timings callback for VIDIOC_SUBDEV_ENUM_DV_TIMINGS() ioctl handler code.

link_validate used by the media controller code to check if the links that belongs to a pipeline can be used for stream.

get_frame_desc get the current low level media bus frame parameters.

set_frame_desc set the low level media bus frame parameters, **fd** array may be adjusted by the subdev driver to device capabilities.

get_mbus_config get the media bus configuration of a remote sub-device. The media bus configuration is usually retrieved from the firmware interface at sub-device probe time, immediately applied to the hardware and eventually adjusted by the driver. Remote sub-devices (usually video receivers) shall use this operation to query the transmitting end bus configuration in order to adjust their own one accordingly. Callers should make sure they get the most up-to-date as possible configuration from the remote end, likely calling this operation as close as possible to stream on time. The operation shall fail if the pad index it has been called on is not valid or in case of unrecoverable failures.

set_mbus_config set the media bus configuration of a remote sub-device. This operations is intended to allow, in combination with the `get_mbus_config` operation, the negotiation of media bus configuration parameters between media sub-devices. The operation shall not fail if the requested configuration is not supported, but the driver shall update the content of the `config` argument to reflect what has been actually applied to the hardware. The operation shall fail if the pad index it has been called on is not valid or in case of unrecoverable failures.

```
struct v4l2_subdev_ops
    Subdev operations
```

Definition

```
struct v4l2_subdev_ops {
    const struct v4l2_subdev_core_ops    *core;
    const struct v4l2_subdev_tuner_ops   *tuner;
    const struct v4l2_subdev_audio_ops   *audio;
    const struct v4l2_subdev_video_ops   *video;
    const struct v4l2_subdev_vbi_ops     *vbi;
    const struct v4l2_subdev_ir_ops      *ir;
    const struct v4l2_subdev_sensor_ops  *sensor;
    const struct v4l2_subdev_pad_ops     *pad;
};
```

Members

core pointer to *struct v4l2_subdev_core_ops*. Can be NULL

tuner pointer to *struct v4l2_subdev_tuner_ops*. Can be NULL

audio pointer to *struct v4l2_subdev_audio_ops*. Can be NULL

video pointer to *struct v4l2_subdev_video_ops*. Can be NULL

vbi pointer to *struct v4l2_subdev_vbi_ops*. Can be NULL

ir pointer to *struct v4l2_subdev_ir_ops*. Can be NULL

sensor pointer to *struct v4l2_subdev_sensor_ops*. Can be NULL

pad pointer to *struct v4l2_subdev_pad_ops*. Can be NULL

struct **v4l2_subdev_internal_ops**
V4L2 subdev internal ops

Definition

```
struct v4l2_subdev_internal_ops {
    int (*registered)(struct v4l2_subdev *sd);
    void (*unregistered)(struct v4l2_subdev *sd);
    int (*open)(struct v4l2_subdev *sd, struct v4l2_subdev_fh *fh);
    int (*close)(struct v4l2_subdev *sd, struct v4l2_subdev_fh *fh);
    void (*release)(struct v4l2_subdev *sd);
};
```

Members

registered called when this subdev is registered. When called the `v4l2_dev` field is set to the correct `v4l2_device`.

unregistered called when this subdev is unregistered. When called the `v4l2_dev` field is still set to the correct `v4l2_device`.

open called when the subdev device node is opened by an application.

close called when the subdev device node is closed. Please note that it is possible for **close** to be called after **unregistered**!

release called when the last user of the subdev device is gone. This happens after the **unregistered** callback and when the last open filehandle to the `v4l-subdevX` device node was closed. If no device node was created for this sub-device, then the **release** callback is called right after the **unregistered** callback. The **release** callback is typically used to free the memory containing the `v4l2_subdev` structure. It is almost certainly required for any sub-device that sets the `V4L2_SUBDEV_FL_HAS_DEVNODE` flag.

Description

Note: Never call this from drivers, only the v4l2 framework can call these ops.

struct **v4l2_subdev_platform_data**
regulators config struct

Definition

```
struct v4l2_subdev_platform_data {
    struct regulator_bulk_data *regulators;
    int num_regulators;
    void *host_priv;
};
```

Members

regulators Optional regulators used to power on/off the subdevice

num_regulators Number of regulators

host_priv Per-subdevice data, specific for a certain video host device

struct **v4l2_subdev**
describes a V4L2 sub-device

Definition

```

struct v4l2_subdev {
#ifdef CONFIG_MEDIA_CONTROLLER;
    struct media_entity entity;
#endif;
    struct list_head list;
    struct module *owner;
    bool owner_v4l2_dev;
    u32 flags;
    struct v4l2_device *v4l2_dev;
    const struct v4l2_subdev_ops *ops;
    const struct v4l2_subdev_internal_ops *internal_ops;
    struct v4l2_ctrl_handler *ctrl_handler;
    char name[V4L2_SUBDEV_NAME_SIZE];
    u32 grp_id;
    void *dev_priv;
    void *host_priv;
    struct video_device *devnode;
    struct device *dev;
    struct fwnode_handle *fwnode;
    struct list_head async_list;
    struct v4l2_async_subdev *asd;
    struct v4l2_async_notifier *notifier;
    struct v4l2_async_notifier *subdev_notifier;
    struct v4l2_subdev_platform_data *pdata;
};

```

Members

entity pointer to *struct media_entity*

list List of sub-devices

owner The owner is the same as the driver's struct device owner.

owner_v4l2_dev true if the sd->owner matches the owner of **v4l2_dev->dev** owner. Initialized by *v4l2_device_register_subdev()*.

flags subdev flags. Can be: **V4L2_SUBDEV_FL_IS_I2C** - Set this flag if this subdev is a i2c device; **V4L2_SUBDEV_FL_IS_SPI** - Set this flag if this subdev is a spi device; **V4L2_SUBDEV_FL_HAS_DEVNODE** - Set this flag if this subdev needs a device node; **V4L2_SUBDEV_FL_HAS_EVENTS** - Set this flag if this subdev generates events.

v4l2_dev pointer to struct *v4l2_device*

ops pointer to struct *v4l2_subdev_ops*

internal_ops pointer to struct *v4l2_subdev_internal_ops*. Never call these internal ops from within a driver!

ctrl_handler The control handler of this subdev. May be NULL.

name Name of the sub-device. Please notice that the name must be unique.

grp_id can be used to group similar subdevs. Value is driver-specific

dev_priv pointer to private data

host_priv pointer to private data used by the device where the subdev is attached.

devnode subdev device node

dev pointer to the physical device, if any

fwnode The `fwnode_handle` of the subdev, usually the same as either `dev->of_node->fwnode` or `dev->fwnode` (whichever is non-NULL).

async_list Links this subdev to a global `subdev_list` or `notifier->done` list.

asd Pointer to respective `struct v4l2_async_subdev`.

notifier Pointer to the managing notifier.

subdev_notifier A sub-device notifier implicitly registered for the sub-device using `v4l2_async_register_subdev_sensor()`.

pdata common part of subdevice platform data

Description

Each instance of a subdev driver should create this struct, either stand-alone or embedded in a larger struct.

This structure should be initialized by `v4l2_subdev_init()` or one of its variants: `v4l2_spi_subdev_init()`, `v4l2_i2c_subdev_init()`.

media_entity_to_v4l2_subdev(ent)

Returns a `struct v4l2_subdev` from the `struct media_entity` embedded in it.

Parameters

ent pointer to `struct media_entity`.

vdev_to_v4l2_subdev(vdev)

Returns a `struct v4l2_subdev` from the `struct video_device` embedded on it.

Parameters

vdev pointer to `struct video_device`

struct **v4l2_subdev_fh**

Used for storing subdev information per file handle

Definition

```
struct v4l2_subdev_fh {
    struct v4l2_fh vfh;
    struct module *owner;
#ifdef CONFIG_VIDEO_V4L2_SUBDEV_API;
    struct v4l2_subdev_state *state;
#endif;
};
```

Members

vfh pointer to `struct v4l2_fh`

owner module pointer to the owner of this file handle

state pointer to `struct v4l2_subdev_state`

to_v4l2_subdev_fh(fh)

Returns a `struct v4l2_subdev_fh` from the `struct v4l2_fh` embedded on it.

Parameters

fh pointer to *struct v4l2_fh*

```
struct v4l2_mbus_framefmt * v4l2_subdev_get_try_format(struct
                                                    v4l2_subdev *sd,      struct
                                                    v4l2_subdev_state *state,
                                                    unsigned int pad)
    ancillary routine to call struct v4l2_subdev_pad_config->try_fmt
```

Parameters

struct v4l2_subdev *sd pointer to *struct v4l2_subdev*

struct v4l2_subdev_state *state pointer to *struct v4l2_subdev_state*

unsigned int pad index of the pad in the *struct v4l2_subdev_state->pads* array

```
struct v4l2_rect * v4l2_subdev_get_try_crop(struct      v4l2_subdev *sd,      struct
                                                    v4l2_subdev_state *state,      unsigned
                                                    int pad)
    ancillary routine to call struct v4l2_subdev_pad_config->try_crop
```

Parameters

struct v4l2_subdev *sd pointer to *struct v4l2_subdev*

struct v4l2_subdev_state *state pointer to *struct v4l2_subdev_state*.

unsigned int pad index of the pad in the *struct v4l2_subdev_state->pads* array.

```
struct v4l2_rect * v4l2_subdev_get_try_compose(struct      v4l2_subdev *sd,      struct
                                                    v4l2_subdev_state *state,      unsigned
                                                    int pad)
    ancillary routine to call struct v4l2_subdev_pad_config->try_compose
```

Parameters

struct v4l2_subdev *sd pointer to *struct v4l2_subdev*

struct v4l2_subdev_state *state pointer to *struct v4l2_subdev_state*.

unsigned int pad index of the pad in the *struct v4l2_subdev_state->pads* array.

```
void v4l2_set_subdevdata(struct v4l2_subdev *sd, void *p)
    Sets V4L2 dev private device data
```

Parameters

struct v4l2_subdev *sd pointer to *struct v4l2_subdev*

void *p pointer to the private device data to be stored.

```
void * v4l2_get_subdevdata(const struct v4l2_subdev *sd)
    Gets V4L2 dev private device data
```

Parameters

const struct v4l2_subdev *sd pointer to *struct v4l2_subdev*

Description

Returns the pointer to the private device data to be stored.

void **v4l2_set_subdev_hostdata**(struct *v4l2_subdev* *sd, void *p)
Sets V4L2 dev private host data

Parameters

struct v4l2_subdev *sd pointer to *struct v4l2_subdev*

void *p pointer to the private data to be stored.

void * **v4l2_get_subdev_hostdata**(const struct *v4l2_subdev* *sd)
Gets V4L2 dev private data

Parameters

const struct v4l2_subdev *sd pointer to *struct v4l2_subdev*

Description

Returns the pointer to the private host data to be stored.

int **v4l2_subdev_get_fwnode_pad_1_to_1**(struct *media_entity* *entity, struct *fwnode_endpoint* *endpoint)
Get pad number from a subdev fwnode endpoint, assuming 1:1 port:pad

Parameters

struct media_entity *entity Pointer to the subdev entity

struct fwnode_endpoint *endpoint Pointer to a parsed fwnode endpoint

Description

This function can be used as the .get_fwnode_pad operation for subdevices that map port numbers and pad indexes 1:1. If the endpoint is owned by the subdevice, the function returns the endpoint port number.

Returns the endpoint port number on success or a negative error code.

int **v4l2_subdev_link_validate_default**(struct *v4l2_subdev* *sd, struct *media_link* *link, struct *v4l2_subdev_format* *source_fmt, struct *v4l2_subdev_format* *sink_fmt)
validates a media link

Parameters

struct v4l2_subdev *sd pointer to *struct v4l2_subdev*

struct media_link *link pointer to *struct media_link*

struct v4l2_subdev_format *source_fmt pointer to struct *v4l2_subdev_format*

struct v4l2_subdev_format *sink_fmt pointer to struct *v4l2_subdev_format*

Description

This function ensures that width, height and the media bus pixel code are equal on both source and sink of the link.

int **v4l2_subdev_link_validate**(struct *media_link* *link)
validates a media link

Parameters

struct media_link *link pointer to *struct media_link*

Description

This function calls the subdev's `link_validate` ops to validate if a media link is valid for streaming. It also internally calls `v4l2_subdev_link_validate_default()` to ensure that width, height and the media bus pixel code are equal on both source and sink of the link.

struct v4l2_subdev_state *v4l2_subdev_alloc_state(*struct v4l2_subdev *sd*)
allocate *v4l2_subdev_state*

Parameters

struct v4l2_subdev *sd pointer to *struct v4l2_subdev* for which the state is being allocated.

Description

Must call `v4l2_subdev_free_state()` when state is no longer needed.

void v4l2_subdev_free_state(*struct v4l2_subdev_state *state*)
free a *v4l2_subdev_state*

Parameters

struct v4l2_subdev_state *state *v4l2_subdev_state* to be freed.

void v4l2_subdev_init(*struct v4l2_subdev *sd*, *const struct v4l2_subdev_ops *ops*)
initializes the sub-device struct

Parameters

struct v4l2_subdev *sd pointer to the *struct v4l2_subdev* to be initialized

const struct v4l2_subdev_ops *ops pointer to *struct v4l2_subdev_ops*.

v4l2_subdev_call(*sd, o, f, args...*)
call an operation of a *v4l2_subdev*.

Parameters

sd pointer to the *struct v4l2_subdev*

o name of the element at *struct v4l2_subdev_ops* that contains **f**. Each element there groups a set of callbacks functions.

f callback function to be called. The callback functions are defined in groups, according to each element at *struct v4l2_subdev_ops*.

args... arguments for **f**.

Example

```
err = v4l2_subdev_call(sd, video, s_std, norm);
```

v4l2_subdev_has_op(*sd, o, f*)
Checks if a subdev defines a certain operation.

Parameters

sd pointer to the *struct v4l2_subdev*

o The group of callback functions in *struct v4l2_subdev_ops* which **f** is a part of.

f callback function to be checked for its existence.

void **v4l2_subdev_notify_event**(struct *v4l2_subdev* *sd, const struct v4l2_event *ev)
Delivers event notification for subdevice

Parameters

struct v4l2_subdev *sd The subdev for which to deliver the event

const struct v4l2_event *ev The event to deliver

Description

Will deliver the specified event to all userspace event listeners which are subscribed to the v4l2 subdev event queue as well as to the bridge driver using the notify callback. The notification type for the notify callback will be `V4L2_DEVICE_NOTIFY_EVENT`.

2.2.12 V4L2 events

The V4L2 events provide a generic way to pass events to user space. The driver must use *v4l2_fh* to be able to support V4L2 events.

Events are subscribed per-filehandle. An event specification consists of a type and is optionally associated with an object identified through the `id` field. If unused, then the `id` is 0. So an event is uniquely identified by the (type, id) tuple.

The *v4l2_fh* struct has a list of subscribed events on its `subscribed` field.

When the user subscribes to an event, a *v4l2_subscribed_event* struct is added to *v4l2_fh*. `subscribed`, one for every subscribed event.

Each *v4l2_subscribed_event* struct ends with a *v4l2_kevent* ringbuffer, with the size given by the caller of *v4l2_event_subscribe()*. This ringbuffer is used to store any events raised by the driver.

So every (type, ID) event tuple will have its own *v4l2_kevent* ringbuffer. This guarantees that if a driver is generating lots of events of one type in a short time, then that will not overwrite events of another type.

But if you get more events of one type than the size of the *v4l2_kevent* ringbuffer, then the oldest event will be dropped and the new one added.

The *v4l2_kevent* struct links into the available list of the *v4l2_fh* struct so *ioctl VIDIOC_DQEVENT* will know which event to dequeue first.

Finally, if the event subscription is associated with a particular object such as a V4L2 control, then that object needs to know about that as well so that an event can be raised by that object. So the `node` field can be used to link the *v4l2_subscribed_event* struct into a list of such objects.

So to summarize:

- *struct v4l2_fh* has two lists: one of the subscribed events, and one of the available events.
- *struct v4l2_subscribed_event* has a ringbuffer of raised (pending) events of that particular type.
- If *struct v4l2_subscribed_event* is associated with a specific object, then that object will have an internal list of *struct v4l2_subscribed_event* so it knows who subscribed an event to that object.

Furthermore, the internal `struct v4l2_subscribed_event` has `merge()` and `replace()` callbacks which drivers can set. These callbacks are called when a new event is raised and there is no more room.

The `replace()` callback allows you to replace the payload of the old event with that of the new event, merging any relevant data from the old payload into the new payload that replaces it. It is called when this event type has a ringbuffer with size is one, i.e. only one event can be stored in the ringbuffer.

The `merge()` callback allows you to merge the oldest event payload into that of the second-oldest event payload. It is called when the ringbuffer has size is greater than one.

This way no status information is lost, just the intermediate steps leading up to that state.

A good example of these `replace/merge` callbacks is in `v4l2-event.c`: `ctrls_replace()` and `ctrls_merge()` callbacks for the control event.

Note: these callbacks can be called from interrupt context, so they must be fast.

In order to queue events to video device, drivers should call:

```
v4l2_event_queue(vdev, ev)
```

The driver's only responsibility is to fill in the type and the data fields. The other fields will be filled in by V4L2.

2.2.12.1 Event subscription

Subscribing to an event is via:

```
v4l2_event_subscribe(fh, sub, elems, ops)
```

This function is used to implement `video_device->ioctl_ops->vidioc_subscribe_event`, but the driver must check first if the driver is able to produce events with specified event id, and then should call `v4l2_event_subscribe()` to subscribe the event.

The `elems` argument is the size of the event queue for this event. If it is 0, then the framework will fill in a default value (this depends on the event type).

The `ops` argument allows the driver to specify a number of callbacks:

Callback	Description
add	called when a new listener gets added (subscribing to the same event twice will only cause this callback to get called once)
del	called when a listener stops listening
replace	replace event 'old' with event 'new'.
merge	merge event 'old' into event 'new'.

All 4 callbacks are optional, if you don't want to specify any callbacks the `ops` argument itself maybe NULL.

2.2.12.2 Unsubscribing an event

Unsubscribing to an event is via:

```
v4l2_event_unsubscribe (fh, sub)
```

This function is used to implement *video_device->ioctl_ops->vidioc_unsubscribe_event*. A driver may call *v4l2_event_unsubscribe()* directly unless it wants to be involved in unsubscription process.

The special type *V4L2_EVENT_ALL* may be used to unsubscribe all events. The drivers may want to handle this in a special way.

2.2.12.3 Check if there's a pending event

Checking if there's a pending event is via:

```
v4l2_event_pending (fh)
```

This function returns the number of pending events. Useful when implementing poll.

2.2.12.4 How events work

Events are delivered to user space through the poll system call. The driver can use *v4l2_fh->wait* (a *wait_queue_head_t*) as the argument for *poll_wait()*.

There are standard and private events. New standard events must use the smallest available event type. The drivers must allocate their events from their own class starting from class base. Class base is *V4L2_EVENT_PRIVATE_START + n * 1000* where *n* is the lowest available number. The first event type in the class is reserved for future use, so the first available event type is 'class base + 1'.

An example on how the V4L2 events may be used can be found in the OMAP 3 ISP driver (*drivers/media/platform/omap3isp*).

A subdev can directly send an event to the *v4l2_device* notify function with *V4L2_DEVICE_NOTIFY_EVENT*. This allows the bridge to map the subdev that sends the event to the video node(s) associated with the subdev that need to be informed about such an event.

V4L2 event functions and data structures

struct **v4l2_kevent**

Internal kernel event struct.

Definition

```
struct v4l2_kevent {
    struct list_head      list;
    struct v4l2_subscribed_event *sev;
    struct v4l2_event     event;
    u64 ts;
};
```


Members

list List node for the v4l2_fh->available list.

sev Pointer to parent v4l2_subscribed_event.

event The event itself.

ts The timestamp of the event.

struct **v4l2_subscribed_event_ops**

Subscribed event operations.

Definition

```
struct v4l2_subscribed_event_ops {
    int (*add)(struct v4l2_subscribed_event *sev, unsigned int elems);
    void (*del)(struct v4l2_subscribed_event *sev);
    void (*replace)(struct v4l2_event *old, const struct v4l2_event *new);
    void (*merge)(const struct v4l2_event *old, struct v4l2_event *new);
};
```

Members

add Optional callback, called when a new listener is added

del Optional callback, called when a listener stops listening

replace Optional callback that can replace event 'old' with event 'new'.

merge Optional callback that can merge event 'old' into event 'new'.

struct **v4l2_subscribed_event**

Internal struct representing a subscribed event.

Definition

```
struct v4l2_subscribed_event {
    struct list_head      list;
    u32 type;
    u32 id;
    u32 flags;
    struct v4l2_fh        *fh;
    struct list_head      node;
    const struct v4l2_subscribed_event_ops *ops;
    unsigned int          elems;
    unsigned int          first;
    unsigned int          in_use;
    struct v4l2_kevent    events[];
};
```

Members

list List node for the v4l2_fh->subscribed list.

type Event type.

id Associated object ID (e.g. control ID). 0 if there isn't any.

flags Copy of v4l2_event_subscription->flags.

fh Filehandle that subscribed to this event.

node List node that hooks into the object's event list (if there is one).

ops `v4l2_subscribed_event_ops`

elems The number of elements in the events array.

first The index of the events containing the oldest available event.

in_use The number of queued events.

events An array of **elems** events.

int **v4l2_event_dequeue**(struct *v4l2_fh* *fh, struct v4l2_event *event, int nonblocking)
Dequeue events from video device.

Parameters

struct v4l2_fh *fh pointer to *struct v4l2_fh*

struct v4l2_event *event pointer to struct v4l2_event

int nonblocking if not zero, waits for an event to arrive

void **v4l2_event_queue**(struct *video_device* *vdev, const struct v4l2_event *ev)
Queue events to video device.

Parameters

struct video_device *vdev pointer to *struct video_device*

const struct v4l2_event *ev pointer to struct v4l2_event

Description

The event will be queued for all *struct v4l2_fh* file handlers.

Note: The driver's only responsibility is to fill in the type and the data fields. The other fields will be filled in by V4L2.

void **v4l2_event_queue_fh**(struct *v4l2_fh* *fh, const struct v4l2_event *ev)
Queue events to video device.

Parameters

struct v4l2_fh *fh pointer to *struct v4l2_fh*

const struct v4l2_event *ev pointer to struct v4l2_event

Description

The event will be queued only for the specified *struct v4l2_fh* file handler.

Note: The driver's only responsibility is to fill in the type and the data fields. The other fields will be filled in by V4L2.

void **v4l2_event_wake_all**(struct *video_device* *vdev)
Wake all filehandles.

Parameters

struct video_device *vdev pointer to *struct video_device*

Description

Used when unregistering a video device.

```
int v4l2_event_pending(struct v4l2_fh *fh)
    Check if an event is available
```

Parameters

struct v4l2_fh *fh pointer to *struct v4l2_fh*

Description

Returns the number of pending events.

```
int v4l2_event_subscribe(struct v4l2_fh *fh, const struct v4l2_event_subscription *sub,
                        unsigned int elems, const struct
                        v4l2_subscribed_event_ops *ops)
    Subscribes to an event
```

Parameters

struct v4l2_fh *fh pointer to *struct v4l2_fh*

const struct v4l2_event_subscription *sub pointer to *struct v4l2_event_subscription*

unsigned int elems size of the events queue

const struct v4l2_subscribed_event_ops *ops pointer to *v4l2_subscribed_event_ops*

Description

Note: if **elems** is zero, the framework will fill in a default value, with is currently 1 element.

```
int v4l2_event_unsubscribe(struct v4l2_fh *fh, const struct
                          v4l2_event_subscription *sub)
    Unsubscribes to an event
```

Parameters

struct v4l2_fh *fh pointer to *struct v4l2_fh*

const struct v4l2_event_subscription *sub pointer to *struct v4l2_event_subscription*

```
void v4l2_event_unsubscribe_all(struct v4l2_fh *fh)
    Unsubscribes to all events
```

Parameters

struct v4l2_fh *fh pointer to *struct v4l2_fh*

```
int v4l2_event_subdev_unsubscribe(struct v4l2_subdev *sd, struct v4l2_fh *fh, struct
                                  v4l2_event_subscription *sub)
    Subdev variant of v4l2_event_unsubscribe()
```

Parameters

struct v4l2_subdev *sd pointer to *struct v4l2_subdev*

struct v4l2_fh *fh pointer to *struct v4l2_fh*

struct v4l2_event_subscription *sub pointer to struct v4l2_event_subscription

Description

Note: This function should be used for the *struct v4l2_subdev_core_ops* `unsubscribe_event` field.

int **v4l2_src_change_event_subscribe**(struct *v4l2_fh* *fh, const struct v4l2_event_subscription *sub)
helper function that calls *v4l2_event_subscribe()* if the event is V4L2_EVENT_SOURCE_CHANGE.

Parameters

struct v4l2_fh *fh pointer to *struct v4l2_fh*

const struct v4l2_event_subscription *sub pointer to struct v4l2_event_subscription

int **v4l2_src_change_event_subdev_subscribe**(struct *v4l2_subdev* *sd, struct *v4l2_fh* *fh, struct v4l2_event_subscription *sub)
Variant of *v4l2_event_subscribe()*, meant to subscribe only events of the type V4L2_EVENT_SOURCE_CHANGE.

Parameters

struct v4l2_subdev *sd pointer to *struct v4l2_subdev*

struct v4l2_fh *fh pointer to *struct v4l2_fh*

struct v4l2_event_subscription *sub pointer to struct v4l2_event_subscription

2.2.13 V4L2 Controls

2.2.13.1 Introduction

The V4L2 control API seems simple enough, but quickly becomes very hard to implement correctly in drivers. But much of the code needed to handle controls is actually not driver specific and can be moved to the V4L core framework.

After all, the only part that a driver developer is interested in is:

- 1) How do I add a control?
- 2) How do I set the control's value? (i.e. `s_ctrl`)

And occasionally:

- 3) How do I get the control's value? (i.e. `g_volatile_ctrl`)
- 4) How do I validate the user's proposed control value? (i.e. `try_ctrl`)

All the rest is something that can be done centrally.

The control framework was created in order to implement all the rules of the V4L2 specification with respect to controls in a central place. And to make life as easy as possible for the driver developer.

Note that the control framework relies on the presence of a struct `v4l2_device` for V4L2 drivers and `struct v4l2_subdev` for sub-device drivers.

2.2.13.2 Objects in the framework

There are two main objects:

The `v4l2_ctrl` object describes the control properties and keeps track of the control's value (both the current value and the proposed new value).

`v4l2_ctrl_handler` is the object that keeps track of controls. It maintains a list of `v4l2_ctrl` objects that it owns and another list of references to controls, possibly to controls owned by other handlers.

2.2.13.3 Basic usage for V4L2 and sub-device drivers

1) Prepare the driver:

```
#include <media/v4l2-ctrls.h>
```

1.1) Add the handler to your driver's top-level struct:

For V4L2 drivers:

```
struct foo_dev {
    ...
    struct v4l2_device v4l2_dev;
    ...
    struct v4l2_ctrl_handler ctrl_handler;
    ...
};
```

For sub-device drivers:

```
struct foo_dev {
    ...
    struct v4l2_subdev sd;
    ...
    struct v4l2_ctrl_handler ctrl_handler;
    ...
};
```

1.2) Initialize the handler:

```
v4l2_ctrl_handler_init(&foo->ctrl_handler, nr_of_controls);
```

The second argument is a hint telling the function how many controls this handler is expected to handle. It will allocate a hashtable based on this information. It is a hint only.

1.3) Hook the control handler into the driver:

For V4L2 drivers:

```
foo->v4l2_dev.ctrl_handler = &foo->ctrl_handler;
```

For sub-device drivers:

```
foo->sd.ctrl_handler = &foo->ctrl_handler;
```

1.4) Clean up the handler at the end:

```
v4l2_ctrl_handler_free(&foo->ctrl_handler);
```

2) Add controls:

You add non-menu controls by calling `v4l2_ctrl_new_std()`:

```
struct v4l2_ctrl *v4l2_ctrl_new_std(struct v4l2_ctrl_handler *hdl,  
    const struct v4l2_ctrl_ops *ops,  
    u32 id, s32 min, s32 max, u32 step, s32 def);
```

Menu and integer menu controls are added by calling `v4l2_ctrl_new_std_menu()`:

```
struct v4l2_ctrl *v4l2_ctrl_new_std_menu(struct v4l2_ctrl_handler *hdl,  
    const struct v4l2_ctrl_ops *ops,  
    u32 id, s32 max, s32 skip_mask, s32 def);
```

Menu controls with a driver specific menu are added by calling `v4l2_ctrl_new_std_menu_items()`:

```
struct v4l2_ctrl *v4l2_ctrl_new_std_menu_items(  
    struct v4l2_ctrl_handler *hdl,  
    const struct v4l2_ctrl_ops *ops, u32 id, s32 max,  
    s32 skip_mask, s32 def, const char * const *qmenu);
```

Standard compound controls can be added by calling `v4l2_ctrl_new_std_compound()`:

```
struct v4l2_ctrl *v4l2_ctrl_new_std_compound(struct v4l2_ctrl_handler *hdl,  
    const struct v4l2_ctrl_ops *ops, u32 id,  
    const union v4l2_ctrl_ptr p_def);
```

Integer menu controls with a driver specific menu can be added by calling `v4l2_ctrl_new_int_menu()`:

```
struct v4l2_ctrl *v4l2_ctrl_new_int_menu(struct v4l2_ctrl_handler *hdl,  
    const struct v4l2_ctrl_ops *ops,  
    u32 id, s32 max, s32 def, const s64 *qmenu_int);
```

These functions are typically called right after the `v4l2_ctrl_handler_init()`:

```
static const s64 exp_bias_qmenu[] = {  
    -2, -1, 0, 1, 2  
};  
static const char * const test_pattern[] = {  
    "Disabled",  
    "Vertical Bars",  
    "Solid Black",  
    "Solid White",  
};  
  
v4l2_ctrl_handler_init(&foo->ctrl_handler, nr_of_controls);  
v4l2_ctrl_new_std(&foo->ctrl_handler, &foo_ctrl_ops,  
    V4L2_CID_BRIGHTNESS, 0, 255, 1, 128);
```

```

v4l2_ctrl_new_std(&foo->ctrl_handler, &foo_ctrl_ops,
                 V4L2_CID_CONTRAST, 0, 255, 1, 128);
v4l2_ctrl_new_std_menu(&foo->ctrl_handler, &foo_ctrl_ops,
                      V4L2_CID_POWER_LINE_FREQUENCY,
                      V4L2_CID_POWER_LINE_FREQUENCY_60HZ, 0,
                      V4L2_CID_POWER_LINE_FREQUENCY_DISABLED);
v4l2_ctrl_new_int_menu(&foo->ctrl_handler, &foo_ctrl_ops,
                      V4L2_CID_EXPOSURE_BIAS,
                      ARRAY_SIZE(exp_bias_qmenu) - 1,
                      ARRAY_SIZE(exp_bias_qmenu) / 2 - 1,
                      exp_bias_qmenu);
v4l2_ctrl_new_std_menu_items(&foo->ctrl_handler, &foo_ctrl_ops,
                             V4L2_CID_TEST_PATTERN, ARRAY_SIZE(test_pattern) - 1, 0,
                             0, test_pattern);
...
if (foo->ctrl_handler.error) {
    int err = foo->ctrl_handler.error;

    v4l2_ctrl_handler_free(&foo->ctrl_handler);
    return err;
}

```

The `v4l2_ctrl_new_std()` function returns the `v4l2_ctrl` pointer to the new control, but if you do not need to access the pointer outside the control ops, then there is no need to store it.

The `v4l2_ctrl_new_std()` function will fill in most fields based on the control ID except for the min, max, step and default values. These are passed in the last four arguments. These values are driver specific while control attributes like type, name, flags are all global. The control's current value will be set to the default value.

The `v4l2_ctrl_new_std_menu()` function is very similar but it is used for menu controls. There is no min argument since that is always 0 for menu controls, and instead of a step there is a `skip_mask` argument: if bit X is 1, then menu item X is skipped.

The `v4l2_ctrl_new_int_menu()` function creates a new standard integer menu control with driver-specific items in the menu. It differs from `v4l2_ctrl_new_std_menu` in that it doesn't have the mask argument and takes as the last argument an array of signed 64-bit integers that form an exact menu item list.

The `v4l2_ctrl_new_std_menu_items()` function is very similar to `v4l2_ctrl_new_std_menu` but takes an extra parameter `qmenu`, which is the driver specific menu for an otherwise standard menu control. A good example for this control is the test pattern control for capture/display/sensors devices that have the capability to generate test patterns. These test patterns are hardware specific, so the contents of the menu will vary from device to device.

Note that if something fails, the function will return NULL or an error and set `ctrl_handler->error` to the error code. If `ctrl_handler->error` was already set, then it will just return and do nothing. This is also true for `v4l2_ctrl_handler_init` if it cannot allocate the internal data structure.

This makes it easy to init the handler and just add all controls and only check the error code at the end. Saves a lot of repetitive error checking.

It is recommended to add controls in ascending control ID order: it will be a bit faster that way.

3) Optionally force initial control setup:

```
v4l2_ctrl_handler_setup(&foo->ctrl_handler);
```

This will call `s_ctrl` for all controls unconditionally. Effectively this initializes the hardware to the default control values. It is recommended that you do this as this ensures that both the internal data structures and the hardware are in sync.

4) Finally: implement the `v4l2_ctrl_ops`

```
static const struct v4l2_ctrl_ops foo_ctrl_ops = {
    .s_ctrl = foo_s_ctrl,
};
```

Usually all you need is `s_ctrl`:

```
static int foo_s_ctrl(struct v4l2_ctrl *ctrl)
{
    struct foo *state = container_of(ctrl->handler, struct foo, ctrl_handler);

    switch (ctrl->id) {
    case V4L2_CID_BRIGHTNESS:
        write_reg(0x123, ctrl->val);
        break;
    case V4L2_CID_CONTRAST:
        write_reg(0x456, ctrl->val);
        break;
    }
    return 0;
}
```

The control ops are called with the `v4l2_ctrl` pointer as argument. The new control value has already been validated, so all you need to do is to actually update the hardware registers.

You're done! And this is sufficient for most of the drivers we have. No need to do any validation of control values, or implement `QUERYCTRL`, `QUERY_EXT_CTRL` and `QUERYMENU`. And `G/S_CTRL` as well as `G/TRY/S_EXT_CTRL`s are automatically supported.

Note: The remainder sections deal with more advanced controls topics and scenarios. In practice the basic usage as described above is sufficient for most drivers.

2.2.13.4 Inheriting Sub-device Controls

When a sub-device is registered with a V4L2 driver by calling `v4l2_device_register_subdev()` and the `ctrl_handler` fields of both `v4l2_subdev` and `v4l2_device` are set, then the controls of the subdev will become automatically available in the V4L2 driver as well. If the subdev driver contains controls that already exist in the V4L2 driver, then those will be skipped (so a V4L2 driver can always override a subdev control).

What happens here is that `v4l2_device_register_subdev()` calls `v4l2_ctrl_add_handler()` adding the controls of the subdev to the controls of `v4l2_device`.

2.2.13.5 Accessing Control Values

The following union is used inside the control framework to access control values:

```
union v4l2_ctrl_ptr {
    s32 *p_s32;
    s64 *p_s64;
    char *p_char;
    void *p;
};
```

The `v4l2_ctrl` struct contains these fields that can be used to access both current and new values:

```
s32 val;
struct {
    s32 val;
} cur;

union v4l2_ctrl_ptr p_new;
union v4l2_ctrl_ptr p_cur;
```

If the control has a simple `s32` type, then:

```
&ctrl->val == ctrl->p_new.p_s32
&ctrl->cur.val == ctrl->p_cur.p_s32
```

For all other types use `ctrl->p_cur.p<something>`. Basically the `val` and `cur.val` fields can be considered an alias since these are used so often.

Within the control ops you can freely use these. The `val` and `cur.val` speak for themselves. The `p_char` pointers point to character buffers of length `ctrl->maximum + 1`, and are always 0-terminated.

Unless the control is marked volatile the `p_cur` field points to the current cached control value. When you create a new control this value is made identical to the default value. After calling `v4l2_ctrl_handler_setup()` this value is passed to the hardware. It is generally a good idea to call this function.

Whenever a new value is set that new value is automatically cached. This means that most drivers do not need to implement the `g_volatile_ctrl()` op. The exception is for controls that return a volatile register such as a signal strength read-out that changes continuously. In that case you will need to implement `g_volatile_ctrl` like this:

```
static int foo_g_volatile_ctrl(struct v4l2_ctrl *ctrl)
{
    switch (ctrl->id) {
        case V4L2_CID_BRIGHTNESS:
            ctrl->val = read_reg(0x123);
            break;
    }
}
```

Note that you use the 'new value' union as well in `g_volatile_ctrl`. In general controls that need to implement `g_volatile_ctrl` are read-only controls. If they are not, a `V4L2_EVENT_CTRL_CH_VALUE` will not be generated when the control changes.

To mark a control as volatile you have to set `V4L2_CTRL_FLAG_VOLATILE`:

```
ctrl = v4l2_ctrl_new_std(&sd->ctrl_handler, ...);
if (ctrl)
    ctrl->flags |= V4L2_CTRL_FLAG_VOLATILE;
```

For `try/s_ctrl` the new values (i.e. as passed by the user) are filled in and you can modify them in `try_ctrl` or set them in `s_ctrl`. The 'cur' union contains the current value, which you can use (but not change!) as well.

If `s_ctrl` returns 0 (OK), then the control framework will copy the new final values to the 'cur' union.

While in `g_volatile/s/try_ctrl` you can access the value of all controls owned by the same handler since the handler's lock is held. If you need to access the value of controls owned by other handlers, then you have to be very careful not to introduce deadlocks.

Outside of the control ops you have to go through to helper functions to get or set a single control value safely in your driver:

```
s32 v4l2_ctrl_g_ctrl(struct v4l2_ctrl *ctrl);
int v4l2_ctrl_s_ctrl(struct v4l2_ctrl *ctrl, s32 val);
```

These functions go through the control framework just as `VIDIOC_G/S_CTRL` ioctls do. Don't use these inside the control ops `g_volatile/s/try_ctrl`, though, that will result in a deadlock since these helpers lock the handler as well.

You can also take the handler lock yourself:

```
mutex_lock(&state->ctrl_handler.lock);
pr_info("String value is '%s'\n", ctrl1->p_cur.p_char);
pr_info("Integer value is '%s'\n", ctrl2->cur.val);
mutex_unlock(&state->ctrl_handler.lock);
```

2.2.13.6 Menu Controls

The `v4l2_ctrl` struct contains this union:

```
union {
    u32 step;
    u32 menu_skip_mask;
};
```

For menu controls `menu_skip_mask` is used. What it does is that it allows you to easily exclude certain menu items. This is used in the `VIDIOC_QUERYMENU` implementation where you can return `-EINVAL` if a certain menu item is not present. Note that `VIDIOC_QUERYCTRL` always returns a step value of 1 for menu controls.

A good example is the MPEG Audio Layer II Bitrate menu control where the menu is a list of standardized possible bitrates. But in practice hardware implementations will only support a subset of those. By setting the skip mask you can tell the framework which menu items should be skipped. Setting it to 0 means that all menu items are supported.

You set this mask either through the `v4l2_ctrl_config` struct for a custom control, or by calling `v4l2_ctrl_new_std_menu()`.

2.2.13.7 Custom Controls

Driver specific controls can be created using `v4l2_ctrl_new_custom()`:

```
static const struct v4l2_ctrl_config ctrl_filter = {
    .ops = &ctrl_custom_ops,
    .id = V4L2_CID_MPEG_CX2341X_VIDEO_SPATIAL_FILTER,
    .name = "Spatial Filter",
    .type = V4L2_CTRL_TYPE_INTEGER,
    .flags = V4L2_CTRL_FLAG_SLIDER,
    .max = 15,
    .step = 1,
};

ctrl = v4l2_ctrl_new_custom(&foo->ctrl_handler, &ctrl_filter, NULL);
```

The last argument is the priv pointer which can be set to driver-specific private data.

The `v4l2_ctrl_config` struct also has a field to set the `is_private` flag.

If the name field is not set, then the framework will assume this is a standard control and will fill in the name, type and flags fields accordingly.

2.2.13.8 Active and Grabbed Controls

If you get more complex relationships between controls, then you may have to activate and deactivate controls. For example, if the Chroma AGC control is on, then the Chroma Gain control is inactive. That is, you may set it, but the value will not be used by the hardware as long as the automatic gain control is on. Typically user interfaces can disable such input fields.

You can set the 'active' status using `v4l2_ctrl_activate()`. By default all controls are active. Note that the framework does not check for this flag. It is meant purely for GUIs. The function is typically called from within `s_ctrl`.

The other flag is the 'grabbed' flag. A grabbed control means that you cannot change it because it is in use by some resource. Typical examples are MPEG bitrate controls that cannot be changed while capturing is in progress.

If a control is set to 'grabbed' using `v4l2_ctrl_grab()`, then the framework will return `-EBUSY` if an attempt is made to set this control. The `v4l2_ctrl_grab()` function is typically called from the driver when it starts or stops streaming.

2.2.13.9 Control Clusters

By default all controls are independent from the others. But in more complex scenarios you can get dependencies from one control to another. In that case you need to 'cluster' them:

```
struct foo {
    struct v4l2_ctrl_handler ctrl_handler;
#define AUDIO_CL_VOLUME (0)
#define AUDIO_CL_MUTE (1)
    struct v4l2_ctrl *audio_cluster[2];
    ...
};
```

```
state->audio_cluster[AUDIO_CL_VOLUME] =
    v4l2_ctrl_new_std(&state->ctrl_handler, ...);
state->audio_cluster[AUDIO_CL_MUTE] =
    v4l2_ctrl_new_std(&state->ctrl_handler, ...);
v4l2_ctrl_cluster(ARRAY_SIZE(state->audio_cluster), state->audio_cluster);
```

From now on whenever one or more of the controls belonging to the same cluster is set (or 'gotten', or 'tried'), only the control ops of the first control ('volume' in this example) is called. You effectively create a new composite control. Similar to how a 'struct' works in C.

So when `s_ctrl` is called with `V4L2_CID_AUDIO_VOLUME` as argument, you should set all two controls belonging to the `audio_cluster`:

```
static int foo_s_ctrl(struct v4l2_ctrl *ctrl)
{
    struct foo *state = container_of(ctrl->handler, struct foo, ctrl_handler);

    switch (ctrl->id) {
    case V4L2_CID_AUDIO_VOLUME: {
        struct v4l2_ctrl *mute = ctrl->cluster[AUDIO_CL_MUTE];

        write_reg(0x123, mute->val ? 0 : ctrl->val);
        break;
    }
    case V4L2_CID_CONTRAST:
        write_reg(0x456, ctrl->val);
        break;
    }
    return 0;
}
```

In the example above the following are equivalent for the `VOLUME` case:

```
ctrl == ctrl->cluster[AUDIO_CL_VOLUME] == state->audio_cluster[AUDIO_CL_VOLUME]
ctrl->cluster[AUDIO_CL_MUTE] == state->audio_cluster[AUDIO_CL_MUTE]
```

In practice using cluster arrays like this becomes very tiresome. So instead the following equivalent method is used:

```
struct {
    /* audio cluster */
    struct v4l2_ctrl *volume;
    struct v4l2_ctrl *mute;
};
```

The anonymous struct is used to clearly 'cluster' these two control pointers, but it serves no other purpose. The effect is the same as creating an array with two control pointers. So you can just do:

```
state->volume = v4l2_ctrl_new_std(&state->ctrl_handler, ...);
state->mute = v4l2_ctrl_new_std(&state->ctrl_handler, ...);
v4l2_ctrl_cluster(2, &state->volume);
```

And in `foo_s_ctrl` you can use these pointers directly: `state->mute->val`.

Note that controls in a cluster may be `NULL`. For example, if for some reason `mute` was never added (because the hardware doesn't support that particular feature), then `mute` will be `NULL`.

So in that case we have a cluster of 2 controls, of which only 1 is actually instantiated. The only restriction is that the first control of the cluster must always be present, since that is the ‘master’ control of the cluster. The master control is the one that identifies the cluster and that provides the pointer to the `v4l2_ctrl_ops` struct that is used for that cluster.

Obviously, all controls in the cluster array must be initialized to either a valid control or to `NULL`.

In rare cases you might want to know which controls of a cluster actually were set explicitly by the user. For this you can check the ‘is_new’ flag of each control. For example, in the case of a volume/mute cluster the ‘is_new’ flag of the mute control would be set if the user called `VIDIOC_S_CTRL` for mute only. If the user would call `VIDIOC_S_EXT_CTRLS` for both mute and volume controls, then the ‘is_new’ flag would be 1 for both controls.

The ‘is_new’ flag is always 1 when called from `v4l2_ctrl_handler_setup()`.

2.2.13.10 Handling autogain/gain-type Controls with Auto Clusters

A common type of control cluster is one that handles ‘auto-foo/foo’-type controls. Typical examples are autogain/gain, autoexposure/exposure, autowhitebalance/red balance/blue balance. In all cases you have one control that determines whether another control is handled automatically by the hardware, or whether it is under manual control from the user.

If the cluster is in automatic mode, then the manual controls should be marked inactive and volatile. When the volatile controls are read the `g_volatile_ctrl` operation should return the value that the hardware’s automatic mode set up automatically.

If the cluster is put in manual mode, then the manual controls should become active again and the volatile flag is cleared (so `g_volatile_ctrl` is no longer called while in manual mode). In addition just before switching to manual mode the current values as determined by the auto mode are copied as the new manual values.

Finally the `V4L2_CTRL_FLAG_UPDATE` should be set for the auto control since changing that control affects the control flags of the manual controls.

In order to simplify this a special variation of `v4l2_ctrl_cluster` was introduced:

```
void v4l2_ctrl_auto_cluster(unsigned ncontrols, struct v4l2_ctrl **controls,
                           u8 manual_val, bool set_volatile);
```

The first two arguments are identical to `v4l2_ctrl_cluster`. The third argument tells the framework which value switches the cluster into manual mode. The last argument will optionally set `V4L2_CTRL_FLAG_VOLATILE` for the non-auto controls. If it is false, then the manual controls are never volatile. You would typically use that if the hardware does not give you the option to read back to values as determined by the auto mode (e.g. if autogain is on, the hardware doesn’t allow you to obtain the current gain value).

The first control of the cluster is assumed to be the ‘auto’ control.

Using this function will ensure that you don’t need to handle all the complex flag and volatile handling.

2.2.13.11 VIDIOC_LOG_STATUS Support

This ioctl allow you to dump the current status of a driver to the kernel log. The `v4l2_ctrl_handler_log_status(ctrl_handler, prefix)` can be used to dump the value of the controls owned by the given handler to the log. You can supply a prefix as well. If the prefix didn't end with a space, then `' '` will be added for you.

2.2.13.12 Different Handlers for Different Video Nodes

Usually the V4L2 driver has just one control handler that is global for all video nodes. But you can also specify different control handlers for different video nodes. You can do that by manually setting the `ctrl_handler` field of `struct video_device`.

That is no problem if there are no subdevs involved but if there are, then you need to block the automatic merging of subdev controls to the global control handler. You do that by simply setting the `ctrl_handler` field in `struct v4l2_device` to `NULL`. Now `v4l2_device_register_subdev()` will no longer merge subdev controls.

After each subdev was added, you will then have to call `v4l2_ctrl_add_handler` manually to add the subdev's control handler (`sd->ctrl_handler`) to the desired control handler. This control handler may be specific to the `video_device` or for a subset of `video_device`'s. For example: the radio device nodes only have audio controls, while the video and vbi device nodes share the same control handler for the audio and video controls.

If you want to have one handler (e.g. for a radio device node) have a subset of another handler (e.g. for a video device node), then you should first add the controls to the first handler, add the other controls to the second handler and finally add the first handler to the second. For example:

```
v4l2_ctrl_new_std(&radio_ctrl_handler, &radio_ops, V4L2_CID_AUDIO_VOLUME, ...);
v4l2_ctrl_new_std(&radio_ctrl_handler, &radio_ops, V4L2_CID_AUDIO_MUTE, ...);
v4l2_ctrl_new_std(&video_ctrl_handler, &video_ops, V4L2_CID_BRIGHTNESS, ...);
v4l2_ctrl_new_std(&video_ctrl_handler, &video_ops, V4L2_CID_CONTRAST, ...);
v4l2_ctrl_add_handler(&video_ctrl_handler, &radio_ctrl_handler, NULL);
```

The last argument to `v4l2_ctrl_add_handler()` is a filter function that allows you to filter which controls will be added. Set it to `NULL` if you want to add all controls.

Or you can add specific controls to a handler:

```
volume = v4l2_ctrl_new_std(&video_ctrl_handler, &ops, V4L2_CID_AUDIO_VOLUME, ...);
v4l2_ctrl_new_std(&video_ctrl_handler, &ops, V4L2_CID_BRIGHTNESS, ...);
v4l2_ctrl_new_std(&video_ctrl_handler, &ops, V4L2_CID_CONTRAST, ...);
```

What you should not do is make two identical controls for two handlers. For example:

```
v4l2_ctrl_new_std(&radio_ctrl_handler, &radio_ops, V4L2_CID_AUDIO_MUTE, ...);
v4l2_ctrl_new_std(&video_ctrl_handler, &video_ops, V4L2_CID_AUDIO_MUTE, ...);
```

This would be bad since muting the radio would not change the video mute control. The rule is to have one control for each hardware 'knob' that you can twiddle.

2.2.13.13 Finding Controls

Normally you have created the controls yourself and you can store the `struct v4l2_ctrl` pointer into your own struct.

But sometimes you need to find a control from another handler that you do not own. For example, if you have to find a volume control from a subdev.

You can do that by calling `v4l2_ctrl_find`:

```
struct v4l2_ctrl *volume;

volume = v4l2_ctrl_find(sd->ctrl_handler, V4L2_CID_AUDIO_VOLUME);
```

Since `v4l2_ctrl_find` will lock the handler you have to be careful where you use it. For example, this is not a good idea:

```
struct v4l2_ctrl_handler ctrl_handler;

v4l2_ctrl_new_std(&ctrl_handler, &video_ops, V4L2_CID_BRIGHTNESS, ...);
v4l2_ctrl_new_std(&ctrl_handler, &video_ops, V4L2_CID_CONTRAST, ...);
```

...and in `video_ops.s_ctrl`:

```
case V4L2_CID_BRIGHTNESS:
    contrast = v4l2_find_ctrl(&ctrl_handler, V4L2_CID_CONTRAST);
    ...
```

When `s_ctrl` is called by the framework the `ctrl_handler.lock` is already taken, so attempting to find another control from the same handler will deadlock.

It is recommended not to use this function from inside the control ops.

2.2.13.14 Preventing Controls inheritance

When one control handler is added to another using `v4l2_ctrl_add_handler`, then by default all controls from one are merged to the other. But a subdev might have low-level controls that make sense for some advanced embedded system, but not when it is used in consumer-level hardware. In that case you want to keep those low-level controls local to the subdev. You can do this by simply setting the 'is_private' flag of the control to 1:

```
static const struct v4l2_ctrl_config ctrl_private = {
    .ops = &ctrl_custom_ops,
    .id = V4L2_CID_...,
    .name = "Some Private Control",
    .type = V4L2_CTRL_TYPE_INTEGER,
    .max = 15,
    .step = 1,
    .is_private = 1,
};

ctrl = v4l2_ctrl_new_custom(&foo->ctrl_handler, &ctrl_private, NULL);
```

These controls will now be skipped when `v4l2_ctrl_add_handler` is called.

2.2.13.15 V4L2_CTRL_TYPE_CTRL_CLASS Controls

Controls of this type can be used by GUIs to get the name of the control class. A fully featured GUI can make a dialog with multiple tabs with each tab containing the controls belonging to a particular control class. The name of each tab can be found by querying a special control with ID <control class | 1>.

Drivers do not have to care about this. The framework will automatically add a control of this type whenever the first control belonging to a new control class is added.

2.2.13.16 Adding Notify Callbacks

Sometimes the platform or bridge driver needs to be notified when a control from a sub-device driver changes. You can set a notify callback by calling this function:

```
void v4l2_ctrl_notify(struct v4l2_ctrl *ctrl,
                    void (*notify)(struct v4l2_ctrl *ctrl, void *priv), void *priv);
```

Whenever the give control changes value the notify callback will be called with a pointer to the control and the priv pointer that was passed with v4l2_ctrl_notify. Note that the control's handler lock is held when the notify function is called.

There can be only one notify function per control handler. Any attempt to set another notify function will cause a WARN_ON.

2.2.13.17 v4l2_ctrl functions and data structures

union v4l2_ctrl_ptr

A pointer to a control value.

Definition

```
union v4l2_ctrl_ptr {
    s32 *p_s32;
    s64 *p_s64;
    u8 *p_u8;
    u16 *p_u16;
    u32 *p_u32;
    char *p_char;
    struct v4l2_ctrl_mpeg2_sequence *p_mpeg2_sequence;
    struct v4l2_ctrl_mpeg2_picture *p_mpeg2_picture;
    struct v4l2_ctrl_mpeg2_quantisation *p_mpeg2_quantisation;
    struct v4l2_ctrl_fwht_params *p_fwht_params;
    struct v4l2_ctrl_h264_sps *p_h264_sps;
    struct v4l2_ctrl_h264_pps *p_h264_pps;
    struct v4l2_ctrl_h264_scaling_matrix *p_h264_scaling_matrix;
    struct v4l2_ctrl_h264_slice_params *p_h264_slice_params;
    struct v4l2_ctrl_h264_decode_params *p_h264_decode_params;
    struct v4l2_ctrl_h264_pred_weights *p_h264_pred_weights;
    struct v4l2_ctrl_vp8_frame *p_vp8_frame;
    struct v4l2_ctrl_hevc_sps *p_hevc_sps;
    struct v4l2_ctrl_hevc_pps *p_hevc_pps;
    struct v4l2_ctrl_hevc_slice_params *p_hevc_slice_params;
    struct v4l2_ctrl_vp9_compressed_hdr *p_vp9_compressed_hdr_probs;
    struct v4l2_ctrl_vp9_frame *p_vp9_frame;
```



```

struct v4l2_ctrl_hdr10_cll_info *p_hdr10_cll;
struct v4l2_ctrl_hdr10_mastering_display *p_hdr10_mastering;
struct v4l2_area *p_area;
void *p;
const void *p_const;
};

```

Members

p_s32 Pointer to a 32-bit signed value.

p_s64 Pointer to a 64-bit signed value.

p_u8 Pointer to a 8-bit unsigned value.

p_u16 Pointer to a 16-bit unsigned value.

p_u32 Pointer to a 32-bit unsigned value.

p_char Pointer to a string.

p_mpeg2_sequence Pointer to a MPEG2 sequence structure.

p_mpeg2_picture Pointer to a MPEG2 picture structure.

p_mpeg2_quantisation Pointer to a MPEG2 quantisation data structure.

p_fwht_params Pointer to a FWHT stateless parameters structure.

p_h264_sps Pointer to a *struct v4l2_ctrl_h264_sps*.

p_h264_pps Pointer to a *struct v4l2_ctrl_h264_pps*.

p_h264_scaling_matrix Pointer to a *struct v4l2_ctrl_h264_scaling_matrix*.

p_h264_slice_params Pointer to a *struct v4l2_ctrl_h264_slice_params*.

p_h264_decode_params Pointer to a *struct v4l2_ctrl_h264_decode_params*.

p_h264_pred_weights Pointer to a *struct v4l2_ctrl_h264_pred_weights*.

p_vp8_frame Pointer to a VP8 frame params structure.

p_hevc_sps Pointer to an HEVC sequence parameter set structure.

p_hevc_pps Pointer to an HEVC picture parameter set structure.

p_hevc_slice_params Pointer to an HEVC slice parameters structure.

p_vp9_compressed_hdr_probs Pointer to a VP9 frame compressed header probs structure.

p_vp9_frame Pointer to a VP9 frame params structure.

p_hdr10_cll Pointer to an HDR10 Content Light Level structure.

p_hdr10_mastering Pointer to an HDR10 Mastering Display structure.

p_area Pointer to an area.

p Pointer to a compound value.

p_const Pointer to a constant compound value.

union *v4l2_ctrl_ptr* **v4l2_ctrl_ptr_create**(void *ptr)

Helper function to return a *v4l2_ctrl_ptr* from a void pointer

Parameters

void *ptr The void pointer

struct **v4l2_ctrl_ops**

The control operations that the driver has to provide.

Definition

```
struct v4l2_ctrl_ops {
    int (*g_volatile_ctrl)(struct v4l2_ctrl *ctrl);
    int (*try_ctrl)(struct v4l2_ctrl *ctrl);
    int (*s_ctrl)(struct v4l2_ctrl *ctrl);
};
```

Members

g_volatile_ctrl Get a new value for this control. Generally only relevant for volatile (and usually read-only) controls such as a control that returns the current signal strength which changes continuously. If not set, then the currently cached value will be returned.

try_ctrl Test whether the control's value is valid. Only relevant when the usual min/max/step checks are not sufficient.

s_ctrl Actually set the new control value. **s_ctrl** is compulsory. The `ctrl->handler->lock` is held when these ops are called, so no one else can access controls owned by that handler.

struct **v4l2_ctrl_type_ops**

The control type operations that the driver has to provide.

Definition

```
struct v4l2_ctrl_type_ops {
    bool (*equal)(const struct v4l2_ctrl *ctrl, u32 idx, union v4l2_ctrl_ptr ptr1, union_
↪v4l2_ctrl_ptr ptr2);
    void (*init)(const struct v4l2_ctrl *ctrl, u32 idx, union v4l2_ctrl_ptr ptr);
    void (*log)(const struct v4l2_ctrl *ctrl);
    int (*validate)(const struct v4l2_ctrl *ctrl, u32 idx, union v4l2_ctrl_ptr ptr);
};
```

Members

equal return true if both values are equal.

init initialize the value.

log log the value.

validate validate the value. Return 0 on success and a negative value otherwise.

v4l2_ctrl_notify_fnc

Typedef: typedef for a notify argument with a function that should be called when a control value has changed.

Syntax

```
void v4l2_ctrl_notify_fnc (struct v4l2_ctrl *ctrl, void *priv)
```

Parameters

struct **v4l2_ctrl *ctrl** pointer to struct *v4l2_ctrl*

void *priv control private data

Description

This typedef definition is used as an argument to *v4l2_ctrl_notify()* and as an argument at struct *v4l2_ctrl_handler*.

struct **v4l2_ctrl**

The control structure.

Definition

```

struct v4l2_ctrl {
    struct list_head node;
    struct list_head ev_subs;
    struct v4l2_ctrl_handler *handler;
    struct v4l2_ctrl **cluster;
    unsigned int ncontrols;
    unsigned int done:1;
    unsigned int is_new:1;
    unsigned int has_changed:1;
    unsigned int is_private:1;
    unsigned int is_auto:1;
    unsigned int is_int:1;
    unsigned int is_string:1;
    unsigned int is_ptr:1;
    unsigned int is_array:1;
    unsigned int has_volatiles:1;
    unsigned int call_notify:1;
    unsigned int manual_mode_value:8;
    const struct v4l2_ctrl_ops *ops;
    const struct v4l2_ctrl_type_ops *type_ops;
    u32 id;
    const char *name;
    enum v4l2_ctrl_type type;
    s64 minimum, maximum, default_value;
    u32 elems;
    u32 elem_size;
    u32 dims[V4L2_CTRL_MAX_DIMS];
    u32 nr_of_dims;
    union {
        u64 step;
        u64 menu_skip_mask;
    };
    union {
        const char * const *qmenu;
        const s64 *qmenu_int;
    };
    unsigned long flags;
    void *priv;
    s32 val;
    struct {
        s32 val;
    } cur;
    union v4l2_ctrl_ptr p_def;
    union v4l2_ctrl_ptr p_new;
    union v4l2_ctrl_ptr p_cur;
};

```

Members

node The list node.

ev_subs The list of control event subscriptions.

handler The handler that owns the control.

cluster Point to start of cluster array.

ncontrols Number of controls in cluster array.

done Internal flag: set for each processed control.

is_new Set when the user specified a new value for this control. It is also set when called from `v4l2_ctrl_handler_setup()`. Drivers should never set this flag.

has_changed Set when the current value differs from the new value. Drivers should never use this flag.

is_private If set, then this control is private to its handler and it will not be added to any other handlers. Drivers can set this flag.

is_auto If set, then this control selects whether the other cluster members are in 'automatic' mode or 'manual' mode. This is used for autogain/gain type clusters. Drivers should never set this flag directly.

is_int If set, then this control has a simple integer value (i.e. it uses `ctrl->val`).

is_string If set, then this control has type `V4L2_CTRL_TYPE_STRING`.

is_ptr If set, then this control is an array and/or has type `>= V4L2_CTRL_COMPOUND_TYPES` and/or has type `V4L2_CTRL_TYPE_STRING`. In other words, `struct v4l2_ext_control` uses field `p` to point to the data.

is_array If set, then this control contains an N-dimensional array.

has_volatiles If set, then one or more members of the cluster are volatile. Drivers should never touch this flag.

call_notify If set, then call the handler's notify function whenever the control's value changes.

manual_mode_value If the `is_auto` flag is set, then this is the value of the auto control that determines if that control is in manual mode. So if the value of the auto control equals this value, then the whole cluster is in manual mode. Drivers should never set this flag directly.

ops The control ops.

type_ops The control type ops.

id The control ID.

name The control name.

type The control type.

minimum The control's minimum value.

maximum The control's maximum value.

default_value The control's default value.

elems The number of elements in the N-dimensional array.

elem_size The size in bytes of the control.

dims The size of each dimension.

nr_of_dims The number of dimensions in **dims**.

{unnamed_union} anonymous

step The control's step value for non-menu controls.

menu_skip_mask The control's skip mask for menu controls. This makes it easy to skip menu items that are not valid. If bit X is set, then menu item X is skipped. Of course, this only works for menus with ≤ 32 menu items. There are no menus that come close to that number, so this is OK. Should we ever need more, then this will have to be extended to a u64 or a bit array.

{unnamed_union} anonymous

qmenu A const char * array for all menu items. Array entries that are empty strings ("") correspond to non-existing menu items (this is in addition to the menu_skip_mask above). The last entry must be NULL. Used only if the **type** is V4L2_CTRL_TYPE_MENU.

qmenu_int A 64-bit integer array for with integer menu items. The size of array must be equal to the menu size, e. g.: $\text{ceil}(\frac{\text{maximum}-\text{minimum}}{\text{step}}) + 1$. Used only if the **type** is V4L2_CTRL_TYPE_INTEGER_MENU.

flags The control's flags.

priv The control's private pointer. For use by the driver. It is untouched by the control framework. Note that this pointer is not freed when the control is deleted. Should this be needed then a new internal bitfield can be added to tell the framework to free this pointer.

val The control's new s32 value.

cur Structure to store the current value.

cur.val The control's current value, if the **type** is represented via a u32 integer (see enum v4l2_ctrl_type).

p_def The control's default value represented via a union which provides a standard way of accessing control types through a pointer (for compound controls only).

p_new The control's new value represented via a union which provides a standard way of accessing control types through a pointer.

p_cur The control's current value represented via a union which provides a standard way of accessing control types through a pointer.

struct **v4l2_ctrl_ref**

The control reference.

Definition

```
struct v4l2_ctrl_ref {
    struct list_head node;
    struct v4l2_ctrl_ref *next;
    struct v4l2_ctrl *ctrl;
    struct v4l2_ctrl_helper *helper;
    bool from_other_dev;
    bool req_done;
}
```

```
bool valid_p_req;
union v4l2_ctrl_ptr p_req;
};
```

Members

node List node for the sorted list.

next Single-link list node for the hash.

ctrl The actual control information.

helper Pointer to helper struct. Used internally in `prepare_ext_ctrls` function at `v4l2-ctrl.c`.

from_other_dev If true, then **ctrl** was defined in another device than the *struct v4l2_ctrl_handler*.

req_done Internal flag: if the control handler containing this control reference is bound to a media request, then this is set when the control has been applied. This prevents applying controls from a cluster with multiple controls twice (when the first control of a cluster is applied, they all are).

valid_p_req If set, then `p_req` contains the control value for the request.

p_req If the control handler containing this control reference is bound to a media request, then this points to the value of the control that must be applied when the request is executed, or to the value of the control at the time that the request was completed. If **valid_p_req** is false, then this control was never set for this request and the control will not be updated when this request is applied.

Description

Each control handler has a list of these refs. The `list_head` is used to keep a sorted-by-control-ID list of all controls, while the next pointer is used to link the control in the hash's bucket.

struct `v4l2_ctrl_handler`

The control handler keeps track of all the controls: both the controls owned by the handler and those inherited from other handlers.

Definition

```
struct v4l2_ctrl_handler {
    struct mutex _lock;
    struct mutex *lock;
    struct list_head ctrls;
    struct list_head ctrl_refs;
    struct v4l2_ctrl_ref *cached;
    struct v4l2_ctrl_ref **buckets;
    v4l2_ctrl_notify_fnc notify;
    void *notify_priv;
    u16 nr_of_buckets;
    int error;
    bool request_is_queued;
    struct list_head requests;
    struct list_head requests_queued;
    struct media_request_object req_obj;
};
```

Members

_lock Default for “lock”.

lock Lock to control access to this handler and its controls. May be replaced by the user right after init.

ctrls The list of controls owned by this handler.

ctrl_refs The list of control references.

cached The last found control reference. It is common that the same control is needed multiple times, so this is a simple optimization.

buckets Buckets for the hashing. Allows for quick control lookup.

notify A notify callback that is called whenever the control changes value. Note that the handler’s lock is held when the notify function is called!

notify_priv Passed as argument to the v4l2_ctrl notify callback.

nr_of_buckets Total number of buckets in the array.

error The error code of the first failed control addition.

request_is_queued True if the request was queued.

requests List to keep track of open control handler request objects. For the parent control handler (**req_obj.ops** == NULL) this is the list header. When the parent control handler is removed, it has to unbind and put all these requests since they refer to the parent.

requests_queued List of the queued requests. This determines the order in which these controls are applied. Once the request is completed it is removed from this list.

req_obj The *struct media_request_object*, used to link into a *struct media_request*. This request object has a refcount.

struct **v4l2_ctrl_config**
Control configuration structure.

Definition

```

struct v4l2_ctrl_config {
    const struct v4l2_ctrl_ops *ops;
    const struct v4l2_ctrl_type_ops *type_ops;
    u32 id;
    const char *name;
    enum v4l2_ctrl_type type;
    s64 min;
    s64 max;
    u64 step;
    s64 def;
    union v4l2_ctrl_ptr p_def;
    u32 dims[V4L2_CTRL_MAX_DIMS];
    u32 elem_size;
    u32 flags;
    u64 menu_skip_mask;
    const char * const *qmenu;
    const s64 *qmenu_int;
    unsigned int is_private:1;
};

```

Members

ops The control ops.

type_ops The control type ops. Only needed for compound controls.

id The control ID.

name The control name.

type The control type.

min The control's minimum value.

max The control's maximum value.

step The control's step value for non-menu controls.

def The control's default value.

p_def The control's default value for compound controls.

dims The size of each dimension.

elem_size The size in bytes of the control.

flags The control's flags.

menu_skip_mask The control's skip mask for menu controls. This makes it easy to skip menu items that are not valid. If bit X is set, then menu item X is skipped. Of course, this only works for menus with ≤ 64 menu items. There are no menus that come close to that number, so this is OK. Should we ever need more, then this will have to be extended to a bit array.

qmenu A const char * array for all menu items. Array entries that are empty strings ("") correspond to non-existing menu items (this is in addition to the menu_skip_mask above). The last entry must be NULL.

qmenu_int A const s64 integer array for all menu items of the type V4L2_CTRL_TYPE_INTEGER_MENU.

is_private If set, then this control is private to its handler and it will not be added to any other handlers.

void **v4l2_ctrl_fill**(u32 *id*, const char ***name*, enum v4l2_ctrl_type **type*, s64 **min*, s64 **max*, u64 **step*, s64 **def*, u32 **flags*)
Fill in the control fields based on the control ID.

Parameters

u32 id ID of the control

const char **name pointer to be filled with a string with the name of the control

enum v4l2_ctrl_type *type pointer for storing the type of the control

s64 *min pointer for storing the minimum value for the control

s64 *max pointer for storing the maximum value for the control

u64 *step pointer for storing the control step

s64 *def pointer for storing the default value for the control

u32 *flags pointer for storing the flags to be used on the control

Description

This works for all standard V4L2 controls. For non-standard controls it will only fill in the given arguments and **name** content will be set to NULL.

This function will overwrite the contents of **name**, **type** and **flags**. The contents of **min**, **max**, **step** and **def** may be modified depending on the type.

Note: Do not use in drivers! It is used internally for backwards compatibility control handling only. Once all drivers are converted to use the new control framework this function will no longer be exported.

```
int v4l2_ctrl_handler_init_class(struct v4l2_ctrl_handler *hdl, unsigned
                               int nr_of_controls_hint, struct lock_class_key *key,
                               const char *name)
```

Initialize the control handler.

Parameters

struct v4l2_ctrl_handler *hdl The control handler.

unsigned int nr_of_controls_hint A hint of how many controls this handler is expected to refer to. This is the total number, so including any inherited controls. It doesn't have to be precise, but if it is way off, then you either waste memory (too many buckets are allocated) or the control lookup becomes slower (not enough buckets are allocated, so there are more slow list lookups). It will always work, though.

struct lock_class_key *key Used by the lock validator if CONFIG_LOCKDEP is set.

const char *name Used by the lock validator if CONFIG_LOCKDEP is set.

Description

Attention: Never use this call directly, always use the `v4l2_ctrl_handler_init()` macro that hides the **key** and **name** arguments.

Return

returns an error if the buckets could not be allocated. This error will also be stored in **hdl->error**.

```
v4l2_ctrl_handler_init(hdl, nr_of_controls_hint)
    helper function to create a static struct lock_class_key and calls
    v4l2_ctrl_handler_init_class()
```

Parameters

hdl The control handler.

nr_of_controls_hint A hint of how many controls this handler is expected to refer to. This is the total number, so including any inherited controls. It doesn't have to be precise, but if it is way off, then you either waste memory (too many buckets are allocated) or the control lookup becomes slower (not enough buckets are allocated, so there are more slow list lookups). It will always work, though.

Description

This helper function creates a static struct `lock_class_key` and calls `v4l2_ctrl_handler_init_class()`, providing a proper name for the lock validator.

Use this helper function to initialize a control handler.

```
void v4l2_ctrl_handler_free(struct v4l2_ctrl_handler *hdl)
```

Free all controls owned by the handler and free the control list.

Parameters

struct v4l2_ctrl_handler *hdl The control handler.

Description

Does nothing if **hdl** == NULL.

```
void v4l2_ctrl_lock(struct v4l2_ctrl *ctrl)
```

Helper function to lock the handler associated with the control.

Parameters

struct v4l2_ctrl *ctrl The control to lock.

```
void v4l2_ctrl_unlock(struct v4l2_ctrl *ctrl)
```

Helper function to unlock the handler associated with the control.

Parameters

struct v4l2_ctrl *ctrl The control to unlock.

```
int __v4l2_ctrl_handler_setup(struct v4l2_ctrl_handler *hdl)
```

Call the `s_ctrl` op for all controls belonging to the handler to initialize the hardware to the current control values. The caller is responsible for acquiring the control handler mutex on behalf of `__v4l2_ctrl_handler_setup()`.

Parameters

struct v4l2_ctrl_handler *hdl The control handler.

Description

Button controls will be skipped, as are read-only controls.

If **hdl** == NULL, then this just returns 0.

```
int v4l2_ctrl_handler_setup(struct v4l2_ctrl_handler *hdl)
```

Call the `s_ctrl` op for all controls belonging to the handler to initialize the hardware to the current control values.

Parameters

struct v4l2_ctrl_handler *hdl The control handler.

Description

Button controls will be skipped, as are read-only controls.

If **hdl** == NULL, then this just returns 0.

```
void v4l2_ctrl_handler_log_status(struct v4l2_ctrl_handler *hdl, const char *prefix)
```

Log all controls owned by the handler.

Parameters

struct v4l2_ctrl_handler *hdl The control handler.

const char *prefix The prefix to use when logging the control values. If the prefix does not end with a space, then “: ” will be added after the prefix. If **prefix == NULL**, then no prefix will be used.

Description

For use with VIDIOC_LOG_STATUS.

Does nothing if **hdl == NULL**.

struct *v4l2_ctrl* * **v4l2_ctrl_new_custom**(struct *v4l2_ctrl_handler* *hdl, const struct *v4l2_ctrl_config* *cfg, void *priv)
Allocate and initialize a new custom V4L2 control.

Parameters

struct v4l2_ctrl_handler *hdl The control handler.

const struct v4l2_ctrl_config *cfg The control’s configuration data.

void *priv The control’s driver-specific private data.

Description

If the *v4l2_ctrl* struct could not be allocated then NULL is returned and **hdl->error** is set to the error code (if it wasn’t set already).

struct *v4l2_ctrl* * **v4l2_ctrl_new_std**(struct *v4l2_ctrl_handler* *hdl, const struct *v4l2_ctrl_ops* *ops, u32 id, s64 min, s64 max, u64 step, s64 def)
Allocate and initialize a new standard V4L2 non-menu control.

Parameters

struct v4l2_ctrl_handler *hdl The control handler.

const struct v4l2_ctrl_ops *ops The control ops.

u32 id The control ID.

s64 min The control’s minimum value.

s64 max The control’s maximum value.

u64 step The control’s step value

s64 def The control’s default value.

Description

If the *v4l2_ctrl* struct could not be allocated, or the control ID is not known, then NULL is returned and **hdl->error** is set to the appropriate error code (if it wasn’t set already).

If **id** refers to a menu control, then this function will return NULL.

Use *v4l2_ctrl_new_std_menu()* when adding menu controls.

struct *v4l2_ctrl* * **v4l2_ctrl_new_std_menu**(struct *v4l2_ctrl_handler* *hdl, const struct *v4l2_ctrl_ops* *ops, u32 id, u8 max, u64 mask, u8 def)
Allocate and initialize a new standard V4L2 menu control.

Parameters

struct v4l2_ctrl_handler *hdl The control handler.

const struct v4l2_ctrl_ops *ops The control ops.

u32 id The control ID.

u8 max The control's maximum value.

u64 mask The control's skip mask for menu controls. This makes it easy to skip menu items that are not valid. If bit X is set, then menu item X is skipped. Of course, this only works for menus with ≤ 64 menu items. There are no menus that come close to that number, so this is OK. Should we ever need more, then this will have to be extended to a bit array.

u8 def The control's default value.

Description

Same as `v4l2_ctrl_new_std()`, but **min** is set to 0 and the **mask** value determines which menu items are to be skipped.

If **id** refers to a non-menu control, then this function will return NULL.

```
struct v4l2_ctrl * v4l2_ctrl_new_std_menu_items(struct v4l2_ctrl_handler *hdl, const
                                             struct v4l2_ctrl_ops *ops, u32 id,
                                             u8 max, u64 mask, u8 def, const char
                                             * const *qmenu)
```

Create a new standard V4L2 menu control with driver specific menu.

Parameters

struct v4l2_ctrl_handler *hdl The control handler.

const struct v4l2_ctrl_ops *ops The control ops.

u32 id The control ID.

u8 max The control's maximum value.

u64 mask The control's skip mask for menu controls. This makes it easy to skip menu items that are not valid. If bit X is set, then menu item X is skipped. Of course, this only works for menus with ≤ 64 menu items. There are no menus that come close to that number, so this is OK. Should we ever need more, then this will have to be extended to a bit array.

u8 def The control's default value.

const char * const *qmenu The new menu.

Description

Same as `v4l2_ctrl_new_std_menu()`, but **qmenu** will be the driver specific menu of this control.

```
struct v4l2_ctrl * v4l2_ctrl_new_std_compound(struct v4l2_ctrl_handler *hdl, const
                                             struct v4l2_ctrl_ops *ops, u32 id, const
                                             union v4l2_ctrl_ptr p_def)
```

Allocate and initialize a new standard V4L2 compound control.

Parameters

struct v4l2_ctrl_handler *hdl The control handler.

const struct v4l2_ctrl_ops *ops The control ops.

u32 id The control ID.

const union v4l2_ctrl_ptr p_def The control's default value.

Description

Sames as *v4l2_ctrl_new_std()*, but with support to compound controls, thanks to the **p_def** field. Use *v4l2_ctrl_ptr_create()* to create **p_def** from a pointer. Use *v4l2_ctrl_ptr_create(NULL)* if the default value of the compound control should be all zeroes.

```
struct v4l2_ctrl *v4l2_ctrl_new_int_menu(struct v4l2_ctrl_handler *hdl, const struct
                                     v4l2_ctrl_ops *ops, u32 id, u8 max, u8 def,
                                     const s64 *qmenu_int)
```

Create a new standard V4L2 integer menu control.

Parameters

struct v4l2_ctrl_handler *hdl The control handler.

const struct v4l2_ctrl_ops *ops The control ops.

u32 id The control ID.

u8 max The control's maximum value.

u8 def The control's default value.

const s64 *qmenu_int The control's menu entries.

Description

Same as *v4l2_ctrl_new_std_menu()*, but **mask** is set to 0 and it additionally takes as an argument an array of integers determining the menu items.

If **id** refers to a non-integer-menu control, then this function will return NULL.

v4l2_ctrl_filter

Typedef: Typedef to define the filter function to be used when adding a control handler.

Syntax

```
bool v4l2_ctrl_filter (const struct v4l2_ctrl *ctrl)
```

Parameters

const struct v4l2_ctrl *ctrl pointer to struct *v4l2_ctrl*.

```
int v4l2_ctrl_add_handler(struct v4l2_ctrl_handler *hdl, struct v4l2_ctrl_handler *add,
                        v4l2_ctrl_filter filter, bool from_other_dev)
```

Add all controls from handler **add** to handler **hdl**.

Parameters

struct v4l2_ctrl_handler *hdl The control handler.

struct v4l2_ctrl_handler *add The control handler whose controls you want to add to the **hdl** control handler.

v4l2_ctrl_filter filter This function will filter which controls should be added.

bool from_other_dev If true, then the controls in **add** were defined in another device than **hdl**.

Description

Does nothing if either of the two handlers is a NULL pointer. If **filter** is NULL, then all controls are added. Otherwise only those controls for which **filter** returns true will be added. In case of an error **hdl->error** will be set to the error code (if it wasn't set already).

bool **v4l2_ctrl_radio_filter**(const struct *v4l2_ctrl* *ctrl)
Standard filter for radio controls.

Parameters

const struct v4l2_ctrl *ctrl The control that is filtered.

Description

This will return true for any controls that are valid for radio device nodes. Those are all of the V4L2_CID_AUDIO_* user controls and all FM transmitter class controls.

This function is to be used with *v4l2_ctrl_add_handler()*.

void **v4l2_ctrl_cluster**(unsigned int ncontrols, struct *v4l2_ctrl* **controls)
Mark all controls in the cluster as belonging to that cluster.

Parameters

unsigned int ncontrols The number of controls in this cluster.

struct v4l2_ctrl **controls The cluster control array of size **ncontrols**.

void **v4l2_ctrl_auto_cluster**(unsigned int ncontrols, struct *v4l2_ctrl* **controls,
u8 manual_val, bool set_volatile)
Mark all controls in the cluster as belonging to that cluster and set it up for autofoo/footype handling.

Parameters

unsigned int ncontrols The number of controls in this cluster.

struct v4l2_ctrl **controls The cluster control array of size **ncontrols**. The first control must be the 'auto' control (e.g. autogain, autoexposure, etc.)

u8 manual_val The value for the first control in the cluster that equals the manual setting.

bool set_volatile If true, then all controls except the first auto control will be volatile.

Description

Use for control groups where one control selects some automatic feature and the other controls are only active whenever the automatic feature is turned off (manual mode). Typical examples: autogain vs gain, auto-whitebalance vs red and blue balance, etc.

The behavior of such controls is as follows:

When the autofoo control is set to automatic, then any manual controls are set to inactive and any reads will call *g_volatile_ctrl* (if the control was marked volatile).

When the autofoo control is set to manual, then any manual controls will be marked active, and any reads will just return the current value without going through *g_volatile_ctrl*.

In addition, this function will set the V4L2_CTRL_FLAG_UPDATE flag on the autofoo control and V4L2_CTRL_FLAG_INACTIVE on the foo control(s) if autofoo is in auto mode.

struct *v4l2_ctrl* * **v4l2_ctrl_find**(struct *v4l2_ctrl_handler* *hdl, u32 id)
Find a control with the given ID.

Parameters

struct v4l2_ctrl_handler *hdl The control handler.

u32 id The control ID to find.

Description

If **hdl** == NULL this will return NULL as well. Will lock the handler so do not use from inside *v4l2_ctrl_ops*.

void **v4l2_ctrl_activate**(struct *v4l2_ctrl* *ctrl, bool active)
Make the control active or inactive.

Parameters

struct v4l2_ctrl *ctrl The control to (de)activate.

bool active True if the control should become active.

Description

This sets or clears the V4L2_CTRL_FLAG_INACTIVE flag atomically. Does nothing if **ctrl** == NULL. This will usually be called from within the *s_ctrl* op. The V4L2_EVENT_CTRL event will be generated afterwards.

This function assumes that the control handler is locked.

void **__v4l2_ctrl_grab**(struct *v4l2_ctrl* *ctrl, bool grabbed)
Unlocked variant of *v4l2_ctrl_grab*.

Parameters

struct v4l2_ctrl *ctrl The control to (de)activate.

bool grabbed True if the control should become grabbed.

Description

This sets or clears the V4L2_CTRL_FLAG_GRABBED flag atomically. Does nothing if **ctrl** == NULL. The V4L2_EVENT_CTRL event will be generated afterwards. This will usually be called when starting or stopping streaming in the driver.

This function assumes that the control handler is locked by the caller.

void **v4l2_ctrl_grab**(struct *v4l2_ctrl* *ctrl, bool grabbed)
Mark the control as grabbed or not grabbed.

Parameters

struct v4l2_ctrl *ctrl The control to (de)activate.

bool grabbed True if the control should become grabbed.

Description

This sets or clears the V4L2_CTRL_FLAG_GRABBED flag atomically. Does nothing if **ctrl** == NULL. The V4L2_EVENT_CTRL event will be generated afterwards. This will usually be called when starting or stopping streaming in the driver.

This function assumes that the control handler is not locked and will take the lock itself.

int `__v4l2_ctrl_modify_range`(struct *v4l2_ctrl* *ctrl, s64 min, s64 max, u64 step, s64 def)
Unlocked variant of *v4l2_ctrl_modify_range()*

Parameters

struct v4l2_ctrl *ctrl The control to update.

s64 min The control's minimum value.

s64 max The control's maximum value.

u64 step The control's step value

s64 def The control's default value.

Description

Update the range of a control on the fly. This works for control types INTEGER, BOOLEAN, MENU, INTEGER MENU and BITMASK. For menu controls the **step** value is interpreted as a menu_skip_mask.

An error is returned if one of the range arguments is invalid for this control type.

The caller is responsible for acquiring the control handler mutex on behalf of *__v4l2_ctrl_modify_range()*.

int **v4l2_ctrl_modify_range**(struct *v4l2_ctrl* *ctrl, s64 min, s64 max, u64 step, s64 def)
Update the range of a control.

Parameters

struct v4l2_ctrl *ctrl The control to update.

s64 min The control's minimum value.

s64 max The control's maximum value.

u64 step The control's step value

s64 def The control's default value.

Description

Update the range of a control on the fly. This works for control types INTEGER, BOOLEAN, MENU, INTEGER MENU and BITMASK. For menu controls the **step** value is interpreted as a menu_skip_mask.

An error is returned if one of the range arguments is invalid for this control type.

This function assumes that the control handler is not locked and will take the lock itself.

void **v4l2_ctrl_notify**(struct *v4l2_ctrl* *ctrl, *v4l2_ctrl_notify_fnc* notify, void *priv)
Function to set a notify callback for a control.

Parameters

struct v4l2_ctrl *ctrl The control.

v4l2_ctrl_notify_fnc notify The callback function.

void *priv The callback private handle, passed as argument to the callback.

Description

This function sets a callback function for the control. If **ctrl** is NULL, then it will do nothing. If **notify** is NULL, then the notify callback will be removed.

There can be only one notify. If another already exists, then a WARN_ON will be issued and the function will do nothing.

```
const char * v4l2_ctrl_get_name(u32 id)
    Get the name of the control
```

Parameters

u32 id The control ID.

Description

This function returns the name of the given control ID or NULL if it isn't a known control.

```
const char * const * v4l2_ctrl_get_menu(u32 id)
    Get the menu string array of the control
```

Parameters

u32 id The control ID.

Description

This function returns the NULL-terminated menu string array name of the given control ID or NULL if it isn't a known menu control.

```
const s64 * v4l2_ctrl_get_int_menu(u32 id, u32 *len)
    Get the integer menu array of the control
```

Parameters

u32 id The control ID.

u32 *len The size of the integer array.

Description

This function returns the integer array of the given control ID or NULL if it if it isn't a known integer menu control.

```
s32 v4l2_ctrl_g_ctrl(struct v4l2_ctrl *ctrl)
    Helper function to get the control's value from within a driver.
```

Parameters

struct v4l2_ctrl *ctrl The control.

Description

This returns the control's value safely by going through the control framework. This function will lock the control's handler, so it cannot be used from within the *v4l2_ctrl_ops* functions.

This function is for integer type controls only.

```
int __v4l2_ctrl_s_ctrl(struct v4l2_ctrl *ctrl, s32 val)
    Unlocked variant of v4l2_ctrl_s_ctrl().
```

Parameters

struct v4l2_ctrl *ctrl The control.

s32 val The new value.

Description

This sets the control's new value safely by going through the control framework. This function assumes the control's handler is already locked, allowing it to be used from within the *v4l2_ctrl_ops* functions.

This function is for integer type controls only.

int **v4l2_ctrl_s_ctrl**(struct *v4l2_ctrl* *ctrl, s32 val)

Helper function to set the control's value from within a driver.

Parameters

struct v4l2_ctrl *ctrl The control.

s32 val The new value.

Description

This sets the control's new value safely by going through the control framework. This function will lock the control's handler, so it cannot be used from within the *v4l2_ctrl_ops* functions.

This function is for integer type controls only.

s64 **v4l2_ctrl_g_ctrl_int64**(struct *v4l2_ctrl* *ctrl)

Helper function to get a 64-bit control's value from within a driver.

Parameters

struct v4l2_ctrl *ctrl The control.

Description

This returns the control's value safely by going through the control framework. This function will lock the control's handler, so it cannot be used from within the *v4l2_ctrl_ops* functions.

This function is for 64-bit integer type controls only.

int **__v4l2_ctrl_s_ctrl_int64**(struct *v4l2_ctrl* *ctrl, s64 val)

Unlocked variant of *v4l2_ctrl_s_ctrl_int64()*.

Parameters

struct v4l2_ctrl *ctrl The control.

s64 val The new value.

Description

This sets the control's new value safely by going through the control framework. This function assumes the control's handler is already locked, allowing it to be used from within the *v4l2_ctrl_ops* functions.

This function is for 64-bit integer type controls only.

int **v4l2_ctrl_s_ctrl_int64**(struct *v4l2_ctrl* *ctrl, s64 val)

Helper function to set a 64-bit control's value from within a driver.

Parameters

struct v4l2_ctrl *ctrl The control.

s64 val The new value.

Description

This sets the control's new value safely by going through the control framework. This function will lock the control's handler, so it cannot be used from within the *v4l2_ctrl_ops* functions.

This function is for 64-bit integer type controls only.

```
int __v4l2_ctrl_s_ctrl_string(struct v4l2_ctrl *ctrl, const char *s)
    Unlocked variant of v4l2_ctrl_s_ctrl_string().
```

Parameters

struct v4l2_ctrl *ctrl The control.

const char *s The new string.

Description

This sets the control's new string safely by going through the control framework. This function assumes the control's handler is already locked, allowing it to be used from within the *v4l2_ctrl_ops* functions.

This function is for string type controls only.

```
int v4l2_ctrl_s_ctrl_string(struct v4l2_ctrl *ctrl, const char *s)
    Helper function to set a control's string value from within a driver.
```

Parameters

struct v4l2_ctrl *ctrl The control.

const char *s The new string.

Description

This sets the control's new string safely by going through the control framework. This function will lock the control's handler, so it cannot be used from within the *v4l2_ctrl_ops* functions.

This function is for string type controls only.

```
int __v4l2_ctrl_s_ctrl_compound(struct v4l2_ctrl *ctrl, enum v4l2_ctrl_type type, const
                               void *p)
    Unlocked variant to set a compound control
```

Parameters

struct v4l2_ctrl *ctrl The control.

enum v4l2_ctrl_type type The type of the data.

const void *p The new compound payload.

Description

This sets the control's new compound payload safely by going through the control framework. This function assumes the control's handler is already locked, allowing it to be used from within the *v4l2_ctrl_ops* functions.

This function is for compound type controls only.

```
int v4l2_ctrl_s_ctrl_compound(struct v4l2_ctrl *ctrl, enum v4l2_ctrl_type type, const
                              void *p)
    Helper function to set a compound control from within a driver.
```

Parameters

struct v4l2_ctrl *ctrl The control.
enum v4l2_ctrl_type type The type of the data.
const void *p The new compound payload.

Description

This sets the control's new compound payload safely by going through the control framework. This function will lock the control's handler, so it cannot be used from within the *v4l2_ctrl_ops* functions.

This function is for compound type controls only.

void **v4l2_ctrl_replace**(struct v4l2_event *old, const struct v4l2_event *new)
Function to be used as a callback to *struct v4l2_subscribed_event_ops* replace()

Parameters

struct v4l2_event *old pointer to struct v4l2_event with the reported event;
const struct v4l2_event *new pointer to struct v4l2_event with the modified event;
void **v4l2_ctrl_merge**(const struct v4l2_event *old, struct v4l2_event *new)
Function to be used as a callback to *struct v4l2_subscribed_event_ops* merge()

Parameters

const struct v4l2_event *old pointer to struct v4l2_event with the reported event;
struct v4l2_event *new pointer to struct v4l2_event with the merged event;
int **v4l2_ctrl_log_status**(struct file *file, void *fh)
helper function to implement VIDIOC_LOG_STATUS ioctl

Parameters

struct file *file pointer to struct file
void *fh unused. Kept just to be compatible to the arguments expected by *struct v4l2_ioctl_ops*.vidioc_log_status.

Description

Can be used as a vidIOC_log_status function that just dumps all controls associated with the filehandle.

int **v4l2_ctrl_subscribe_event**(struct *v4l2_fh* *fh, const struct
v4l2_event_subscription *sub)
Subscribes to an event

Parameters

struct v4l2_fh *fh pointer to *struct v4l2_fh*
const struct v4l2_event_subscription *sub pointer to struct
v4l2_event_subscription

Description

Can be used as a vidIOC_subscribe_event function that just subscribes control events.

`__poll_t v4l2_ctrl_poll`(struct file *file, struct poll_table_struct *wait)
function to be used as a callback to the poll() That just polls for control events.

Parameters

struct file *file pointer to struct file

struct poll_table_struct *wait pointer to struct poll_table_struct

int **v4l2_ctrl_request_setup**(struct *media_request* *req, struct *v4l2_ctrl_handler* *parent)
helper function to apply control values in a request

Parameters

struct media_request *req The request

struct v4l2_ctrl_handler *parent The parent control handler ('priv' in *media_request_object_find()*)

Description

This is a helper function to call the control handler's `s_ctrl` callback with the control values contained in the request. Do note that this approach of applying control values in a request is only applicable to memory-to-memory devices.

void **v4l2_ctrl_request_complete**(struct *media_request* *req, struct *v4l2_ctrl_handler* *parent)
Complete a control handler request object

Parameters

struct media_request *req The request

struct v4l2_ctrl_handler *parent The parent control handler ('priv' in *media_request_object_find()*)

Description

This function is to be called on each control handler that may have had a request object associated with it, i.e. control handlers of a driver that supports requests.

The function first obtains the values of any volatile controls in the control handler and attach them to the request. Then, the function completes the request object.

struct *v4l2_ctrl_handler* * **v4l2_ctrl_request_hdl_find**(struct *media_request* *req, struct *v4l2_ctrl_handler* *parent)

Find the control handler in the request

Parameters

struct media_request *req The request

struct v4l2_ctrl_handler *parent The parent control handler ('priv' in *media_request_object_find()*)

Description

This function finds the control handler in the request. It may return NULL if not found. When done, you must call `v4l2_ctrl_request_put_hdl()` with the returned handler pointer.

If the request is not in state `VALIDATING` or `QUEUED`, then this function will always return `NULL`.

Note that in state `VALIDATING` the `req_queue_mutex` is held, so no objects can be added or deleted from the request.

In state `QUEUED` it is the driver that will have to ensure this.

`void v4l2_ctrl_request_hdl_put(struct v4l2_ctrl_handler *hdl)`
Put the control handler

Parameters

`struct v4l2_ctrl_handler *hdl` Put this control handler

Description

This function released the control handler previously obtained from `v4l2_ctrl_request_hdl_find()`.

`struct v4l2_ctrl *v4l2_ctrl_request_hdl_ctrl_find(struct v4l2_ctrl_handler *hdl, u32 id)`
Find a control with the given ID.

Parameters

`struct v4l2_ctrl_handler *hdl` The control handler from the request.

`u32 id` The ID of the control to find.

Description

This function returns a pointer to the control if this control is part of the request or `NULL` otherwise.

`int v4l2_queryctrl(struct v4l2_ctrl_handler *hdl, struct v4l2_queryctrl *qc)`
Helper function to implement `VIDIOC_QUERYCTRL` ioctl

Parameters

`struct v4l2_ctrl_handler *hdl` pointer to `struct v4l2_ctrl_handler`

`struct v4l2_queryctrl *qc` pointer to `struct v4l2_queryctrl`

Description

If `hdl == NULL` then they will all return `-EINVAL`.

`int v4l2_query_ext_ctrl(struct v4l2_ctrl_handler *hdl, struct v4l2_query_ext_ctrl *qc)`
Helper function to implement `VIDIOC_QUERY_EXT_CTRL` ioctl

Parameters

`struct v4l2_ctrl_handler *hdl` pointer to `struct v4l2_ctrl_handler`

`struct v4l2_query_ext_ctrl *qc` pointer to `struct v4l2_query_ext_ctrl`

Description

If `hdl == NULL` then they will all return `-EINVAL`.

`int v4l2_querymenu(struct v4l2_ctrl_handler *hdl, struct v4l2_querymenu *qm)`
Helper function to implement `VIDIOC_QUERYMENU` ioctl

Parameters

struct v4l2_ctrl_handler *hdl pointer to *struct v4l2_ctrl_handler*

struct v4l2_querymenu *qm pointer to *struct v4l2_querymenu*

Description

If hdl == NULL then they will all return -EINVAL.

int **v4l2_g_ctrl**(struct *v4l2_ctrl_handler* *hdl, struct v4l2_control *ctrl)
 Helper function to implement *VIDIOC_G_CTRL* ioctl

Parameters

struct v4l2_ctrl_handler *hdl pointer to *struct v4l2_ctrl_handler*

struct v4l2_control *ctrl pointer to struct v4l2_control

Description

If hdl == NULL then they will all return -EINVAL.

int **v4l2_s_ctrl**(struct *v4l2_fh* *fh, struct *v4l2_ctrl_handler* *hdl, struct v4l2_control *ctrl)
 Helper function to implement *VIDIOC_S_CTRL* ioctl

Parameters

struct v4l2_fh *fh pointer to *struct v4l2_fh*

struct v4l2_ctrl_handler *hdl pointer to *struct v4l2_ctrl_handler*

struct v4l2_control *ctrl pointer to struct v4l2_control

Description

If hdl == NULL then they will all return -EINVAL.

int **v4l2_g_ext_ctrls**(struct *v4l2_ctrl_handler* *hdl, struct *video_device* *vdev, struct *media_device* *mdev, struct v4l2_ext_controls *c)
 Helper function to implement *VIDIOC_G_EXT_CTRL*s ioctl

Parameters

struct v4l2_ctrl_handler *hdl pointer to *struct v4l2_ctrl_handler*

struct video_device *vdev pointer to *struct video_device*

struct media_device *mdev pointer to *struct media_device*

struct v4l2_ext_controls *c pointer to struct v4l2_ext_controls

Description

If hdl == NULL then they will all return -EINVAL.

int **v4l2_try_ext_ctrls**(struct *v4l2_ctrl_handler* *hdl, struct *video_device* *vdev, struct *media_device* *mdev, struct v4l2_ext_controls *c)
 Helper function to implement *VIDIOC_TRY_EXT_CTRL*s ioctl

Parameters

struct v4l2_ctrl_handler *hdl pointer to *struct v4l2_ctrl_handler*

struct video_device *vdev pointer to *struct video_device*

struct media_device *mdev pointer to *struct media_device*

struct v4l2_ext_controls *c pointer to `struct v4l2_ext_controls`

Description

If `hdl == NULL` then they will all return `-EINVAL`.

int **v4l2_s_ext_ctrls**(`struct v4l2_fh *fh`, `struct v4l2_ctrl_handler *hdl`, `struct video_device *vdev`, `struct media_device *mdev`, `struct v4l2_ext_controls *c`)

Helper function to implement `VIDIOC_S_EXT_CTRL`s ioctl

Parameters

struct v4l2_fh *fh pointer to `struct v4l2_fh`

struct v4l2_ctrl_handler *hdl pointer to `struct v4l2_ctrl_handler`

struct video_device *vdev pointer to `struct video_device`

struct media_device *mdev pointer to `struct media_device`

struct v4l2_ext_controls *c pointer to `struct v4l2_ext_controls`

Description

If `hdl == NULL` then they will all return `-EINVAL`.

int **v4l2_ctrl_subdev_subscribe_event**(`struct v4l2_subdev *sd`, `struct v4l2_fh *fh`, `struct v4l2_event_subscription *sub`)

Helper function to implement as a `struct v4l2_subdev_core_ops` `subscribe_event` function that just subscribes control events.

Parameters

struct v4l2_subdev *sd pointer to `struct v4l2_subdev`

struct v4l2_fh *fh pointer to `struct v4l2_fh`

struct v4l2_event_subscription *sub pointer to `struct v4l2_event_subscription`

int **v4l2_ctrl_subdev_log_status**(`struct v4l2_subdev *sd`)

Log all controls owned by subdev's control handler.

Parameters

struct v4l2_subdev *sd pointer to `struct v4l2_subdev`

int **v4l2_ctrl_new_fwnode_properties**(`struct v4l2_ctrl_handler *hdl`, `const struct v4l2_ctrl_ops *ctrl_ops`, `const struct v4l2_fwnode_device_properties *p`)

Register controls for the device properties

Parameters

struct v4l2_ctrl_handler *hdl pointer to `struct v4l2_ctrl_handler` to register controls on

const struct v4l2_ctrl_ops *ctrl_ops pointer to `struct v4l2_ctrl_ops` to register controls with

const struct v4l2_fwnode_device_properties *p pointer to `struct v4l2_fwnode_device_properties`

Description

This function registers controls associated to device properties, using the property values contained in **p** parameter, if the property has been set to a value.

Currently the following v4l2 controls are parsed and registered: - V4L2_CID_CAMERA_ORIENTATION - V4L2_CID_CAMERA_SENSOR_ROTATION;

Controls already registered by the caller with the **hdl** control handler are not overwritten. Callers should register the controls they want to handle themselves before calling this function.

Return

0 on success, a negative error code on failure.

2.2.14 Videobuf Framework

Author: Jonathan Corbet <corbet@lwn.net>

Current as of 2.6.33

Note: The videobuf framework was deprecated in favor of videobuf2. Shouldn't be used on new drivers.

2.2.14.1 Introduction

The videobuf layer functions as a sort of glue layer between a V4L2 driver and user space. It handles the allocation and management of buffers for the storage of video frames. There is a set of functions which can be used to implement many of the standard POSIX I/O system calls, including `read()`, `poll()`, and, happily, `mmap()`. Another set of functions can be used to implement the bulk of the V4L2 `ioctl()` calls related to streaming I/O, including buffer allocation, queueing and dequeuing, and streaming control. Using videobuf imposes a few design decisions on the driver author, but the payback comes in the form of reduced code in the driver and a consistent implementation of the V4L2 user-space API.

2.2.14.2 Buffer types

Not all video devices use the same kind of buffers. In fact, there are (at least) three common variations:

- Buffers which are scattered in both the physical and (kernel) virtual address spaces. (Almost) all user-space buffers are like this, but it makes great sense to allocate kernel-space buffers this way as well when it is possible. Unfortunately, it is not always possible; working with this kind of buffer normally requires hardware which can do scatter/gather DMA operations.
- Buffers which are physically scattered, but which are virtually contiguous; buffers allocated with `vmalloc()`, in other words. These buffers are just as hard to use for DMA operations, but they can be useful in situations where DMA is not available but virtually-contiguous buffers are convenient.

- Buffers which are physically contiguous. Allocation of this kind of buffer can be unreliable on fragmented systems, but simpler DMA controllers cannot deal with anything else.

Videobuf can work with all three types of buffers, but the driver author must pick one at the outset and design the driver around that decision.

[It's worth noting that there's a fourth kind of buffer: "overlay" buffers which are located within the system's video memory. The overlay functionality is considered to be deprecated for most use, but it still shows up occasionally in system-on-chip drivers where the performance benefits merit the use of this technique. Overlay buffers can be handled as a form of scattered buffer, but there are very few implementations in the kernel and a description of this technique is currently beyond the scope of this document.]

2.2.14.3 Data structures, callbacks, and initialization

Depending on which type of buffers are being used, the driver should include one of the following files:

```
<media/videobuf-dma-sg.h>          /* Physically scattered */
<media/videobuf-vmalloc.h>        /* vmalloc() buffers    */
<media/videobuf-dma-contig.h>     /* Physically contiguous */
```

The driver's data structure describing a V4L2 device should include a struct `videobuf_queue` instance for the management of the buffer queue, along with a `list_head` for the queue of available buffers. There will also need to be an interrupt-safe spinlock which is used to protect (at least) the queue.

The next step is to write four simple callbacks to help videobuf deal with the management of buffers:

```
struct videobuf_queue_ops {
    int (*buf_setup)(struct videobuf_queue *q,
                    unsigned int *count, unsigned int *size);
    int (*buf_prepare)(struct videobuf_queue *q,
                      struct videobuf_buffer *vb,
                      enum v4l2_field field);
    void (*buf_queue)(struct videobuf_queue *q,
                     struct videobuf_buffer *vb);
    void (*buf_release)(struct videobuf_queue *q,
                       struct videobuf_buffer *vb);
};
```

`buf_setup()` is called early in the I/O process, when streaming is being initiated; its purpose is to tell videobuf about the I/O stream. The count parameter will be a suggested number of buffers to use; the driver should check it for rationality and adjust it if need be. As a practical rule, a minimum of two buffers are needed for proper streaming, and there is usually a maximum (which cannot exceed 32) which makes sense for each device. The size parameter should be set to the expected (maximum) size for each frame of data.

Each buffer (in the form of a struct `videobuf_buffer` pointer) will be passed to `buf_prepare()`, which should set the buffer's size, width, height, and field fields properly. If the buffer's state field is `VIDEObUF_NEEDS_INIT`, the driver should pass it to:

```
int videobuf_iolock(struct videobuf_queue* q, struct videobuf_buffer *vb,
                   struct v4l2_framebuffer *fbuf);
```

Among other things, this call will usually allocate memory for the buffer. Finally, the `buf_prepare()` function should set the buffer's state to `VIDEObUF_PREPARED`.

When a buffer is queued for I/O, it is passed to `buf_queue()`, which should put it onto the driver's list of available buffers and set its state to `VIDEObUF_QUEUED`. Note that this function is called with the queue spinlock held; if it tries to acquire it as well things will come to a screeching halt. Yes, this is the voice of experience. Note also that videobuf may wait on the first buffer in the queue; placing other buffers in front of it could again gum up the works. So use `list_add_tail()` to enqueue buffers.

Finally, `buf_release()` is called when a buffer is no longer intended to be used. The driver should ensure that there is no I/O active on the buffer, then pass it to the appropriate free routine(s):

```
/* Scatter/gather drivers */
int videobuf_dma_unmap(struct videobuf_queue *q,
                      struct videobuf_dmabuf *dma);
int videobuf_dma_free(struct videobuf_dmabuf *dma);

/* vmalloc drivers */
void videobuf_vmalloc_free (struct videobuf_buffer *buf);

/* Contiguous drivers */
void videobuf_dma_contig_free(struct videobuf_queue *q,
                              struct videobuf_buffer *buf);
```

One way to ensure that a buffer is no longer under I/O is to pass it to:

```
int videobuf_waiton(struct videobuf_buffer *vb, int non_blocking, int intr);
```

Here, `vb` is the buffer, `non_blocking` indicates whether non-blocking I/O should be used (it should be zero in the `buf_release()` case), and `intr` controls whether an interruptible wait is used.

2.2.14.4 File operations

At this point, much of the work is done; much of the rest is slipping videobuf calls into the implementation of the other driver callbacks. The first step is in the `open()` function, which must initialize the videobuf queue. The function to use depends on the type of buffer used:

```
void videobuf_queue_sg_init(struct videobuf_queue *q,
                           struct videobuf_queue_ops *ops,
                           struct device *dev,
                           spinlock_t *irqlock,
                           enum v4l2_buf_type type,
                           enum v4l2_field field,
                           unsigned int msize,
                           void *priv);

void videobuf_queue_vmalloc_init(struct videobuf_queue *q,
                                 struct videobuf_queue_ops *ops,
                                 struct device *dev,
                                 spinlock_t *irqlock,
                                 enum v4l2_buf_type type,
                                 enum v4l2_field field,
                                 unsigned int msize,
                                 void *priv);
```

```
void videobuf_queue_dma_contig_init(struct videobuf_queue *q,
                                   struct videobuf_queue_ops *ops,
                                   struct device *dev,
                                   spinlock_t *irqlock,
                                   enum v4l2_buf_type type,
                                   enum v4l2_field field,
                                   unsigned int msize,
                                   void *priv);
```

In each case, the parameters are the same: `q` is the queue structure for the device, `ops` is the set of callbacks as described above, `dev` is the device structure for this video device, `irqlock` is an interrupt-safe spinlock to protect access to the data structures, `type` is the buffer type used by the device (cameras will use `V4L2_BUF_TYPE_VIDEO_CAPTURE`, for example), `field` describes which field is being captured (often `V4L2_FIELD_NONE` for progressive devices), `msize` is the size of any containing structure used around `struct videobuf_buffer`, and `priv` is a private data pointer which shows up in the `priv_data` field of `struct videobuf_queue`. Note that these are void functions which, evidently, are immune to failure.

V4L2 capture drivers can be written to support either of two APIs: the `read()` system call and the rather more complicated streaming mechanism. As a general rule, it is necessary to support both to ensure that all applications have a chance of working with the device. Videobuf makes it easy to do that with the same code. To implement `read()`, the driver need only make a call to one of:

```
ssize_t videobuf_read_one(struct videobuf_queue *q,
                          char __user *data, size_t count,
                          loff_t *ppos, int nonblocking);

ssize_t videobuf_read_stream(struct videobuf_queue *q,
                              char __user *data, size_t count,
                              loff_t *ppos, int vbihack, int nonblocking);
```

Either one of these functions will read frame data into `data`, returning the amount actually read; the difference is that `videobuf_read_one()` will only read a single frame, while `videobuf_read_stream()` will read multiple frames if they are needed to satisfy the count requested by the application. A typical driver `read()` implementation will start the capture engine, call one of the above functions, then stop the engine before returning (though a smarter implementation might leave the engine running for a little while in anticipation of another `read()` call happening in the near future).

The `poll()` function can usually be implemented with a direct call to:

```
unsigned int videobuf_poll_stream(struct file *file,
                                  struct videobuf_queue *q,
                                  poll_table *wait);
```

Note that the actual wait queue eventually used will be the one associated with the first available buffer.

When streaming I/O is done to kernel-space buffers, the driver must support the `mmap()` system call to enable user space to access the data. In many V4L2 drivers, the often-complex `mmap()` implementation simplifies to a single call to:

```
int videobuf_mmap_mapper(struct videobuf_queue *q,
                          struct vm_area_struct *vma);
```

Everything else is handled by the videobuf code.

The `release()` function requires two separate videobuf calls:

```
void videobuf_stop(struct videobuf_queue *q);
int videobuf_mmap_free(struct videobuf_queue *q);
```

The call to `videobuf_stop()` terminates any I/O in progress - though it is still up to the driver to stop the capture engine. The call to `videobuf_mmap_free()` will ensure that all buffers have been unmapped; if so, they will all be passed to the `buf_release()` callback. If buffers remain mapped, `videobuf_mmap_free()` returns an error code instead. The purpose is clearly to cause the closing of the file descriptor to fail if buffers are still mapped, but every driver in the 2.6.32 kernel cheerfully ignores its return value.

2.2.14.5 ioctl() operations

The V4L2 API includes a very long list of driver callbacks to respond to the many `ioctl()` commands made available to user space. A number of these - those associated with streaming I/O - turn almost directly into videobuf calls. The relevant helper functions are:

```
int videobuf_reqbufs(struct videobuf_queue *q,
                    struct v4l2_requestbuffers *req);
int videobuf_querybuf(struct videobuf_queue *q, struct v4l2_buffer *b);
int videobuf_qbuf(struct videobuf_queue *q, struct v4l2_buffer *b);
int videobuf_dqbuf(struct videobuf_queue *q, struct v4l2_buffer *b,
                  int nonblocking);
int videobuf_streamon(struct videobuf_queue *q);
int videobuf_streamoff(struct videobuf_queue *q);
```

So, for example, a `VIDIOC_REQBUFS` call turns into a call to the driver's `vidioc_reqbufs()` callback which, in turn, usually only needs to locate the proper `struct videobuf_queue` pointer and pass it to `videobuf_reqbufs()`. These support functions can replace a great deal of buffer management boilerplate in a lot of V4L2 drivers.

The `vidioc_streamon()` and `vidioc_streamoff()` functions will be a bit more complex, of course, since they will also need to deal with starting and stopping the capture engine.

2.2.14.6 Buffer allocation

Thus far, we have talked about buffers, but have not looked at how they are allocated. The scatter/gather case is the most complex on this front. For allocation, the driver can leave buffer allocation entirely up to the videobuf layer; in this case, buffers will be allocated as anonymous user-space pages and will be very scattered indeed. If the application is using user-space buffers, no allocation is needed; the videobuf layer will take care of calling `get_user_pages()` and filling in the scatterlist array.

If the driver needs to do its own memory allocation, it should be done in the `vidioc_reqbufs()` function, *after* calling `videobuf_reqbufs()`. The first step is a call to:

```
struct videobuf_dmabuf *videobuf_to_dma(struct videobuf_buffer *buf);
```

The returned `videobuf_dmabuf` structure (defined in `<media/videobuf-dma-sg.h>`) includes a couple of relevant fields:

```
struct scatterlist *sglist;
int                sglen;
```

The driver must allocate an appropriately-sized scatterlist array and populate it with pointers to the pieces of the allocated buffer; `sglen` should be set to the length of the array.

Drivers using the `vmalloc()` method need not (and cannot) concern themselves with buffer allocation at all; videobuf will handle those details. The same is normally true of contiguous-DMA drivers as well; videobuf will allocate the buffers (with `dma_alloc_coherent()`) when it sees fit. That means that these drivers may be trying to do high-order allocations at any time, an operation which is not always guaranteed to work. Some drivers play tricks by allocating DMA space at system boot time; videobuf does not currently play well with those drivers.

As of 2.6.31, contiguous-DMA drivers can work with a user-supplied buffer, as long as that buffer is physically contiguous. Normal user-space allocations will not meet that criterion, but buffers obtained from other kernel drivers, or those contained within huge pages, will work with these drivers.

2.2.14.7 Filling the buffers

The final part of a videobuf implementation has no direct callback - it's the portion of the code which actually puts frame data into the buffers, usually in response to interrupts from the device. For all types of drivers, this process works approximately as follows:

- Obtain the next available buffer and make sure that somebody is actually waiting for it.
- Get a pointer to the memory and put video data there.
- Mark the buffer as done and wake up the process waiting for it.

Step (1) above is done by looking at the driver-managed `list_head` structure - the one which is filled in the `buf_queue()` callback. Because starting the engine and enqueueing buffers are done in separate steps, it's possible for the engine to be running without any buffers available - in the `vmalloc()` case especially. So the driver should be prepared for the list to be empty. It is equally possible that nobody is yet interested in the buffer; the driver should not remove it from the list or fill it until a process is waiting on it. That test can be done by examining the buffer's `done` field (a `wait_queue_head_t` structure) with `waitqueue_active()`.

A buffer's state should be set to `VIDEObUF_ACTIVE` before being mapped for DMA; that ensures that the videobuf layer will not try to do anything with it while the device is transferring data.

For scatter/gather drivers, the needed memory pointers will be found in the scatterlist structure described above. Drivers using the `vmalloc()` method can get a memory pointer with:

```
void *videobuf_to_vmalloc(struct videobuf_buffer *buf);
```

For contiguous DMA drivers, the function to use is:

```
dma_addr_t videobuf_to_dma_contig(struct videobuf_buffer *buf);
```

The contiguous DMA API goes out of its way to hide the kernel-space address of the DMA buffer from drivers.

The final step is to set the `size` field of the relevant `videobuf_buffer` structure to the actual size of the captured image, set state to `VIDEObUF_DONE`, then call `wake_up()` on the done queue. At this point, the buffer is owned by the videobuf layer and the driver should not touch it again.

Developers who are interested in more information can go into the relevant header files; there are a few low-level functions declared there which have not been talked about here. Note also that all of these calls are exported GPL-only, so they will not be available to non-GPL kernel modules.

2.2.15 V4L2 videobuf2 functions and data structures

enum `vb2_memory`

type of memory model used to make the buffers visible on userspace.

Constants

VB2_MEMORY_UNKNOWN Buffer status is unknown or it is not used yet on userspace.

VB2_MEMORY_MMAP The buffers are allocated by the Kernel and it is memory mapped via `mmap()` `ioctl`. This model is also used when the user is using the buffers via `read()` or `write()` system calls.

VB2_MEMORY_USERPTR The buffers was allocated in userspace and it is memory mapped via `mmap()` `ioctl`.

VB2_MEMORY_DMABUF The buffers are passed to userspace via DMA buffer.

struct `vb2_mem_ops`

memory handling/memory allocator operations.

Definition

```
struct vb2_mem_ops {
    void (*alloc)(struct vb2_buffer *vb, struct device *dev, unsigned long size);
    void (*put)(void *buf_priv);
    struct dma_buf *(*get_dmabuf)(struct vb2_buffer *vb, void *buf_priv, unsigned long
    ↪ flags);
    void *(*get_userptr)(struct vb2_buffer *vb, struct device *dev, unsigned long vaddr,
    ↪ unsigned long size);
    void (*put_userptr)(void *buf_priv);
    void (*prepare)(void *buf_priv);
    void (*finish)(void *buf_priv);
    void *(*attach_dmabuf)(struct vb2_buffer *vb, struct device *dev, struct dma_buf *dbuf,
    ↪ unsigned long size);
    void (*detach_dmabuf)(void *buf_priv);
    int (*map_dmabuf)(void *buf_priv);
    void (*unmap_dmabuf)(void *buf_priv);
    void *(*vaddr)(struct vb2_buffer *vb, void *buf_priv);
    void *(*cookie)(struct vb2_buffer *vb, void *buf_priv);
    unsigned int (*num_users)(void *buf_priv);
    int (*mmap)(void *buf_priv, struct vm_area_struct *vma);
};
```

Members

alloc allocate video memory and, optionally, allocator private data, return `ERR_PTR()` on failure or a pointer to allocator private, per-buffer data on success; the returned private structure will then be passed as **buf_priv** argument to other ops in this structure. The size argument to this function shall be *page aligned*.

put inform the allocator that the buffer will no longer be used; usually will result in the allocator freeing the buffer (if no other users of this buffer are present); the **buf_priv** argument is the allocator private per-buffer structure previously returned from the alloc callback.

get_dmabuf acquire userspace memory for a hardware operation; used for DMABUF memory types.

get_userptr acquire userspace memory for a hardware operation; used for USERPTR memory types; vaddr is the address passed to the videobuf layer when queuing a video buffer of USERPTR type; should return an allocator private per-buffer structure associated with the buffer on success, ERR_PTR() on failure; the returned private structure will then be passed as **buf_priv** argument to other ops in this structure.

put_userptr inform the allocator that a USERPTR buffer will no longer be used.

prepare called every time the buffer is passed from userspace to the driver, useful for cache synchronisation, optional.

finish called every time the buffer is passed back from the driver to the userspace, also optional.

attach_dmabuf attach a shared struct dma_buf for a hardware operation; used for DMABUF memory types; dev is the alloc device dbuf is the shared dma_buf; returns ERR_PTR() on failure; allocator private per-buffer structure on success; this needs to be used for further accesses to the buffer.

detach_dmabuf inform the exporter of the buffer that the current DMABUF buffer is no longer used; the **buf_priv** argument is the allocator private per-buffer structure previously returned from the attach_dmabuf callback.

map_dmabuf request for access to the dmabuf from allocator; the allocator of dmabuf is informed that this driver is going to use the dmabuf.

unmap_dmabuf releases access control to the dmabuf - allocator is notified that this driver is done using the dmabuf for now.

vaddr return a kernel virtual address to a given memory buffer associated with the passed private structure or NULL if no such mapping exists.

cookie return allocator specific cookie for a given memory buffer associated with the passed private structure or NULL if not available.

num_users return the current number of users of a memory buffer; return 1 if the videobuf layer (or actually the driver using it) is the only user.

mmap setup a userspace mapping for a given memory buffer under the provided virtual memory region.

Description

Those operations are used by the videobuf2 core to implement the memory handling/memory allocators for each type of supported streaming I/O method.

Note:

- 1) Required ops for USERPTR types: get_userptr, put_userptr.
- 2) Required ops for MMAP types: alloc, put, num_users, mmap.
- 3) Required ops for read/write access types: alloc, put, num_users, vaddr.

- 4) Required ops for DMABUF types: `attach_dmabuf`, `detach_dmabuf`, `map_dmabuf`, `unmap_dmabuf`.

struct **vb2_plane**
plane information.

Definition

```
struct vb2_plane {
    void *mem_priv;
    struct dma_buf      *dbuf;
    unsigned int        dbuf_mapped;
    unsigned int        bytesused;
    unsigned int        length;
    unsigned int        min_length;
    union {
        unsigned int    offset;
        unsigned long   userptr;
        int fd;
    } m;
    unsigned int        data_offset;
};
```

Members

mem_priv private data with this plane.

dbuf `dma_buf` - shared buffer object.

dbuf_mapped flag to show whether `dbuf` is mapped or not

bytesused number of bytes occupied by data in the plane (payload).

length size of this plane (NOT the payload) in bytes. The maximum valid size is `MAX_UINT - PAGE_SIZE`.

min_length minimum required size of this plane (NOT the payload) in bytes. **length** is always greater or equal to **min_length**, and like **length**, it is limited to `MAX_UINT - PAGE_SIZE`.

m Union with memtype-specific data.

m.offset when memory in the associated `struct vb2_buffer` is `VB2_MEMORY_MMAP`, equals the offset from the start of the device memory for this plane (or is a “cookie” that should be passed to `mmap()` called on the video node).

m.userptr when memory is `VB2_MEMORY_USERPTR`, a userspace pointer pointing to this plane.

m.fd when memory is `VB2_MEMORY_DMABUF`, a userspace file descriptor associated with this plane.

data_offset offset in the plane to the start of data; usually 0, unless there is a header in front of the data.

Description

Should contain enough information to be able to cover all the fields of `struct v4l2_plane` at `videodev2.h`.

enum **vb2_io_modes**
queue access methods.

Constants

VB2_MMAP driver supports MMAP with streaming API.

VB2_USERPTR driver supports USERPTR with streaming API.

VB2_READ driver supports read() style access.

VB2_WRITE driver supports write() style access.

VB2_DMABUF driver supports DMABUF with streaming API.

enum **vb2_buffer_state**
current video buffer state.

Constants

VB2_BUF_STATE_DEQUEUED buffer under userspace control.

VB2_BUF_STATE_IN_REQUEST buffer is queued in media request.

VB2_BUF_STATE_PREPARING buffer is being prepared in videobuf.

VB2_BUF_STATE_QUEUED buffer queued in videobuf, but not in driver.

VB2_BUF_STATE_ACTIVE buffer queued in driver and possibly used in a hardware operation.

VB2_BUF_STATE_DONE buffer returned from driver to videobuf, but not yet dequeued to userspace.

VB2_BUF_STATE_ERROR same as above, but the operation on the buffer has ended with an error, which will be reported to the userspace when it is dequeued.

struct **vb2_buffer**
represents a video buffer.

Definition

```
struct vb2_buffer {
    struct vb2_queue      *vb2_queue;
    unsigned int         index;
    unsigned int         type;
    unsigned int         memory;
    unsigned int         num_planes;
    u64 timestamp;
    struct media_request  *request;
    struct media_request_object req_obj;
};
```

Members

vb2_queue pointer to *struct vb2_queue* with the queue to which this driver belongs.

index id number of the buffer.

type buffer type.

memory the method, in which the actual data is passed.

num_planes number of planes in the buffer on an internal driver queue.

timestamp frame timestamp in ns.

request the request this buffer is associated with.

req_obj used to bind this buffer to a request. This request object has a refcount.

struct **vb2_ops**
driver-specific callbacks.

Definition

```
struct vb2_ops {
    int (*queue_setup)(struct vb2_queue *q, unsigned int *num_buffers, unsigned int *num_
    ↪planes, unsigned int sizes[], struct device *alloc_devs[]);
    void (*wait_prepare)(struct vb2_queue *q);
    void (*wait_finish)(struct vb2_queue *q);
    int (*buf_out_validate)(struct vb2_buffer *vb);
    int (*buf_init)(struct vb2_buffer *vb);
    int (*buf_prepare)(struct vb2_buffer *vb);
    void (*buf_finish)(struct vb2_buffer *vb);
    void (*buf_cleanup)(struct vb2_buffer *vb);
    int (*start_streaming)(struct vb2_queue *q, unsigned int count);
    void (*stop_streaming)(struct vb2_queue *q);
    void (*buf_queue)(struct vb2_buffer *vb);
    void (*buf_request_complete)(struct vb2_buffer *vb);
};
```

Members

queue_setup called from VIDIOC_REQBUFS() and VIDIOC_CREATE_BUFS() handlers before memory allocation. It can be called twice: if the original number of requested buffers could not be allocated, then it will be called a second time with the actually allocated number of buffers to verify if that is OK. The driver should return the required number of buffers in **num_buffers*, the required number of planes per buffer in **num_planes*, the size of each plane should be set in the *sizes[]* array and optional per-plane allocator specific device in the *alloc_devs[]* array. When called from VIDIOC_REQBUFS(), **num_planes* == 0, the driver has to use the currently configured format to determine the plane sizes and **num_buffers* is the total number of buffers that are being allocated. When called from VIDIOC_CREATE_BUFS(), **num_planes* != 0 and it describes the requested number of planes and *sizes[]* contains the requested plane sizes. In this case **num_buffers* are being allocated additionally to *q->num_buffers*. If either **num_planes* or the requested sizes are invalid callback must return *-EINVAL*.

wait_prepare release any locks taken while calling vb2 functions; it is called before an ioctl needs to wait for a new buffer to arrive; required to avoid a deadlock in blocking access type.

wait_finish reacquire all locks released in the previous callback; required to continue operation after sleeping while waiting for a new buffer to arrive.

buf_out_validate called when the output buffer is prepared or queued to a request; drivers can use this to validate userspace-provided information; this is required only for OUTPUT queues.

buf_init called once after allocating a buffer (in MMAP case) or after acquiring a new USERPTR buffer; drivers may perform additional buffer-related initialization; initialization failure (return != 0) will prevent queue setup from completing successfully; optional.

buf_prepare called every time the buffer is queued from userspace and from the VIDIOC_PREPARE_BUF() ioctl; drivers may perform any initialization required before each hardware operation in this callback; drivers can access/modify the buffer here as it is still

synced for the CPU; drivers that support `VIDIOC_CREATE_BUFS()` must also validate the buffer size; if an error is returned, the buffer will not be queued in driver; optional.

buf_finish called before every dequeue of the buffer back to userspace; the buffer is synced for the CPU, so drivers can access/modify the buffer contents; drivers may perform any operations required before userspace accesses the buffer; optional. The buffer state can be one of the following: `DONE` and `ERROR` occur while streaming is in progress, and the `PREPARED` state occurs when the queue has been canceled and all pending buffers are being returned to their default `DEQUEUED` state. Typically you only have to do something if the state is `VB2_BUF_STATE_DONE`, since in all other cases the buffer contents will be ignored anyway.

buf_cleanup called once before the buffer is freed; drivers may perform any additional cleanup; optional.

start_streaming called once to enter 'streaming' state; the driver may receive buffers with **buf_queue** callback before **start_streaming** is called; the driver gets the number of already queued buffers in `count` parameter; driver can return an error if hardware fails, in that case all buffers that have been already given by the **buf_queue** callback are to be returned by the driver by calling `vb2_buffer_done()` with `VB2_BUF_STATE_QUEUED`. If you need a minimum number of buffers before you can start streaming, then set `vb2_queue->min_buffers_needed`. If that is non-zero then **start_streaming** won't be called until at least that many buffers have been queued up by userspace.

stop_streaming called when 'streaming' state must be disabled; driver should stop any DMA transactions or wait until they finish and give back all buffers it got from `buf_queue` callback by calling `vb2_buffer_done()` with either `VB2_BUF_STATE_DONE` or `VB2_BUF_STATE_ERROR`; may use `vb2_wait_for_all_buffers()` function

buf_queue passes buffer `vb` to the driver; driver may start hardware operation on this buffer; driver should give the buffer back by calling `vb2_buffer_done()` function; it is always called after calling `VIDIOC_STREAMON()` ioctl; might be called before **start_streaming** callback if user pre-queued buffers before calling `VIDIOC_STREAMON()`.

buf_request_complete a buffer that was never queued to the driver but is associated with a queued request was canceled. The driver will have to mark associated objects in the request as completed; required if requests are supported.

Description

These operations are not called from interrupt context except where mentioned specifically.

struct `vb2_buf_ops`

driver-specific callbacks.

Definition

```
struct vb2_buf_ops {
    int (*verify_planes_array)(struct vb2_buffer *vb, const void *pb);
    void (*init_buffer)(struct vb2_buffer *vb);
    void (*fill_user_buffer)(struct vb2_buffer *vb, void *pb);
    int (*fill_vb2_buffer)(struct vb2_buffer *vb, struct vb2_plane *planes);
    void (*copy_timestamp)(struct vb2_buffer *vb, const void *pb);
};
```

Members

verify_planes_array Verify that a given user space structure contains enough planes for the buffer. This is called for each dequeued buffer.

init_buffer given a *vb2_buffer* initialize the extra data after *struct vb2_buffer*. For V4L2 this is a *struct vb2_v4l2_buffer*.

fill_user_buffer given a *vb2_buffer* fill in the userspace structure. For V4L2 this is a *struct v4l2_buffer*.

fill_vb2_buffer given a userspace structure, fill in the *vb2_buffer*. If the userspace structure is invalid, then this op will return an error.

copy_timestamp copy the timestamp from a userspace structure to the *struct vb2_buffer*.

struct vb2_queue
a videobuf queue.

Definition

```

struct vb2_queue {
    unsigned int             type;
    unsigned int            io_modes;
    struct device           *dev;
    unsigned long           dma_attrs;
    unsigned int            bidirectional:1;
    unsigned int            fileio_read_once:1;
    unsigned int            fileio_write_immediately:1;
    unsigned int            allow_zero_bytesused:1;
    unsigned int            quirk_poll_must_check_waiting_for_buffers:1;
    unsigned int            supports_requests:1;
    unsigned int            requires_requests:1;
    unsigned int            uses_qbuf:1;
    unsigned int            uses_requests:1;
    unsigned int            allow_cache_hints:1;
    unsigned int            non_coherent_mem:1;
    struct mutex            *lock;
    void *owner;
    const struct vb2_ops    *ops;
    const struct vb2_mem_ops *mem_ops;
    const struct vb2_buf_ops *buf_ops;
    void *drv_priv;
    u32 subsystem_flags;
    unsigned int            buf_struct_size;
    u32 timestamp_flags;
    gfp_t gfp_flags;
    u32 min_buffers_needed;
    struct device           *alloc_devs[VB2_MAX_PLANES];
};

```

Members

type private buffer type whose content is defined by the vb2-core caller. For example, for V4L2, it should match the types defined on enum *v4l2_buf_type*.

io_modes supported io methods (see *enum vb2_io_modes*).

dev device to use for the default allocation context if the driver doesn't fill in the **alloc_devs** array.

dma_attrs DMA attributes to use for the DMA.

bidirectional when this flag is set the DMA direction for the buffers of this queue will be overridden with DMA_BIDIRECTIONAL direction. This is useful in cases where the hardware (firmware) writes to a buffer which is mapped as read (DMA_TO_DEVICE), or reads from buffer which is mapped for write (DMA_FROM_DEVICE) in order to satisfy some internal hardware restrictions or adds a padding needed by the processing algorithm. In case the DMA mapping is not bidirectional but the hardware (firmware) trying to access the buffer (in the opposite direction) this could lead to an IOMMU protection faults.

fileio_read_once report EOF after reading the first buffer

fileio_write_immediately queue buffer after each write() call

allow_zero_bytesused allow bytesused == 0 to be passed to the driver

quirk_poll_must_check_waiting_for_buffers Return EPOLLERR at poll when QBUF has not been called. This is a vb1 idiom that has been adopted also by vb2.

supports_requests this queue supports the Request API.

requires_requests this queue requires the Request API. If this is set to 1, then supports_requests must be set to 1 as well.

uses_qbuf qbuf was used directly for this queue. Set to 1 the first time this is called. Set to 0 when the queue is canceled. If this is 1, then you cannot queue buffers from a request.

uses_requests requests are used for this queue. Set to 1 the first time a request is queued. Set to 0 when the queue is canceled. If this is 1, then you cannot queue buffers directly.

allow_cache_hints when set user-space can pass cache management hints in order to skip cache flush/invalidation on ->prepare() or/and ->finish().

non_coherent_mem when set queue will attempt to allocate buffers using non-coherent memory.

lock pointer to a mutex that protects the *struct vb2_queue*. The driver can set this to a mutex to let the v4l2 core serialize the queuing ioctls. If the driver wants to handle locking itself, then this should be set to NULL. This lock is not used by the videobuf2 core API.

owner The filehandle that 'owns' the buffers, i.e. the filehandle that called reqbufs, create_buffers or started fileio. This field is not used by the videobuf2 core API, but it allows drivers to easily associate an owner filehandle with the queue.

ops driver-specific callbacks

mem_ops memory allocator specific callbacks

buf_ops callbacks to deliver buffer information. between user-space and kernel-space.

drv_priv driver private data.

subsystem_flags Flags specific to the subsystem (V4L2/DVB/etc.). Not used by the vb2 core.

buf_struct_size size of the driver-specific buffer structure; "0" indicates the driver doesn't want to use a custom buffer structure type. In that case a subsystem-specific struct will be used (in the case of V4L2 that is sizeof(struct vb2_v4l2_buffer)). The first field of the driver-specific buffer structure must be the subsystem-specific struct (vb2_v4l2_buffer in the case of V4L2).

timestamp_flags Timestamp flags; V4L2_BUF_FLAG_TIMESTAMP_* and V4L2_BUF_FLAG_TSTAMP_SRC_*

gfp_flags additional gfp flags used when allocating the buffers. Typically this is 0, but it may be e.g. GFP_DMA or __GFP_DMA32 to force the buffer allocation to a specific memory zone.

min_buffers_needed the minimum number of buffers needed before **start_streaming** can be called. Used when a DMA engine cannot be started unless at least this number of buffers have been queued into the driver.

alloc_devs struct device memory type/allocator-specific per-plane device

bool **vb2_queue_allows_cache_hints**(struct *vb2_queue* *q)

Return true if the queue allows cache and memory consistency hints.

Parameters

struct vb2_queue *q pointer to *struct vb2_queue* with videobuf2 queue

void * **vb2_plane_vaddr**(struct *vb2_buffer* *vb, unsigned int *plane_no*)

Return a kernel virtual address of a given plane.

Parameters

struct vb2_buffer *vb pointer to *struct vb2_buffer* to which the plane in question belongs to.

unsigned int plane_no plane number for which the address is to be returned.

Description

This function returns a kernel virtual address of a given plane if such a mapping exist, NULL otherwise.

void * **vb2_plane_cookie**(struct *vb2_buffer* *vb, unsigned int *plane_no*)

Return allocator specific cookie for the given plane.

Parameters

struct vb2_buffer *vb pointer to *struct vb2_buffer* to which the plane in question belongs to.

unsigned int plane_no plane number for which the cookie is to be returned.

Description

This function returns an allocator specific cookie for a given plane if available, NULL otherwise. The allocator should provide some simple static inline function, which would convert this cookie to the allocator specific type that can be used directly by the driver to access the buffer. This can be for example physical address, pointer to scatter list or IOMMU mapping.

void **vb2_buffer_done**(struct *vb2_buffer* *vb, enum *vb2_buffer_state* state)

inform videobuf that an operation on a buffer is finished.

Parameters

struct vb2_buffer *vb pointer to *struct vb2_buffer* to be used.

enum vb2_buffer_state state state of the buffer, as defined by *enum vb2_buffer_state*. Either VB2_BUF_STATE_DONE if the operation finished successfully, VB2_BUF_STATE_ERROR if the operation finished with an error or VB2_BUF_STATE_QUEUED.

Description

This function should be called by the driver after a hardware operation on a buffer is finished and the buffer may be returned to userspace. The driver cannot use this buffer anymore until

it is queued back to it by videobuf by the means of `vb2_ops->buf_queue` callback. Only buffers previously queued to the driver by `vb2_ops->buf_queue` can be passed to this function.

While streaming a buffer can only be returned in state DONE or ERROR. The `vb2_ops->start_streaming` op can also return them in case the DMA engine cannot be started for some reason. In that case the buffers should be returned with state QUEUED to put them back into the queue.

```
void vb2_discard_done(struct vb2_queue *q)
    discard all buffers marked as DONE.
```

Parameters

struct *vb2_queue* *q pointer to *struct vb2_queue* with videobuf2 queue.

Description

This function is intended to be used with suspend/resume operations. It discards all 'done' buffers as they would be too old to be requested after resume.

Drivers must stop the hardware and synchronize with interrupt handlers and/or delayed works before calling this function to make sure no buffer will be touched by the driver and/or hardware.

```
int vb2_wait_for_all_buffers(struct vb2_queue *q)
    wait until all buffers are given back to vb2.
```

Parameters

struct *vb2_queue* *q pointer to *struct vb2_queue* with videobuf2 queue.

Description

This function will wait until all buffers that have been given to the driver by `vb2_ops->buf_queue` are given back to vb2 with `vb2_buffer_done()`. It doesn't call `vb2_ops->wait_prepare/vb2_ops->wait_finish` pair. It is intended to be called with all locks taken, for example from `vb2_ops->stop_streaming` callback.

```
void vb2_core_querybuf(struct vb2_queue *q, unsigned int index, void *pb)
    query video buffer information.
```

Parameters

struct *vb2_queue* *q pointer to *struct vb2_queue* with videobuf2 queue.

unsigned int *index* id number of the buffer.

void *pb buffer struct passed from userspace.

Description

Videobuf2 core helper to implement VIDIOC_QUERYBUF() operation. It is called internally by VB2 by an API-specific handler, like `videobuf2-v4l2.h`.

The passed buffer should have been verified.

This function fills the relevant information for the userspace.

Return

returns zero on success; an error code otherwise.

`int vb2_core_reqbufs(struct vb2_queue *q, enum vb2_memory memory, unsigned int flags, unsigned int *count)`
Initiate streaming.

Parameters

struct vb2_queue *q pointer to *struct vb2_queue* with videobuf2 queue.

enum vb2_memory memory memory type, as defined by *enum vb2_memory*.

unsigned int flags auxiliary queue/buffer management flags. Currently, the only used flag is `V4L2_MEMORY_FLAG_NON_COHERENT`.

unsigned int *count requested buffer count.

Description

Videobuf2 core helper to implement `VIDIOC_REQBUF()` operation. It is called internally by VB2 by an API-specific handler, like `videobuf2-v4l2.h`.

This function:

- 1) verifies streaming parameters passed from the userspace;
- 2) sets up the queue;
- 3) negotiates number of buffers and planes per buffer with the driver to be used during streaming;
- 4) allocates internal buffer structures (*struct vb2_buffer*), according to the agreed parameters;
- 5) for MMAP memory type, allocates actual video memory, using the memory handling/allocation routines provided during queue initialization.

If `req->count` is 0, all the memory will be freed instead.

If the queue has been allocated previously by a previous `vb2_core_reqbufs()` call and the queue is not busy, memory will be reallocated.

Return

returns zero on success; an error code otherwise.

`int vb2_core_create_bufs(struct vb2_queue *q, enum vb2_memory memory, unsigned int flags, unsigned int *count, unsigned int requested_planes, const unsigned int requested_sizes[])`
Allocate buffers and any required auxiliary structs

Parameters

struct vb2_queue *q pointer to *struct vb2_queue* with videobuf2 queue.

enum vb2_memory memory memory type, as defined by *enum vb2_memory*.

unsigned int flags auxiliary queue/buffer management flags.

unsigned int *count requested buffer count.

unsigned int requested_planes number of planes requested.

const unsigned int requested_sizes[] array with the size of the planes.

Description

Videobuf2 core helper to implement VIDIOC_CREATE_BUFS() operation. It is called internally by VB2 by an API-specific handler, like videobuf2-v4l2.h.

This function:

- 1) verifies parameter sanity;
- 2) calls the *vb2_ops->queue_setup* queue operation;
- 3) performs any necessary memory allocations.

Return

returns zero on success; an error code otherwise.

int **vb2_core_prepare_buf**(struct *vb2_queue* *q, unsigned int *index*, void *pb)
Pass ownership of a buffer from userspace to the kernel.

Parameters

struct vb2_queue *q pointer to *struct vb2_queue* with videobuf2 queue.

unsigned int *index* id number of the buffer.

void *pb buffer structure passed from userspace to *v4l2_ioctl_ops->vidioc_prepare_buf* handler in driver.

Description

Videobuf2 core helper to implement VIDIOC_PREPARE_BUF() operation. It is called internally by VB2 by an API-specific handler, like videobuf2-v4l2.h.

The passed buffer should have been verified.

This function calls *vb2_ops->buf_prepare* callback in the driver (if provided), in which driver-specific buffer initialization can be performed.

Return

returns zero on success; an error code otherwise.

int **vb2_core_qbuf**(struct *vb2_queue* *q, unsigned int *index*, void *pb, struct *media_request* *req)
Queue a buffer from userspace

Parameters

struct vb2_queue *q pointer to *struct vb2_queue* with videobuf2 queue.

unsigned int *index* id number of the buffer

void *pb buffer structure passed from userspace to *v4l2_ioctl_ops->vidioc_qbuf* handler in driver

struct media_request *req pointer to *struct media_request*, may be NULL.

Description

Videobuf2 core helper to implement VIDIOC_QBUF() operation. It is called internally by VB2 by an API-specific handler, like videobuf2-v4l2.h.

This function:

- 1) If **req** is non-NULL, then the buffer will be bound to this media request and it returns. The buffer will be prepared and queued to the driver (i.e. the next two steps) when the request itself is queued.
- 2) if necessary, calls `vb2_ops->buf_prepare` callback in the driver (if provided), in which driver-specific buffer initialization can be performed;
- 3) if streaming is on, queues the buffer in driver by the means of `vb2_ops->buf_queue` callback for processing.

Return

returns zero on success; an error code otherwise.

```
int vb2_core_dqbuf(struct vb2_queue *q, unsigned int *pindex, void *pb,
                  bool nonblocking)
    Dequeue a buffer to the userspace
```

Parameters

struct vb2_queue *q pointer to `struct vb2_queue` with videobuf2 queue

unsigned int *pindex pointer to the buffer index. May be NULL

void *pb buffer structure passed from userspace to `v4l2_ioctl_ops->vidioc_dqbuf` handler in driver.

bool nonblocking if true, this call will not sleep waiting for a buffer if no buffers ready for dequeuing are present. Normally the driver would be passing `(file->f_flags & O_NONBLOCK)` here.

Description

Videobuf2 core helper to implement `VIDIOC_DQBUF()` operation. It is called internally by VB2 by an API-specific handler, like `videobuf2-v4l2.h`.

This function:

- 1) calls `buf_finish` callback in the driver (if provided), in which driver can perform any additional operations that may be required before returning the buffer to userspace, such as cache sync,
- 2) the buffer struct members are filled with relevant information for the userspace.

Return

returns zero on success; an error code otherwise.

```
int vb2_core_streamon(struct vb2_queue *q, unsigned int type)
    Implements VB2 stream ON logic
```

Parameters

struct vb2_queue *q pointer to `struct vb2_queue` with videobuf2 queue

unsigned int type type of the queue to be started. For V4L2, this is defined by enum `v4l2_buf_type` type.

Description

Videobuf2 core helper to implement `VIDIOC_STREAMON()` operation. It is called internally by VB2 by an API-specific handler, like `videobuf2-v4l2.h`.

Return

returns zero on success; an error code otherwise.

int **vb2_core_streamoff**(struct *vb2_queue* *q, unsigned int type)
Implements VB2 stream OFF logic

Parameters

struct vb2_queue *q pointer to *struct vb2_queue* with videobuf2 queue

unsigned int type type of the queue to be started. For V4L2, this is defined by enum *v4l2_buf_type* type.

Description

Videobuf2 core helper to implement VIDIOC_STREAMOFF() operation. It is called internally by VB2 by an API-specific handler, like *videobuf2-v4l2.h*.

Return

returns zero on success; an error code otherwise.

int **vb2_core_expbuf**(struct *vb2_queue* *q, int *fd, unsigned int type, unsigned int index,
unsigned int plane, unsigned int flags)
Export a buffer as a file descriptor.

Parameters

struct vb2_queue *q pointer to *struct vb2_queue* with videobuf2 queue.

int *fd pointer to the file descriptor associated with DMABUF (set by driver).

unsigned int type buffer type.

unsigned int index id number of the buffer.

unsigned int plane index of the plane to be exported, 0 for single plane queues

unsigned int flags file flags for newly created file, as defined at *include/uapi/asm-generic/fcntl.h*. Currently, the only used flag is *O_CLOEXEC*. is supported, refer to manual of open syscall for more details.

Description

Videobuf2 core helper to implement VIDIOC_EXPBUF() operation. It is called internally by VB2 by an API-specific handler, like *videobuf2-v4l2.h*.

Return

returns zero on success; an error code otherwise.

int **vb2_core_queue_init**(struct *vb2_queue* *q)
initialize a videobuf2 queue

Parameters

struct vb2_queue *q pointer to *struct vb2_queue* with videobuf2 queue. This structure should be allocated in driver

Description

The *vb2_queue* structure should be allocated by the driver. The driver is responsible of clearing it's content and setting initial values for some required entries before calling this function.

Note: The following fields at **q** should be set before calling this function: *vb2_queue->ops*, *vb2_queue->mem_ops*, *vb2_queue->type*.

void **vb2_core_queue_release**(struct *vb2_queue* *q)
 stop streaming, release the queue and free memory

Parameters

struct vb2_queue *q pointer to *struct vb2_queue* with videobuf2 queue.

Description

This function stops streaming and performs necessary clean ups, including freeing video buffer memory. The driver is responsible for freeing the *struct vb2_queue* itself.

void **vb2_queue_error**(struct *vb2_queue* *q)
 signal a fatal error on the queue

Parameters

struct vb2_queue *q pointer to *struct vb2_queue* with videobuf2 queue.

Description

Flag that a fatal unrecoverable error has occurred and wake up all processes waiting on the queue. Polling will now set EPOLLERR and queuing and dequeuing buffers will return -EIO.

The error flag will be cleared when canceling the queue, either from *vb2_streamoff()* or *vb2_queue_release()*. Drivers should thus not call this function before starting the stream, otherwise the error flag will remain set until the queue is released when closing the device node.

int **vb2_mmap**(struct *vb2_queue* *q, struct *vm_area_struct* *vma)
 map video buffers into application address space.

Parameters

struct vb2_queue *q pointer to *struct vb2_queue* with videobuf2 queue.

struct vm_area_struct *vma pointer to *struct vm_area_struct* with the vma passed to the mmap file operation handler in the driver.

Description

Should be called from mmap file operation handler of a driver. This function maps one plane of one of the available video buffers to userspace. To map whole video memory allocated on reqbufs, this function has to be called once per each plane per each buffer previously allocated.

When the userspace application calls mmap, it passes to it an offset returned to it earlier by the means of *v4l2_ioctl_ops->vidioc_querybuf* handler. That offset acts as a "cookie", which is then used to identify the plane to be mapped.

This function finds a plane with a matching offset and a mapping is performed by the means of a provided memory operation.

The return values from this function are intended to be directly returned from the mmap handler in driver.

unsigned long **vb2_get_unmapped_area**(struct *vb2_queue* *q, unsigned long *addr*, unsigned long *len*, unsigned long *pgoff*, unsigned long *flags*)
map video buffers into application address space.

Parameters

struct vb2_queue *q pointer to *struct vb2_queue* with videobuf2 queue.

unsigned long addr memory address.

unsigned long len buffer size.

unsigned long pgoff page offset.

unsigned long flags memory flags.

Description

This function is used in noMMU platforms to propose address mapping for a given buffer. It's intended to be used as a handler for the `file_operations->get_unmapped_area` operation.

This is called by the `mmap()` syscall routines will call this to get a proposed address for the mapping, when `!CONFIG_MMU`.

`__poll_t vb2_core_poll`(struct *vb2_queue* *q, struct file *file, poll_table *wait)
implements poll syscall() logic.

Parameters

struct vb2_queue *q pointer to *struct vb2_queue* with videobuf2 queue.

struct file *file struct file argument passed to the poll file operation handler.

poll_table *wait poll_table wait argument passed to the poll file operation handler.

Description

This function implements poll file operation handler for a driver. For CAPTURE queues, if a buffer is ready to be dequeued, the userspace will be informed that the file descriptor of a video device is available for reading. For OUTPUT queues, if a buffer is ready to be dequeued, the file descriptor will be reported as available for writing.

The return values from this function are intended to be directly returned from poll handler in driver.

`size_t vb2_read`(struct *vb2_queue* *q, char __user *data, size_t count, loff_t *ppos, int nonblock)
implements read() syscall logic.

Parameters

struct vb2_queue *q pointer to *struct vb2_queue* with videobuf2 queue.

char __user *data pointed to target userspace buffer

size_t count number of bytes to read

loff_t *ppos file handle position tracking pointer

int nonblock mode selector (1 means blocking calls, 0 means nonblocking)

`size_t vb2_write(struct vb2_queue *q, const char __user *data, size_t count, loff_t *ppos, int nonblock)`
implements write() syscall logic.

Parameters

struct vb2_queue *q pointer to *struct vb2_queue* with videobuf2 queue.

const char __user *data pointed to target userspace buffer

size_t count number of bytes to write

loff_t *ppos file handle position tracking pointer

int nonblock mode selector (1 means blocking calls, 0 means nonblocking)

vb2_thread_fnc

Typedef: callback function for use with `vb2_thread`.

Syntax

```
int vb2_thread_fnc (struct vb2_buffer *vb, void *priv)
```

Parameters

struct vb2_buffer *vb pointer to *struct vb2_buffer*.

void *priv pointer to a private data.

Description

This is called whenever a buffer is dequeued in the thread.

`int vb2_thread_start(struct vb2_queue *q, vb2_thread_fnc fnc, void *priv, const char *thread_name)`
start a thread for the given queue.

Parameters

struct vb2_queue *q pointer to *struct vb2_queue* with videobuf2 queue.

vb2_thread_fnc fnc *vb2_thread_fnc* callback function.

void *priv priv pointer passed to the callback function.

const char *thread_name the name of the thread. This will be prefixed with "vb2-".

Description

This starts a thread that will queue and dequeue until an error occurs or `vb2_thread_stop()` is called.

Attention: This function should not be used for anything else but the videobuf2-dvb support. If you think you have another good use-case for this, then please contact the linux-media mailing list first.

`int vb2_thread_stop(struct vb2_queue *q)`
stop the thread for the given queue.

Parameters

struct vb2_queue *q pointer to *struct vb2_queue* with videobuf2 queue.

bool **vb2_is_streaming**(struct *vb2_queue* *q)
return streaming status of the queue.

Parameters

struct vb2_queue *q pointer to *struct vb2_queue* with videobuf2 queue.

bool **vb2_fileio_is_active**(struct *vb2_queue* *q)
return true if fileio is active.

Parameters

struct vb2_queue *q pointer to *struct vb2_queue* with videobuf2 queue.

Description

This returns true if read() or write() is used to stream the data as opposed to stream I/O. This is almost never an important distinction, except in rare cases. One such case is that using read() or write() to stream a format using V4L2_FIELD_ALTERNATE is not allowed since there is no way you can pass the field information of each buffer to/from userspace. A driver that supports this field format should check for this in the *vb2_ops->queue_setup* op and reject it if this function returns true.

bool **vb2_is_busy**(struct *vb2_queue* *q)
return busy status of the queue.

Parameters

struct vb2_queue *q pointer to *struct vb2_queue* with videobuf2 queue.

Description

This function checks if queue has any buffers allocated.

void * **vb2_get_drv_priv**(struct *vb2_queue* *q)
return driver private data associated with the queue.

Parameters

struct vb2_queue *q pointer to *struct vb2_queue* with videobuf2 queue.

void **vb2_set_plane_payload**(struct *vb2_buffer* *vb, unsigned int *plane_no*, unsigned long *size*)
set bytesused for the plane **plane_no**.

Parameters

struct vb2_buffer *vb pointer to *struct vb2_buffer* to which the plane in question belongs to.

unsigned int plane_no plane number for which payload should be set.

unsigned long size payload in bytes.

unsigned long **vb2_get_plane_payload**(struct *vb2_buffer* *vb, unsigned int *plane_no*)
get bytesused for the plane *plane_no*

Parameters

struct vb2_buffer *vb pointer to *struct vb2_buffer* to which the plane in question belongs to.

unsigned int plane_no plane number for which payload should be set.

unsigned long **vb2_plane_size**(struct *vb2_buffer* *vb, unsigned int *plane_no*)
return plane size in bytes.

Parameters

struct vb2_buffer *vb pointer to *struct vb2_buffer* to which the plane in question belongs to.

unsigned int *plane_no* plane number for which size should be returned.

bool **vb2_start_streaming_called**(struct *vb2_queue* *q)
return streaming status of driver.

Parameters

struct vb2_queue *q pointer to *struct vb2_queue* with videobuf2 queue.

void **vb2_clear_last_buffer_dequeued**(struct *vb2_queue* *q)
clear last buffer dequeued flag of queue.

Parameters

struct vb2_queue *q pointer to *struct vb2_queue* with videobuf2 queue.

struct *vb2_buffer* * **vb2_get_buffer**(struct *vb2_queue* *q, unsigned int *index*)
get a buffer from a queue

Parameters

struct vb2_queue *q pointer to *struct vb2_queue* with videobuf2 queue.

unsigned int *index* buffer index

Description

This function obtains a buffer from a queue, by its index. Keep in mind that there is no ref-counting involved in this operation, so the buffer lifetime should be taken into consideration.

bool **vb2_buffer_in_use**(struct *vb2_queue* *q, struct *vb2_buffer* *vb)
return true if the buffer is in use and the queue cannot be freed (by the means of VIDIOC_REQBUFS(0)) call.

Parameters

struct vb2_queue *q pointer to *struct vb2_queue* with videobuf2 queue.

struct vb2_buffer *vb buffer for which plane size should be returned.

int **vb2_verify_memory_type**(struct *vb2_queue* *q, enum *vb2_memory* *memory*, unsigned int *type*)
Check whether the memory type and buffer type passed to a buffer operation are compatible with the queue.

Parameters

struct vb2_queue *q pointer to *struct vb2_queue* with videobuf2 queue.

enum *vb2_memory* *memory* memory model, as defined by enum *vb2_memory*.

unsigned int *type* private buffer type whose content is defined by the vb2-core caller. For example, for V4L2, it should match the types defined on enum *v4l2_buf_type*.

bool **vb2_request_object_is_buffer**(struct *media_request_object* *obj)
return true if the object is a buffer

Parameters

struct media_request_object *obj the request object.

unsigned int **vb2_request_buffer_cnt**(struct *media_request* *req)
return the number of buffers in the request

Parameters

struct media_request *req the request.

struct **vb2_v4l2_buffer**
video buffer information for v4l2.

Definition

```
struct vb2_v4l2_buffer {
    struct vb2_buffer      vb2_buf;
    __u32 flags;
    __u32 field;
    struct v4l2_timecode   timecode;
    __u32 sequence;
    __s32 request_fd;
    bool is_held;
    struct vb2_plane       planes[VB2_MAX_PLANES];
};
```

Members

vb2_buf embedded struct *vb2_buffer*.

flags buffer informational flags.

field field order of the image in the buffer, as defined by *enum v4l2_field*.

timecode frame timecode.

sequence sequence count of this frame.

request_fd the request_fd associated with this buffer

is_held if true, then this capture buffer was held

planes plane information (userptr/fd, length, bytesused, data_offset).

Description

Should contain enough information to be able to cover all the fields of struct *v4l2_buffer* at videodev2.h.

int **vb2_find_timestamp**(const struct *vb2_queue* *q, u64 timestamp, unsigned int start_idx)
Find buffer with given timestamp in the queue

Parameters

const struct vb2_queue *q pointer to *struct vb2_queue* with videobuf2 queue.

u64 timestamp the timestamp to find.

unsigned int start_idx the start index (usually 0) in the buffer array to start searching from. Note that there may be multiple buffers with the same timestamp value, so you can restart the search by setting **start_idx** to the previously found index + 1.

Description

Returns the buffer index of the buffer with the given **timestamp**, or -1 if no buffer with **timestamp** was found.

int **vb2_reqbufs**(struct *vb2_queue* *q, struct v4l2_requestbuffers *req)
 Wrapper for *vb2_core_reqbufs()* that also verifies the memory and type values.

Parameters

struct vb2_queue *q pointer to *struct vb2_queue* with videobuf2 queue.

struct v4l2_requestbuffers *req struct v4l2_requestbuffers passed from userspace to *v4l2_ioctl_ops->vidioc_reqbufs* handler in driver.

int **vb2_create_bufs**(struct *vb2_queue* *q, struct v4l2_create_buffers *create)
 Wrapper for *vb2_core_create_bufs()* that also verifies the memory and type values.

Parameters

struct vb2_queue *q pointer to *struct vb2_queue* with videobuf2 queue.

struct v4l2_create_buffers *create creation parameters, passed from userspace to *v4l2_ioctl_ops->vidioc_create_bufs* handler in driver

int **vb2_prepare_buf**(struct *vb2_queue* *q, struct *media_device* *mdev, struct v4l2_buffer *b)
 Pass ownership of a buffer from userspace to the kernel

Parameters

struct vb2_queue *q pointer to *struct vb2_queue* with videobuf2 queue.

struct media_device *mdev pointer to *struct media_device*, may be NULL.

struct v4l2_buffer *b buffer structure passed from userspace to *v4l2_ioctl_ops->vidioc_prepare_buf* handler in driver

Description

Should be called from *v4l2_ioctl_ops->vidioc_prepare_buf* ioctl handler of a driver.

This function:

- 1) verifies the passed buffer,
- 2) calls *vb2_ops->buf_prepare* callback in the driver (if provided), in which driver-specific buffer initialization can be performed.
- 3) if **b->request_fd** is non-zero and **mdev->ops->req_queue** is set, then bind the prepared buffer to the request.

The return values from this function are intended to be directly returned from *v4l2_ioctl_ops->vidioc_prepare_buf* handler in driver.

int **vb2_qbuf**(struct *vb2_queue* *q, struct *media_device* *mdev, struct v4l2_buffer *b)
 Queue a buffer from userspace

Parameters

struct vb2_queue *q pointer to *struct vb2_queue* with videobuf2 queue.

struct media_device *mdev pointer to *struct media_device*, may be NULL.

struct v4l2_buffer *b buffer structure passed from userspace to *v4l2_ioctl_ops->vidioc_qbuf* handler in driver

Description

Should be called from *v4l2_ioctl_ops->vidioc_qbuf* handler of a driver.

This function:

- 1) verifies the passed buffer;
- 2) if **b->request_fd** is non-zero and **mdev->ops->req_queue** is set, then bind the buffer to the request.
- 3) if necessary, calls *vb2_ops->buf_prepare* callback in the driver (if provided), in which driver-specific buffer initialization can be performed;
- 4) if streaming is on, queues the buffer in driver by the means of *vb2_ops->buf_queue* callback for processing.

The return values from this function are intended to be directly returned from *v4l2_ioctl_ops->vidioc_qbuf* handler in driver.

int **vb2_expbuf**(struct *vb2_queue* *q, struct *v4l2_exportbuffer* *eb)
Export a buffer as a file descriptor

Parameters

struct vb2_queue *q pointer to *struct vb2_queue* with videobuf2 queue.

struct v4l2_exportbuffer *eb export buffer structure passed from userspace to *v4l2_ioctl_ops->vidioc_expbuf* handler in driver

Description

The return values from this function are intended to be directly returned from *v4l2_ioctl_ops->vidioc_expbuf* handler in driver.

int **vb2_dqbuf**(struct *vb2_queue* *q, struct *v4l2_buffer* *b, bool *nonblocking*)
Dequeue a buffer to the userspace

Parameters

struct vb2_queue *q pointer to *struct vb2_queue* with videobuf2 queue.

struct v4l2_buffer *b buffer structure passed from userspace to *v4l2_ioctl_ops->vidioc_dqbuf* handler in driver

bool nonblocking if true, this call will not sleep waiting for a buffer if no buffers ready for dequeuing are present. Normally the driver would be passing (*file->f_flags* & *O_NONBLOCK*) here

Description

Should be called from *v4l2_ioctl_ops->vidioc_dqbuf* ioctl handler of a driver.

This function:

- 1) verifies the passed buffer;

- 2) calls `vb2_ops->buf_finish` callback in the driver (if provided), in which driver can perform any additional operations that may be required before returning the buffer to userspace, such as cache sync;
- 3) the buffer struct members are filled with relevant information for the userspace.

The return values from this function are intended to be directly returned from `v4l2_ioctl_ops->vidioc_dqbuf` handler in driver.

```
int vb2_streamon(struct vb2_queue *q, enum v4l2_buf_type type)
    start streaming
```

Parameters

struct vb2_queue *q pointer to `struct vb2_queue` with videobuf2 queue.

enum v4l2_buf_type type type argument passed from userspace to `vidioc_streamon` handler, as defined by enum `v4l2_buf_type`.

Description

Should be called from `v4l2_ioctl_ops->vidioc_streamon` handler of a driver.

This function:

- 1) verifies current state
- 2) passes any previously queued buffers to the driver and starts streaming

The return values from this function are intended to be directly returned from `v4l2_ioctl_ops->vidioc_streamon` handler in the driver.

```
int vb2_streamoff(struct vb2_queue *q, enum v4l2_buf_type type)
    stop streaming
```

Parameters

struct vb2_queue *q pointer to `struct vb2_queue` with videobuf2 queue.

enum v4l2_buf_type type type argument passed from userspace to `vidioc_streamoff` handler

Description

Should be called from `vidioc_streamoff` handler of a driver.

This function:

- 1) verifies current state,
- 2) stop streaming and dequeues any queued buffers, including those previously passed to the driver (after waiting for the driver to finish).

This call can be used for pausing playback. The return values from this function are intended to be directly returned from `vidioc_streamoff` handler in the driver

```
int vb2_queue_init(struct vb2_queue *q)
    initialize a videobuf2 queue
```

Parameters

struct vb2_queue *q pointer to `struct vb2_queue` with videobuf2 queue.

Description

The `vb2_queue` structure should be allocated by the driver. The driver is responsible of clearing it's content and setting initial values for some required entries before calling this function. `q->ops`, `q->mem_ops`, `q->type` and `q->io_modes` are mandatory. Please refer to the [struct vb2_queue](#) description in `include/media/videobuf2-core.h` for more information.

```
int vb2_queue_init_name(struct vb2_queue *q, const char *name)
    initialize a videobuf2 queue with a name
```

Parameters

struct vb2_queue *q pointer to [struct vb2_queue](#) with videobuf2 queue.
const char *name the queue name

Description

This function initializes the `vb2_queue` exactly like [vb2_queue_init\(\)](#), and additionally sets the queue name. The queue name is used for logging purpose, and should uniquely identify the queue within the context of the device it belongs to. This is useful to attribute kernel log messages to the right queue for m2m devices or other devices that handle multiple queues.

```
void vb2_queue_release(struct vb2_queue *q)
    stop streaming, release the queue and free memory
```

Parameters

struct vb2_queue *q pointer to [struct vb2_queue](#) with videobuf2 queue.

Description

This function stops streaming and performs necessary clean ups, including freeing video buffer memory. The driver is responsible for freeing the `vb2_queue` structure itself.

```
int vb2_queue_change_type(struct vb2_queue *q, unsigned int type)
    change the type of an inactive vb2_queue
```

Parameters

struct vb2_queue *q pointer to [struct vb2_queue](#) with videobuf2 queue.
unsigned int type the type to change to (`V4L2_BUF_TYPE_VIDEO_*`)

Description

This function changes the type of the `vb2_queue`. This is only possible if the queue is not busy (i.e. no buffers have been allocated).

[vb2_queue_change_type\(\)](#) can be used to support multiple buffer types using the same queue. The driver can implement `v4l2_ioctl_ops.vidioc_reqbufs` and `v4l2_ioctl_ops.vidioc_create_bufs` functions and call [vb2_queue_change_type\(\)](#) before calling `vb2_ioctl_reqbufs()` or `vb2_ioctl_create_bufs()`, and thus "lock" the buffer type until the buffers have been released.

```
__poll_t vb2_poll(struct vb2_queue *q, struct file *file, poll_table *wait)
    implements poll userspace operation
```

Parameters

struct vb2_queue *q pointer to [struct vb2_queue](#) with videobuf2 queue.
struct file *file file argument passed to the poll file operation handler
poll_table *wait wait argument passed to the poll file operation handler

Description

This function implements poll file operation handler for a driver. For CAPTURE queues, if a buffer is ready to be dequeued, the userspace will be informed that the file descriptor of a video device is available for reading. For OUTPUT queues, if a buffer is ready to be dequeued, the file descriptor will be reported as available for writing.

If the driver uses *struct v4l2_fh*, then *vb2_poll()* will also check for any pending events.

The return values from this function are intended to be directly returned from poll handler in driver.

```
void vb2_video_unregister_device(struct video_device *vdev)
    unregister the video device and release queue
```

Parameters

struct video_device *vdev pointer to *struct video_device*

Description

If the driver uses *vb2_fop_release()/vb2_fop_release()*, then it should use *vb2_video_unregister_device()* instead of *video_unregister_device()*.

This function will call *video_unregister_device()* and then release the *vb2_queue* if streaming is in progress. This will stop streaming and this will simplify the unbind sequence since after this call all subdevs will have stopped streaming as well.

```
void vb2_ops_wait_prepare(struct vb2_queue *vq)
    helper function to lock a struct vb2_queue
```

Parameters

struct vb2_queue *vq pointer to *struct vb2_queue*

Description

..note:: only use if *vq->lock* is non-NULL.

```
void vb2_ops_wait_finish(struct vb2_queue *vq)
    helper function to unlock a struct vb2_queue
```

Parameters

struct vb2_queue *vq pointer to *struct vb2_queue*

Description

..note:: only use if *vq->lock* is non-NULL.

```
struct vb2_vmarea_handler
    common vma refcount tracking handler.
```

Definition

```
struct vb2_vmarea_handler {
    refcount_t *refcount;
    void (*put)(void *arg);
    void *arg;
};
```

Members

refcount pointer to `refcount_t` entry in the buffer.

put callback to function that decreases buffer `refcount`.

arg argument for **put** callback.

2.2.16 V4L2 DV Timings functions

struct `v4l2_fract` **v4l2_calc_timeperframe**(const struct `v4l2_dv_timings` **t*)
helper function to calculate timeperframe based `v4l2_dv_timings` fields.

Parameters

const struct v4l2_dv_timings *t Timings for the video mode.

Description

Calculates the expected timeperframe using the pixel clock value and horizontal/vertical measures. This means that `v4l2_dv_timings` structure must be correctly and fully filled.

v4l2_check_dv_timings_fnc

Typedef: timings check callback

Syntax

```
bool v4l2_check_dv_timings_fnc (const struct v4l2_dv_timings *t, void *handle)
```

Parameters

const struct v4l2_dv_timings *t the `v4l2_dv_timings` struct.

void *handle a handle from the driver.

Description

Returns true if the given timings are valid.

```
bool v4l2_valid_dv_timings(const struct v4l2_dv_timings *t, const struct v4l2_dv_timings_cap *cap, v4l2_check_dv_timings_fnc fnc, void *fnc_handle)
```

are these timings valid?

Parameters

const struct v4l2_dv_timings *t the `v4l2_dv_timings` struct.

const struct v4l2_dv_timings_cap *cap the `v4l2_dv_timings_cap` capabilities.

v4l2_check_dv_timings_fnc fnc callback to check if this timing is OK. May be NULL.

void *fnc_handle a handle that is passed on to **fnc**.

Description

Returns true if the given `dv_timings` struct is supported by the hardware capabilities and the callback function (if non-NULL), returns false otherwise.

```
int v4l2_enum_dv_timings_cap(struct v4l2_enum_dv_timings *t, const struct v4l2_dv_timings_cap *cap, v4l2_check_dv_timings_fnc fnc, void *fnc_handle)
```

Helper function to enumerate possible DV timings based on capabilities

Parameters

struct v4l2_enum_dv_timings *t the v4l2_enum_dv_timings struct.

const struct v4l2_dv_timings_cap *cap the v4l2_dv_timings_cap capabilities.

v4l2_check_dv_timings_fnc fnc callback to check if this timing is OK. May be NULL.

void *fnc_handle a handle that is passed on to **fnc**.

Description

This enumerates dv_timings using the full list of possible CEA-861 and DMT timings, filtering out any timings that are not supported based on the hardware capabilities and the callback function (if non-NULL).

If a valid timing for the given index is found, it will fill in **t** and return 0, otherwise it returns -EINVAL.

```
bool v4l2_find_dv_timings_cap(struct v4l2_dv_timings *t,
                             const struct v4l2_dv_timings_cap *cap,
                             unsigned pclock_delta,
                             v4l2_check_dv_timings_fnc fnc, void *fnc_handle)
```

Find the closest timings struct

Parameters

struct v4l2_dv_timings *t the v4l2_enum_dv_timings struct.

const struct v4l2_dv_timings_cap *cap the v4l2_dv_timings_cap capabilities.

unsigned pclock_delta maximum delta between t->pixelclock and the timing struct under consideration.

v4l2_check_dv_timings_fnc fnc callback to check if a given timings struct is OK. May be NULL.

void *fnc_handle a handle that is passed on to **fnc**.

Description

This function tries to map the given timings to an entry in the full list of possible CEA-861 and DMT timings, filtering out any timings that are not supported based on the hardware capabilities and the callback function (if non-NULL).

On success it will fill in **t** with the found timings and it returns true. On failure it will return false.

```
bool v4l2_find_dv_timings_cea861_vic(struct v4l2_dv_timings *t, u8 vic)
    find timings based on CEA-861 VIC
```

Parameters

struct v4l2_dv_timings *t the timings data.

u8 vic CEA-861 VIC code

Description

On success it will fill in **t** with the found timings and it returns true. On failure it will return false.

bool v4l2_match_dv_timings(const struct v4l2_dv_timings **measured*, const struct v4l2_dv_timings **standard*, unsigned *pclock_delta*, bool *match_reduced_fps*)
do two timings match?

Parameters

const struct v4l2_dv_timings *measured the measured timings data.

const struct v4l2_dv_timings *standard the timings according to the standard.

unsigned pclock_delta maximum delta in Hz between standard->pixelclock and the measured timings.

bool match_reduced_fps if true, then fail if V4L2_DV_FL_REDUCED_FPS does not match.

Description

Returns true if the two timings match, returns false otherwise.

void v4l2_print_dv_timings(const char **dev_prefix*, const char **prefix*, const struct v4l2_dv_timings **t*, bool *detailed*)
log the contents of a dv_timings struct

Parameters

const char *dev_prefix device prefix for each log line.

const char *prefix additional prefix for each log line, may be NULL.

const struct v4l2_dv_timings *t the timings data.

bool detailed if true, give a detailed log.

bool v4l2_detect_cvt(unsigned *frame_height*, unsigned *hfreq*, unsigned *vsync*, unsigned *active_width*, u32 *polarities*, bool *interlaced*, struct v4l2_dv_timings **fmt*)
detect if the given timings follow the CVT standard

Parameters

unsigned frame_height the total height of the frame (including blanking) in lines.

unsigned hfreq the horizontal frequency in Hz.

unsigned vsync the height of the vertical sync in lines.

unsigned active_width active width of image (does not include blanking). This information is needed only in case of version 2 of reduced blanking. In other cases, this parameter does not have any effect on timings.

u32 polarities the horizontal and vertical polarities (same as struct v4l2_bt_timings polarities).

bool interlaced if this flag is true, it indicates interlaced format

struct v4l2_dv_timings *fmt the resulting timings.

Description

This function will attempt to detect if the given values correspond to a valid CVT format. If so, then it will return true, and *fmt* will be filled in with the found CVT timings.

bool **v4l2_detect_gtf**(unsigned *frame_height*, unsigned *hfreq*, unsigned *vsync*,
u32 *polarities*, bool *interlaced*, struct v4l2_fract *aspect*, struct
v4l2_dv_timings **fmt*)
detect if the given timings follow the GTF standard

Parameters

unsigned **frame_height** the total height of the frame (including blanking) in lines.

unsigned **hfreq** the horizontal frequency in Hz.

unsigned **vsync** the height of the vertical sync in lines.

u32 **polarities** the horizontal and vertical polarities (same as struct v4l2_bt_timings polarities).

bool **interlaced** if this flag is true, it indicates interlaced format

struct v4l2_fract **aspect** preferred aspect ratio. GTF has no method of determining the aspect ratio in order to derive the image width from the image height, so it has to be passed explicitly. Usually the native screen aspect ratio is used for this. If it is not filled in correctly, then 16:9 will be assumed.

struct v4l2_dv_timings ***fmt** the resulting timings.

Description

This function will attempt to detect if the given values correspond to a valid GTF format. If so, then it will return true, and *fmt* will be filled in with the found GTF timings.

struct v4l2_fract **v4l2_calc_aspect_ratio**(u8 *hor_landscape*, u8 *vert_portrait*)
calculate the aspect ratio based on bytes 0x15 and 0x16 from the EDID.

Parameters

u8 **hor_landscape** byte 0x15 from the EDID.

u8 **vert_portrait** byte 0x16 from the EDID.

Description

Determines the aspect ratio from the EDID. See VESA Enhanced EDID standard, release A, rev 2, section 3.6.2: "Horizontal and Vertical Screen Size or Aspect Ratio"

struct v4l2_fract **v4l2_dv_timings_aspect_ratio**(const struct v4l2_dv_timings **t*)
calculate the aspect ratio based on the v4l2_dv_timings information.

Parameters

const struct v4l2_dv_timings ***t** the timings data.

bool **can_reduce_fps**(struct v4l2_bt_timings **bt*)
check if conditions for reduced fps are true.

Parameters

struct v4l2_bt_timings ***bt** v4l2 timing structure

Description

For different timings reduced fps is allowed if the following conditions are met:

- For CVT timings: if reduced blanking v2 (*vsync* == 8) is true.

- For CEA861 timings: if `V4L2_DV_FL_CAN_REDUCE_FPS` flag is true.

struct **v4l2_hdmi_colorimetry**

describes the HDMI colorimetry information

Definition

```
struct v4l2_hdmi_colorimetry {
    enum v4l2_colorspace colorspace;
    enum v4l2_ycbcr_encoding ycbcr_enc;
    enum v4l2_quantization quantization;
    enum v4l2_xfer_func xfer_func;
};
```

Members

colorspace *enum v4l2_colorspace*, the colorspace

ycbcr_enc *enum v4l2_ycbcr_encoding*, Y'CbCr encoding

quantization *enum v4l2_quantization*, colorspace quantization

xfer_func *enum v4l2_xfer_func*, colorspace transfer function

2.2.17 V4L2 flash functions and data structures

struct **v4l2_flash_ctrl_data**

flash control initialization data, filled basing on the features declared by the LED flash class driver in the `v4l2_flash_config`

Definition

```
struct v4l2_flash_ctrl_data {
    struct v4l2_ctrl_config config;
    u32 cid;
};
```

Members

config initialization data for a control

cid contains v4l2 flash control id if the config field was initialized, 0 otherwise

struct **v4l2_flash_ops**

V4L2 flash operations

Definition

```
struct v4l2_flash_ops {
    int (*external_strobe_set)(struct v4l2_flash *v4l2_flash, bool enable);
    enum led_brightness (*intensity_to_led_brightness) (struct v4l2_flash *v4l2_flash,
    ↪ s32 intensity);
    s32 (*led_brightness_to_intensity) (struct v4l2_flash *v4l2_flash, enum led_
    ↪ brightness);
};
```

Members

external_strobe_set Setup strobing the flash by hardware pin state assertion.

intensity_to_led_brightness Convert intensity to brightness in a device specific manner

led_brightness_to_intensity convert brightness to intensity in a device specific manner.

struct **v4l2_flash_config**

V4L2 Flash sub-device initialization data

Definition

```
struct v4l2_flash_config {
    char dev_name[32];
    struct led_flash_setting intensity;
    u32 flash_faults;
    unsigned int has_external_strobe:1;
};
```

Members

dev_name the name of the media entity, unique in the system

intensity non-flash strobe constraints for the LED

flash_faults bitmask of flash faults that the LED flash class device can report; corresponding LED_FAULT* bit definitions are available in the header file <linux/led-class-flash.h>

has_external_strobe external strobe capability

struct **v4l2_flash**

Flash sub-device context

Definition

```
struct v4l2_flash {
    struct led_classdev_flash *fled_cdev;
    struct led_classdev *iled_cdev;
    const struct v4l2_flash_ops *ops;
    struct v4l2_subdev sd;
    struct v4l2_ctrl_handler hdl;
    struct v4l2_ctrl **ctrls;
};
```

Members

fled_cdev LED flash class device controlled by this sub-device

iled_cdev LED class device representing indicator LED associated with the LED flash class device

ops V4L2 specific flash ops

sd V4L2 sub-device

hdl flash controls handler

ctrls array of pointers to controls, whose values define the sub-device state

struct *v4l2_flash* * **v4l2_subdev_to_v4l2_flash**(struct *v4l2_subdev* *sd)

Returns a *struct v4l2_flash* from the *struct v4l2_subdev* embedded on it.

Parameters

struct *v4l2_subdev* *sd pointer to *struct v4l2_subdev*

`struct v4l2_flash * v4l2_ctrl_to_v4l2_flash(struct v4l2_ctrl *c)`

Returns a `struct v4l2_flash` from the `struct v4l2_ctrl` embedded on it.

Parameters

struct v4l2_ctrl *c pointer to `struct v4l2_ctrl`

`struct v4l2_flash * v4l2_flash_init(struct device *dev, struct fwnode_handle *fwn, struct led_classdev_flash *fled_cdev, const struct v4l2_flash_ops *ops, struct v4l2_flash_config *config)`

initialize V4L2 flash led sub-device

Parameters

struct device *dev flash device, e.g. an I2C device

struct fwnode_handle *fwn fwnode_handle of the LED, may be NULL if the same as device's

struct led_classdev_flash *fled_cdev LED flash class device to wrap

const struct v4l2_flash_ops *ops V4L2 Flash device ops

struct v4l2_flash_config *config initialization data for V4L2 Flash sub-device

Description

Create V4L2 Flash sub-device wrapping given LED subsystem device. The ops pointer is stored by the V4L2 flash framework. No references are held to config nor its contents once this function has returned.

Return

A valid pointer, or, when an error occurs, the return value is encoded using `ERR_PTR()`. Use `IS_ERR()` to check and `PTR_ERR()` to obtain the numeric return value.

`struct v4l2_flash * v4l2_flash_indicator_init(struct device *dev, struct fwnode_handle *fwn, struct led_classdev *iled_cdev, struct v4l2_flash_config *config)`

initialize V4L2 indicator sub-device

Parameters

struct device *dev flash device, e.g. an I2C device

struct fwnode_handle *fwn fwnode_handle of the LED, may be NULL if the same as device's

struct led_classdev *iled_cdev LED flash class device representing the indicator LED

struct v4l2_flash_config *config initialization data for V4L2 Flash sub-device

Description

Create V4L2 Flash sub-device wrapping given LED subsystem device. The ops pointer is stored by the V4L2 flash framework. No references are held to config nor its contents once this function has returned.

Return

A valid pointer, or, when an error occurs, the return value is encoded using `ERR_PTR()`. Use `IS_ERR()` to check and `PTR_ERR()` to obtain the numeric return value.

void **v4l2_flash_release**(struct *v4l2_flash* **v4l2_flash*)
 release V4L2 Flash sub-device

Parameters

struct v4l2_flash *v4l2_flash the V4L2 Flash sub-device to release

Description

Release V4L2 Flash sub-device.

2.2.18 V4L2 Media Controller functions and data structures

int **v4l2_mc_create_media_graph**(struct *media_device* **mdev*)
 create Media Controller links at the graph.

Parameters

struct media_device *mdev pointer to the *media_device* struct.

Description

Add links between the entities commonly found on PC customer's hardware at the V4L2 side: camera sensors, audio and video PLL-IF decoders, tuners, analog TV decoder and I/O entities (video, VBI and Software Defined Radio).

Note: Webcams are modelled on a very simple way: the sensor is connected directly to the I/O entity. All dirty details, like scaler and crop HW are hidden. While such mapping is enough for v4l2 interface centric PC-consumer's hardware, V4L2 subdev centric camera hardware should not use this routine, as it will not build the right graph.

int **v4l_enable_media_source**(struct *video_device* **vdev*)
 Hold media source for exclusive use if free

Parameters

struct video_device *vdev pointer to *struct video_device*

Description

This interface calls `enable_source` handler to determine if media source is free for use. The `enable_source` handler is responsible for checking if the media source is free and start a pipeline between the media source and the media entity associated with the video device. This interface should be called from v4l2-core and dvb-core interfaces that change the source configuration.

Return

returns zero on success or a negative error code.

void **v4l_disable_media_source**(struct *video_device* **vdev*)
 Release media source

Parameters

struct video_device *vdev pointer to *struct video_device*

Description

This interface calls `disable_source` handler to release the media source. The `disable_source` handler stops the active media pipeline between the media source and the media entity associated with the video device.

Return

returns zero on success or a negative error code.

```
int v4l2_create_fwnode_links_to_pad(struct v4l2_subdev *src_sd, struct media_pad *sink, u32 flags)
```

Create fwnode-based links from a source subdev to a sink subdev pad.

Parameters

struct v4l2_subdev *src_sd pointer to a source subdev

struct media_pad *sink pointer to a subdev sink pad

u32 flags the link flags

Description

This function searches for fwnode endpoint connections from a source subdevice to a single sink pad, and if suitable connections are found, translates them into media links to that pad. The function can be called by the sink subdevice, in its v4l2-async notifier subdev bound callback, to create links from a bound source subdevice.

The **flags** argument specifies the link flags. The caller shall ensure that the flags are valid regardless of the number of links that may be created. For instance, setting the `MEDIA_LNK_FL_ENABLED` flag will cause all created links to be enabled, which isn't valid if more than one link is created.

Note: Any sink subdevice that calls this function must implement the `.get_fwnode_pad` media operation in order to verify endpoints passed to the sink are owned by the sink.

Return 0 on success or a negative error code on failure.

```
int v4l2_create_fwnode_links(struct v4l2_subdev *src_sd, struct v4l2_subdev *sink_sd)
```

Create fwnode-based links from a source subdev to a sink subdev.

Parameters

struct v4l2_subdev *src_sd pointer to a source subdevice

struct v4l2_subdev *sink_sd pointer to a sink subdevice

Description

This function searches for any and all fwnode endpoint connections between source and sink subdevices, and translates them into media links. The function can be called by the sink subdevice, in its v4l2-async notifier subdev bound callback, to create all links from a bound source subdevice.

Note: Any sink subdevice that calls this function must implement the `.get_fwnode_pad` media operation in order to verify endpoints passed to the sink are owned by the sink.

Return 0 on success or a negative error code on failure.

int **v4l2_pipeline_pm_get**(struct *media_entity* *entity)

Increase the use count of a pipeline

Parameters

struct media_entity *entity The root entity of a pipeline

Description

Update the use count of all entities in the pipeline and power entities on.

This function is intended to be called in video node open. It uses *struct media_entity*.use_count to track the power status. The use of this function should be paired with *v4l2_pipeline_link_notify()*.

Return 0 on success or a negative error code on failure.

void **v4l2_pipeline_pm_put**(struct *media_entity* *entity)

Decrease the use count of a pipeline

Parameters

struct media_entity *entity The root entity of a pipeline

Description

Update the use count of all entities in the pipeline and power entities off.

This function is intended to be called in video node release. It uses *struct media_entity*.use_count to track the power status. The use of this function should be paired with *v4l2_pipeline_link_notify()*.

int **v4l2_pipeline_link_notify**(struct *media_link* *link, u32 flags, unsigned int notification)

Link management notification callback

Parameters

struct media_link *link The link

u32 flags New link flags that will be applied

unsigned int notification The link's state change notification type (MEDIA_DEV_NOTIFY_*)

Description

React to link management on powered pipelines by updating the use count of all entities in the source and sink sides of the link. Entities are powered on or off accordingly. The use of this function should be paired with *v4l2_pipeline_pm_{get,put}()*.

Return 0 on success or a negative error code on failure. Powering entities off is assumed to never fail. This function will not fail for disconnection events.

2.2.19 V4L2 Media Bus functions and data structures

enum **v4l2_mbus_type**
media bus type

Constants

V4L2_MBUS_UNKNOWN unknown bus type, no V4L2 mediabus configuration

V4L2_MBUS_PARALLEL parallel interface with hsync and vsync

V4L2_MBUS_BT656 parallel interface with embedded synchronisation, can also be used for BT.1120

V4L2_MBUS_CSI1 MIPI CSI-1 serial interface

V4L2_MBUS_CCP2 CCP2 (Compact Camera Port 2)

V4L2_MBUS_CSI2_DPHY MIPI CSI-2 serial interface, with D-PHY

V4L2_MBUS_CSI2_CPHY MIPI CSI-2 serial interface, with C-PHY

V4L2_MBUS_INVALID invalid bus type (keep as last)

struct **v4l2_mbus_config**
media bus configuration

Definition

```
struct v4l2_mbus_config {
    enum v4l2_mbus_type type;
    unsigned int flags;
};
```

Members

type in: interface type

flags in / out: configuration flags, depending on **type**

void **v4l2_fill_pix_format**(struct *v4l2_pix_format* **pix_fmt*, const struct *v4l2_mbus_framefmt* **mbus_fmt*)
Ancillary routine that fills a *struct v4l2_pix_format* fields from a struct *v4l2_mbus_framefmt*.

Parameters

struct v4l2_pix_format *pix_fmt pointer to *struct v4l2_pix_format* to be filled

const struct v4l2_mbus_framefmt *mbus_fmt pointer to struct *v4l2_mbus_framefmt* to be used as model

void **v4l2_fill_mbus_format**(struct *v4l2_mbus_framefmt* **mbus_fmt*, const struct *v4l2_pix_format* **pix_fmt*, u32 *code*)
Ancillary routine that fills a struct *v4l2_mbus_framefmt* from a *struct v4l2_pix_format* and a data format code.

Parameters

struct v4l2_mbus_framefmt *mbus_fmt pointer to struct *v4l2_mbus_framefmt* to be filled

const struct v4l2_pix_format *pix_fmt pointer to *struct v4l2_pix_format* to be used as model

u32 code data format code (from enum `v4l2_mbus_pixelcode`)

```
void v4l2_fill_pix_format_mplane(struct v4l2_pix_format_mplane *pix_mp_fmt, const
                               struct v4l2_mbus_framefmt *mbus_fmt)
```

Ancillary routine that fills a `struct v4l2_pix_format_mplane` fields from a media bus structure.

Parameters

struct v4l2_pix_format_mplane *pix_mp_fmt pointer to `struct v4l2_pix_format_mplane` to be filled

const struct v4l2_mbus_framefmt *mbus_fmt pointer to `struct v4l2_mbus_framefmt` to be used as model

```
void v4l2_fill_mbus_format_mplane(struct v4l2_mbus_framefmt *mbus_fmt, const
                                  struct v4l2_pix_format_mplane *pix_mp_fmt)
```

Ancillary routine that fills a `struct v4l2_mbus_framefmt` from a `struct v4l2_pix_format_mplane`.

Parameters

struct v4l2_mbus_framefmt *mbus_fmt pointer to `struct v4l2_mbus_framefmt` to be filled

const struct v4l2_pix_format_mplane *pix_mp_fmt pointer to `struct v4l2_pix_format_mplane` to be used as model

2.2.20 V4L2 Memory to Memory functions and data structures

struct v4l2_m2m_ops

mem-to-mem device driver callbacks

Definition

```
struct v4l2_m2m_ops {
    void (*device_run)(void *priv);
    int (*job_ready)(void *priv);
    void (*job_abort)(void *priv);
};
```

Members

device_run required. Begin the actual job (transaction) inside this callback. The job does NOT have to end before this callback returns (and it will be the usual case). When the job finishes, `v4l2_m2m_job_finish()` or `v4l2_m2m_buf_done_and_job_finish()` has to be called.

job_ready optional. Should return 0 if the driver does not have a job fully prepared to run yet (i.e. it will not be able to finish a transaction without sleeping). If not provided, it will be assumed that one source and one destination buffer are all that is required for the driver to perform one full transaction. This method may not sleep.

job_abort optional. Informs the driver that it has to abort the currently running transaction as soon as possible (i.e. as soon as it can stop the device safely; e.g. in the next interrupt handler), even if the transaction would not have been finished by then. After the driver performs the necessary steps, it has to call `v4l2_m2m_job_finish()` or

`v4l2_m2m_buf_done_and_job_finish()` as if the transaction ended normally. This function does not have to (and will usually not) wait until the device enters a state when it can be stopped.

struct **v4l2_m2m_queue_ctx**

represents a queue for buffers ready to be processed

Definition

```
struct v4l2_m2m_queue_ctx {
    struct vb2_queue      q;
    struct list_head      rdy_queue;
    spinlock_t rdy_spinlock;
    u8 num_rdy;
    bool buffered;
};
```

Members

q pointer to struct `vb2_queue`

rdy_queue List of V4L2 mem-to-mem queues

rdy_spinlock spin lock to protect the struct usage

num_rdy number of buffers ready to be processed

buffered is the queue buffered?

Description

Queue for buffers ready to be processed as soon as this instance receives access to the device.

struct **v4l2_m2m_ctx**

Memory to memory context structure

Definition

```
struct v4l2_m2m_ctx {
    struct mutex          *q_lock;
    bool new_frame;
    bool is_draining;
    struct vb2_v4l2_buffer *last_src_buf;
    bool next_buf_last;
    bool has_stopped;
    struct v4l2_m2m_dev   *m2m_dev;
    struct v4l2_m2m_queue_ctx cap_q_ctx;
    struct v4l2_m2m_queue_ctx out_q_ctx;
    struct list_head      queue;
    unsigned long         job_flags;
    wait_queue_head_t finished;
    void *priv;
};
```

Members

q_lock struct mutex lock

new_frame valid in the `device_run` callback: if true, then this starts a new frame; if false, then this is a new slice for an existing frame. This is always true unless

V4L2_BUF_CAP_SUPPORTS_M2M_HOLD_CAPTURE_BUF is set, which indicates slicing support.

is_draining indicates device is in draining phase

last_src_buf indicate the last source buffer for draining

next_buf_last next capture queud buffer will be tagged as last

has_stopped indicate the device has been stopped

m2m_dev opaque pointer to the internal data to handle M2M context

cap_q_ctx Capture (output to memory) queue context

out_q_ctx Output (input from memory) queue context

queue List of memory to memory contexts

job_flags Job queue flags, used internally by v4l2-mem2mem.c: TRANS_QUEUED, TRANS_RUNNING and TRANS_ABORT.

finished Wait queue used to signalize when a job queue finished.

priv Instance private data

Description

The memory to memory context is specific to a file handle, NOT to e.g. a device.

struct **v4l2_m2m_buffer**

Memory to memory buffer

Definition

```
struct v4l2_m2m_buffer {
    struct vb2_v4l2_buffer vb;
    struct list_head list;
};
```

Members

vb pointer to struct *vb2_v4l2_buffer*

list list of m2m buffers

void * **v4l2_m2m_get_curr_priv**(struct v4l2_m2m_dev **m2m_dev*)

return driver private data for the currently running instance or NULL if no instance is running

Parameters

struct v4l2_m2m_dev *m2m_dev opaque pointer to the internal data to handle M2M context

struct *vb2_queue* * **v4l2_m2m_get_vq**(struct *v4l2_m2m_ctx* **m2m_ctx*, enum *v4l2_buf_type* *type*)

return vb2_queue for the given type

Parameters

struct v4l2_m2m_ctx *m2m_ctx m2m context assigned to the instance given by struct *v4l2_m2m_ctx*

enum v4l2_buf_type type type of the V4L2 buffer, as defined by enum *v4l2_buf_type*

void **v4l2_m2m_try_schedule**(struct *v4l2_m2m_ctx* **m2m_ctx*)

check whether an instance is ready to be added to the pending job queue and add it if so.

Parameters

struct v4l2_m2m_ctx **m2m_ctx* m2m context assigned to the instance given by struct *v4l2_m2m_ctx*

Description

There are three basic requirements an instance has to meet to be able to run: 1) at least one source buffer has to be queued, 2) at least one destination buffer has to be queued, 3) streaming has to be on.

If a queue is buffered (for example a decoder hardware ringbuffer that has to be drained before doing streamoff), allow scheduling without v4l2 buffers on that queue.

There may also be additional, custom requirements. In such case the driver should supply a custom callback (*job_ready* in *v4l2_m2m_ops*) that should return 1 if the instance is ready. An example of the above could be an instance that requires more than one src/dst buffer per transaction.

void **v4l2_m2m_job_finish**(struct *v4l2_m2m_dev* **m2m_dev*, struct *v4l2_m2m_ctx* **m2m_ctx*)

inform the framework that a job has been finished and have it clean up

Parameters

struct v4l2_m2m_dev **m2m_dev* opaque pointer to the internal data to handle M2M context

struct v4l2_m2m_ctx **m2m_ctx* m2m context assigned to the instance given by struct *v4l2_m2m_ctx*

Description

Called by a driver to yield back the device after it has finished with it. Should be called as soon as possible after reaching a state which allows other instances to take control of the device.

This function has to be called only after *v4l2_m2m_ops->device_run* callback has been called on the driver. To prevent recursion, it should not be called directly from the *v4l2_m2m_ops->device_run* callback though.

void **v4l2_m2m_buf_done_and_job_finish**(struct *v4l2_m2m_dev* **m2m_dev*, struct *v4l2_m2m_ctx* **m2m_ctx*, enum *vb2_buffer_state* *state*)

return source/destination buffers with *state* and inform the framework that a job has been finished and have it clean up

Parameters

struct v4l2_m2m_dev **m2m_dev* opaque pointer to the internal data to handle M2M context

struct v4l2_m2m_ctx **m2m_ctx* m2m context assigned to the instance given by struct *v4l2_m2m_ctx*

enum vb2_buffer_state *state* vb2 buffer state passed to *v4l2_m2m_buf_done*().

Description

Drivers that set *V4L2_BUF_CAP_SUPPORTS_M2M_HOLD_CAPTURE_BUF* must use this function instead of *job_finish*() to take held buffers into account. It is optional for other drivers.

This function removes the source buffer from the ready list and returns it with the given state. The same is done for the destination buffer, unless it is marked 'held'. In that case the buffer is kept on the ready list.

After that the job is finished (see `job_finish()`).

This allows for multiple output buffers to be used to fill in a single capture buffer. This is typically used by stateless decoders where multiple e.g. H.264 slices contribute to a single decoded frame.

void **v4l2_m2m_clear_state**(struct *v4l2_m2m_ctx* **m2m_ctx*)
clear encoding/decoding state

Parameters

struct *v4l2_m2m_ctx* **m2m_ctx* m2m context assigned to the instance given by struct *v4l2_m2m_ctx*

void **v4l2_m2m_mark_stopped**(struct *v4l2_m2m_ctx* **m2m_ctx*)
set current encoding/decoding state as stopped

Parameters

struct *v4l2_m2m_ctx* **m2m_ctx* m2m context assigned to the instance given by struct *v4l2_m2m_ctx*

bool **v4l2_m2m_dst_buf_is_last**(struct *v4l2_m2m_ctx* **m2m_ctx*)
return the current encoding/decoding session draining management state of next queued capture buffer

Parameters

struct *v4l2_m2m_ctx* **m2m_ctx* m2m context assigned to the instance given by struct *v4l2_m2m_ctx*

Description

This last capture buffer should be tagged with `V4L2_BUF_FLAG_LAST` to notify the end of the capture session.

bool **v4l2_m2m_has_stopped**(struct *v4l2_m2m_ctx* **m2m_ctx*)
return the current encoding/decoding session stopped state

Parameters

struct *v4l2_m2m_ctx* **m2m_ctx* m2m context assigned to the instance given by struct *v4l2_m2m_ctx*

bool **v4l2_m2m_is_last_draining_src_buf**(struct *v4l2_m2m_ctx* **m2m_ctx*, struct *vb2_v4l2_buffer* **vbuf*)
return the output buffer draining state in the current encoding/decoding session

Parameters

struct *v4l2_m2m_ctx* **m2m_ctx* m2m context assigned to the instance given by struct *v4l2_m2m_ctx*

struct *vb2_v4l2_buffer* **vbuf* pointer to struct *v4l2_buffer*

Description

This will identify the last output buffer queued before a session stop was required, leading to an actual encoding/decoding session stop state in the encoding/decoding process after being processed.

```
void v4l2_m2m_last_buffer_done(struct v4l2_m2m_ctx *m2m_ctx, struct
                               vb2_v4l2_buffer *vbuf)
    marks the buffer with LAST flag and DONE
```

Parameters

struct v4l2_m2m_ctx *m2m_ctx m2m context assigned to the instance given by struct `v4l2_m2m_ctx`

struct vb2_v4l2_buffer *vbuf pointer to struct `v4l2_buffer`

void **v4l2_m2m_suspend**(struct v4l2_m2m_dev *m2m_dev)
stop new jobs from being run and wait for current job to finish

Parameters

struct v4l2_m2m_dev *m2m_dev opaque pointer to the internal data to handle M2M context

Description

Called by a driver in the suspend hook. Stop new jobs from being run, and wait for current running job to finish.

void **v4l2_m2m_resume**(struct v4l2_m2m_dev *m2m_dev)
resume job running and try to run a queued job

Parameters

struct v4l2_m2m_dev *m2m_dev opaque pointer to the internal data to handle M2M context

Description

Called by a driver in the resume hook. This reverts the operation of `v4l2_m2m_suspend()` and allows job to be run. Also try to run a queued job if there is any.

```
int v4l2_m2m_reqbufs(struct file *file, struct v4l2_m2m_ctx *m2m_ctx, struct
                    v4l2_requestbuffers *reqbufs)
    multi-queue-aware REQBUFS multiplexer
```

Parameters

struct file *file pointer to struct `file`

struct v4l2_m2m_ctx *m2m_ctx m2m context assigned to the instance given by struct `v4l2_m2m_ctx`

struct v4l2_requestbuffers *reqbufs pointer to struct `v4l2_requestbuffers`

```
int v4l2_m2m_querybuf(struct file *file, struct v4l2_m2m_ctx *m2m_ctx, struct
                     v4l2_buffer *buf)
    multi-queue-aware QUERYBUF multiplexer
```

Parameters

struct file *file pointer to struct `file`

struct v4l2_m2m_ctx *m2m_ctx m2m context assigned to the instance given by struct `v4l2_m2m_ctx`

struct v4l2_buffer *buf pointer to struct `v4l2_buffer`

Description

See `v4l2_m2m_mmap()` documentation for details.

```
int v4l2_m2m_qbuf(struct file *file, struct v4l2_m2m_ctx *m2m_ctx, struct
                v4l2_buffer *buf)
    enqueue a source or destination buffer, depending on the type
```

Parameters

struct file *file pointer to struct file

struct v4l2_m2m_ctx *m2m_ctx m2m context assigned to the instance given by struct `v4l2_m2m_ctx`

struct v4l2_buffer *buf pointer to struct v4l2_buffer

```
int v4l2_m2m_dqbuf(struct file *file, struct v4l2_m2m_ctx *m2m_ctx, struct
                 v4l2_buffer *buf)
    dequeue a source or destination buffer, depending on the type
```

Parameters

struct file *file pointer to struct file

struct v4l2_m2m_ctx *m2m_ctx m2m context assigned to the instance given by struct `v4l2_m2m_ctx`

struct v4l2_buffer *buf pointer to struct v4l2_buffer

```
int v4l2_m2m_prepare_buf(struct file *file, struct v4l2_m2m_ctx *m2m_ctx, struct
                       v4l2_buffer *buf)
    prepare a source or destination buffer, depending on the type
```

Parameters

struct file *file pointer to struct file

struct v4l2_m2m_ctx *m2m_ctx m2m context assigned to the instance given by struct `v4l2_m2m_ctx`

struct v4l2_buffer *buf pointer to struct v4l2_buffer

```
int v4l2_m2m_create_bufs(struct file *file, struct v4l2_m2m_ctx *m2m_ctx, struct
                       v4l2_create_buffers *create)
    create a source or destination buffer, depending on the type
```

Parameters

struct file *file pointer to struct file

struct v4l2_m2m_ctx *m2m_ctx m2m context assigned to the instance given by struct `v4l2_m2m_ctx`

struct v4l2_create_buffers *create pointer to struct v4l2_create_buffers

```
int v4l2_m2m_expbuf(struct file *file, struct v4l2_m2m_ctx *m2m_ctx, struct
                  v4l2_exportbuffer *eb)
    export a source or destination buffer, depending on the type
```

Parameters

struct file *file pointer to struct file

struct v4l2_m2m_ctx *m2m_ctx m2m context assigned to the instance given by struct [v4l2_m2m_ctx](#)

struct v4l2_exportbuffer *eb pointer to struct [v4l2_exportbuffer](#)

int **v4l2_m2m_streamon**(struct file *file, struct [v4l2_m2m_ctx](#) *m2m_ctx, enum v4l2_buf_type type)
turn on streaming for a video queue

Parameters

struct file *file pointer to struct file

struct v4l2_m2m_ctx *m2m_ctx m2m context assigned to the instance given by struct [v4l2_m2m_ctx](#)

enum v4l2_buf_type type type of the V4L2 buffer, as defined by enum [v4l2_buf_type](#)

int **v4l2_m2m_streamoff**(struct file *file, struct [v4l2_m2m_ctx](#) *m2m_ctx, enum v4l2_buf_type type)
turn off streaming for a video queue

Parameters

struct file *file pointer to struct file

struct v4l2_m2m_ctx *m2m_ctx m2m context assigned to the instance given by struct [v4l2_m2m_ctx](#)

enum v4l2_buf_type type type of the V4L2 buffer, as defined by enum [v4l2_buf_type](#)

void **v4l2_m2m_update_start_streaming_state**(struct [v4l2_m2m_ctx](#) *m2m_ctx, struct [vb2_queue](#) *q)
update the encoding/decoding session state when a start of streaming of a video queue is requested

Parameters

struct v4l2_m2m_ctx *m2m_ctx m2m context assigned to the instance given by struct [v4l2_m2m_ctx](#)

struct vb2_queue *q queue

void **v4l2_m2m_update_stop_streaming_state**(struct [v4l2_m2m_ctx](#) *m2m_ctx, struct [vb2_queue](#) *q)
update the encoding/decoding session state when a stop of streaming of a video queue is requested

Parameters

struct v4l2_m2m_ctx *m2m_ctx m2m context assigned to the instance given by struct [v4l2_m2m_ctx](#)

struct vb2_queue *q queue

int **v4l2_m2m_encoder_cmd**(struct file *file, struct [v4l2_m2m_ctx](#) *m2m_ctx, struct v4l2_encoder_cmd *ec)
execute an encoder command

Parameters

struct file *file pointer to struct file

struct v4l2_m2m_ctx *m2m_ctx m2m context assigned to the instance given by struct [v4l2_m2m_ctx](#)

struct v4l2_encoder_cmd *ec pointer to the encoder command

int **v4l2_m2m_decoder_cmd**(struct file *file, struct [v4l2_m2m_ctx](#) *m2m_ctx, struct v4l2_decoder_cmd *dc)
execute a decoder command

Parameters

struct file *file pointer to struct file

struct v4l2_m2m_ctx *m2m_ctx m2m context assigned to the instance given by struct [v4l2_m2m_ctx](#)

struct v4l2_decoder_cmd *dc pointer to the decoder command

__poll_t v4l2_m2m_poll(struct file *file, struct [v4l2_m2m_ctx](#) *m2m_ctx, struct poll_table_struct *wait)
poll replacement, for destination buffers only

Parameters

struct file *file pointer to struct file

struct v4l2_m2m_ctx *m2m_ctx m2m context assigned to the instance given by struct [v4l2_m2m_ctx](#)

struct poll_table_struct *wait pointer to struct poll_table_struct

Description

Call from the driver's poll() function. Will poll both queues. If a buffer is available to dequeue (with dqbuf) from the source queue, this will indicate that a non-blocking write can be performed, while read will be returned in case of the destination queue.

int **v4l2_m2m_mmap**(struct file *file, struct [v4l2_m2m_ctx](#) *m2m_ctx, struct vm_area_struct *vma)
source and destination queues-aware mmap multiplexer

Parameters

struct file *file pointer to struct file

struct v4l2_m2m_ctx *m2m_ctx m2m context assigned to the instance given by struct [v4l2_m2m_ctx](#)

struct vm_area_struct *vma pointer to struct vm_area_struct

Description

Call from driver's mmap() function. Will handle mmap() for both queues seamlessly for videobuffer, which will receive normal per-queue offsets and proper videobuf queue pointers. The differentiation is made outside videobuf by adding a predefined offset to buffers from one of the queues and subtracting it before passing it back to videobuf. Only drivers (and thus applications) receive modified offsets.

struct v4l2_m2m_dev * **v4l2_m2m_init**(const struct [v4l2_m2m_ops](#) *m2m_ops)
initialize per-driver m2m data

Parameters

const struct v4l2_m2m_ops *m2m_ops pointer to *struct v4l2_m2m_ops*

Description

Usually called from driver's `probe()` function.

Return

returns an opaque pointer to the internal data to handle M2M context

void v4l2_m2m_release(struct *v4l2_m2m_dev* **m2m_dev*)
cleans up and frees a *m2m_dev* structure

Parameters

struct v4l2_m2m_dev *m2m_dev opaque pointer to the internal data to handle M2M context

Description

Usually called from driver's `remove()` function.

struct v4l2_m2m_ctx *v4l2_m2m_ctx_init(struct *v4l2_m2m_dev* **m2m_dev*,
void **drv_priv*, int (**queue_init*)(void **priv*,
struct *vb2_queue* **src_vq*, struct *vb2_queue*
**dst_vq*)

allocate and initialize a m2m context

Parameters

struct v4l2_m2m_dev *m2m_dev opaque pointer to the internal data to handle M2M context

void *drv_priv driver's instance private data

int (*queue_init)(void *priv, struct vb2_queue *src_vq, struct vb2_queue *dst_vq)
a callback for queue type-specific initialization function to be used for initializing videobuf_queues

Description

Usually called from driver's `open()` function.

void v4l2_m2m_ctx_release(struct *v4l2_m2m_ctx* **m2m_ctx*)
release m2m context

Parameters

struct v4l2_m2m_ctx *m2m_ctx m2m context assigned to the instance given by struct *v4l2_m2m_ctx*

Description

Usually called from driver's `release()` function.

void v4l2_m2m_buf_queue(struct *v4l2_m2m_ctx* **m2m_ctx*, struct *vb2_v4l2_buffer* **vbuf*)
add a buffer to the proper ready buffers list.

Parameters

struct v4l2_m2m_ctx *m2m_ctx m2m context assigned to the instance given by struct *v4l2_m2m_ctx*

struct vb2_v4l2_buffer *vbuf pointer to struct *vb2_v4l2_buffer*

Description

Call from videobuf_queue_ops->ops->buf_queue, videobuf_queue_ops callback.

unsigned int **v4l2_m2m_num_src_bufs_ready**(struct *v4l2_m2m_ctx* **m2m_ctx*)
return the number of source buffers ready for use

Parameters

struct v4l2_m2m_ctx **m2m_ctx* m2m context assigned to the instance given by struct *v4l2_m2m_ctx*

unsigned int **v4l2_m2m_num_dst_bufs_ready**(struct *v4l2_m2m_ctx* **m2m_ctx*)
return the number of destination buffers ready for use

Parameters

struct v4l2_m2m_ctx **m2m_ctx* m2m context assigned to the instance given by struct *v4l2_m2m_ctx*

struct *vb2_v4l2_buffer* * **v4l2_m2m_next_buf**(struct *v4l2_m2m_queue_ctx* **q_ctx*)
return next buffer from the list of ready buffers

Parameters

struct v4l2_m2m_queue_ctx **q_ctx* pointer to struct **v4l2_m2m_queue_ctx**

struct *vb2_v4l2_buffer* * **v4l2_m2m_next_src_buf**(struct *v4l2_m2m_ctx* **m2m_ctx*)
return next source buffer from the list of ready buffers

Parameters

struct v4l2_m2m_ctx **m2m_ctx* m2m context assigned to the instance given by struct *v4l2_m2m_ctx*

struct *vb2_v4l2_buffer* * **v4l2_m2m_next_dst_buf**(struct *v4l2_m2m_ctx* **m2m_ctx*)
return next destination buffer from the list of ready buffers

Parameters

struct v4l2_m2m_ctx **m2m_ctx* m2m context assigned to the instance given by struct *v4l2_m2m_ctx*

struct *vb2_v4l2_buffer* * **v4l2_m2m_last_buf**(struct *v4l2_m2m_queue_ctx* **q_ctx*)
return last buffer from the list of ready buffers

Parameters

struct v4l2_m2m_queue_ctx **q_ctx* pointer to struct **v4l2_m2m_queue_ctx**

struct *vb2_v4l2_buffer* * **v4l2_m2m_last_src_buf**(struct *v4l2_m2m_ctx* **m2m_ctx*)
return last destination buffer from the list of ready buffers

Parameters

struct v4l2_m2m_ctx **m2m_ctx* m2m context assigned to the instance given by struct *v4l2_m2m_ctx*

struct *vb2_v4l2_buffer* * **v4l2_m2m_last_dst_buf**(struct *v4l2_m2m_ctx* **m2m_ctx*)
return last destination buffer from the list of ready buffers

Parameters

struct v4l2_m2m_ctx *m2m_ctx m2m context assigned to the instance given by struct *v4l2_m2m_ctx*

v4l2_m2m_for_each_dst_buf(m2m_ctx, b)
iterate over a list of destination ready buffers

Parameters

m2m_ctx m2m context assigned to the instance given by struct *v4l2_m2m_ctx*

b current buffer of type *struct v4l2_m2m_buffer*

v4l2_m2m_for_each_src_buf(m2m_ctx, b)
iterate over a list of source ready buffers

Parameters

m2m_ctx m2m context assigned to the instance given by struct *v4l2_m2m_ctx*

b current buffer of type *struct v4l2_m2m_buffer*

v4l2_m2m_for_each_dst_buf_safe(m2m_ctx, b, n)
iterate over a list of destination ready buffers safely

Parameters

m2m_ctx m2m context assigned to the instance given by struct *v4l2_m2m_ctx*

b current buffer of type *struct v4l2_m2m_buffer*

n used as temporary storage

v4l2_m2m_for_each_src_buf_safe(m2m_ctx, b, n)
iterate over a list of source ready buffers safely

Parameters

m2m_ctx m2m context assigned to the instance given by struct *v4l2_m2m_ctx*

b current buffer of type *struct v4l2_m2m_buffer*

n used as temporary storage

struct vb2_queue *v4l2_m2m_get_src_vq(struct v4l2_m2m_ctx *m2m_ctx)
return vb2_queue for source buffers

Parameters

struct v4l2_m2m_ctx *m2m_ctx m2m context assigned to the instance given by struct *v4l2_m2m_ctx*

struct vb2_queue *v4l2_m2m_get_dst_vq(struct v4l2_m2m_ctx *m2m_ctx)
return vb2_queue for destination buffers

Parameters

struct v4l2_m2m_ctx *m2m_ctx m2m context assigned to the instance given by struct *v4l2_m2m_ctx*

struct vb2_v4l2_buffer *v4l2_m2m_buf_remove(struct v4l2_m2m_queue_ctx *q_ctx)
take off a buffer from the list of ready buffers and return it

Parameters

struct v4l2_m2m_queue_ctx *q_ctx pointer to struct *v4l2_m2m_queue_ctx*

struct *vb2_v4l2_buffer* * **v4l2_m2m_src_buf_remove**(struct *v4l2_m2m_ctx* **m2m_ctx*)
take off a source buffer from the list of ready buffers and return it

Parameters

struct *v4l2_m2m_ctx* **m2m_ctx* m2m context assigned to the instance given by struct *v4l2_m2m_ctx*

struct *vb2_v4l2_buffer* * **v4l2_m2m_dst_buf_remove**(struct *v4l2_m2m_ctx* **m2m_ctx*)
take off a destination buffer from the list of ready buffers and return it

Parameters

struct *v4l2_m2m_ctx* **m2m_ctx* m2m context assigned to the instance given by struct *v4l2_m2m_ctx*

void **v4l2_m2m_buf_remove_by_buf**(struct *v4l2_m2m_queue_ctx* **q_ctx*, struct *vb2_v4l2_buffer* **vbuf*)
take off exact buffer from the list of ready buffers

Parameters

struct *v4l2_m2m_queue_ctx* **q_ctx* pointer to struct **v4l2_m2m_queue_ctx**

struct *vb2_v4l2_buffer* **vbuf* the buffer to be removed

void **v4l2_m2m_src_buf_remove_by_buf**(struct *v4l2_m2m_ctx* **m2m_ctx*, struct *vb2_v4l2_buffer* **vbuf*)
take off exact source buffer from the list of ready buffers

Parameters

struct *v4l2_m2m_ctx* **m2m_ctx* m2m context assigned to the instance given by struct *v4l2_m2m_ctx*

struct *vb2_v4l2_buffer* **vbuf* the buffer to be removed

void **v4l2_m2m_dst_buf_remove_by_buf**(struct *v4l2_m2m_ctx* **m2m_ctx*, struct *vb2_v4l2_buffer* **vbuf*)
take off exact destination buffer from the list of ready buffers

Parameters

struct *v4l2_m2m_ctx* **m2m_ctx* m2m context assigned to the instance given by struct *v4l2_m2m_ctx*

struct *vb2_v4l2_buffer* **vbuf* the buffer to be removed

void **v4l2_m2m_buf_copy_metadata**(const struct *vb2_v4l2_buffer* **out_vb*, struct *vb2_v4l2_buffer* **cap_vb*, bool *copy_frame_flags*)
copy buffer metadata from the output buffer to the capture buffer

Parameters

const struct *vb2_v4l2_buffer* **out_vb* the output buffer that is the source of the metadata.

struct *vb2_v4l2_buffer* **cap_vb* the capture buffer that will receive the metadata.

bool *copy_frame_flags* copy the KEY/B/PFRAME flags as well.

Description

This helper function copies the timestamp, timecode (if the TIMECODE buffer flag was set), field and the TIMECODE, KEYFRAME, BFRAME, PFRAME and TSTAMP_SRC_MASK flags from **out_vb** to **cap_vb**.

If **copy_frame_flags** is false, then the KEYFRAME, BFRAME and PFRAME flags are not copied. This is typically needed for encoders that set this bits explicitly.

2.2.21 V4L2 async kAPI

enum **v4l2_async_match_type**

type of asynchronous subdevice logic to be used in order to identify a match

Constants

V4L2_ASYNC_MATCH_I2C Match will check for I2C adapter ID and address

V4L2_ASYNC_MATCH_FWNODE Match will use firmware node

Description

This enum is used by the asynchronous sub-device logic to define the algorithm that will be used to match an asynchronous device.

struct **v4l2_async_subdev**

sub-device descriptor, as known to a bridge

Definition

```
struct v4l2_async_subdev {
    enum v4l2_async_match_type match_type;
    union {
        struct fwnode_handle *fwnode;
        struct {
            int adapter_id;
            unsigned short address;
        } i2c;
    } match;
    struct list_head list;
    struct list_head asd_list;
};
```

Members

match_type type of match that will be used

match union of per-bus type matching data sets

match.fwnode

pointer to struct **fwnode_handle** to be matched. Used if **match_type** is **V4L2_ASYNC_MATCH_FWNODE**.

match.i2c embedded struct with I2C parameters to be matched. Both **match.i2c.adapter_id** and **match.i2c.address** should be matched. Used if **match_type** is **V4L2_ASYNC_MATCH_I2C**.

match.i2c.adapter_id

I2C adapter ID to be matched. Used if **match_type** is **V4L2_ASYNC_MATCH_I2C**.

`match.i2c.address`

I2C address to be matched. Used if **match_type** is `V4L2_ASYNC_MATCH_I2C`.

list used to link `struct v4l2_async_subdev` objects, waiting to be probed, to a notifier->waiting list

asd_list used to add `struct v4l2_async_subdev` objects to the master notifier **asd_list**

Description

When this struct is used as a member in a driver specific struct, the driver specific struct shall contain the `struct v4l2_async_subdev` as its first member.

struct **v4l2_async_notifier_operations**
Asynchronous V4L2 notifier operations

Definition

```
struct v4l2_async_notifier_operations {
    int (*bound)(struct v4l2_async_notifier *notifier, struct v4l2_subdev *subdev, struct
    ↪v4l2_async_subdev *asd);
    int (*complete)(struct v4l2_async_notifier *notifier);
    void (*unbind)(struct v4l2_async_notifier *notifier, struct v4l2_subdev *subdev,
    ↪struct v4l2_async_subdev *asd);
};
```

Members

bound a subdevice driver has successfully probed one of the subdevices

complete All subdevices have been probed successfully. The complete callback is only executed for the root notifier.

unbind a subdevice is leaving

struct **v4l2_async_notifier**
v4l2_device notifier data

Definition

```
struct v4l2_async_notifier {
    const struct v4l2_async_notifier_operations *ops;
    struct v4l2_device *v4l2_dev;
    struct v4l2_subdev *sd;
    struct v4l2_async_notifier *parent;
    struct list_head asd_list;
    struct list_head waiting;
    struct list_head done;
    struct list_head list;
};
```

Members

ops notifier operations

v4l2_dev v4l2_device of the root notifier, NULL otherwise

sd sub-device that registered the notifier, NULL otherwise

parent parent notifier

asd_list master list of *struct v4l2_async_subdev*

waiting list of *struct v4l2_async_subdev*, waiting for their drivers

done list of *struct v4l2_subdev*, already probed

list member in a global list of notifiers

void **v4l2_async_debug_init**(struct dentry *debugfs_dir)
Initialize debugging tools.

Parameters

struct dentry *debugfs_dir pointer to the parent debugfs struct dentry

void **v4l2_async_nf_init**(struct *v4l2_async_notifier* *notifier)
Initialize a notifier.

Parameters

struct v4l2_async_notifier *notifier pointer to *struct v4l2_async_notifier*

Description

This function initializes the notifier **asd_list**. It must be called before adding a subdevice to a notifier, using one of: *v4l2_async_nf_add_fwnode_remote()*, *v4l2_async_nf_add_fwnode()*, *v4l2_async_nf_add_i2c()*, *__v4l2_async_nf_add_subdev()* or *v4l2_async_nf_parse_fwnode_endpoints()*.

int **__v4l2_async_nf_add_subdev**(struct *v4l2_async_notifier* *notifier, struct *v4l2_async_subdev* *asd)
Add an async subdev to the notifier's master asd list.

Parameters

struct v4l2_async_notifier *notifier pointer to *struct v4l2_async_notifier*

struct v4l2_async_subdev *asd pointer to *struct v4l2_async_subdev*

Description

warning: Drivers should avoid using this function and instead use one of: *v4l2_async_nf_add_fwnode()*, *v4l2_async_nf_add_fwnode_remote()* or *v4l2_async_nf_add_i2c()*.

Call this function before registering a notifier to link the provided **asd** to the notifiers master **asd_list**. The **asd** must be allocated with *k*alloc()* as it will be freed by the framework when the notifier is destroyed.

v4l2_async_nf_add_fwnode(*notifier, fwnode, type*)
Allocate and add a fwnode async subdev to the notifier's master asd_list.

Parameters

notifier pointer to *struct v4l2_async_notifier*

fwnode fwnode handle of the sub-device to be matched, pointer to struct *fwnode_handle*

type Type of the driver's async sub-device struct. The *struct v4l2_async_subdev* shall be the first member of the driver's async sub-device struct, i.e. both begin at the same memory address.

Description

Allocate a fwnode-matched asd of size `asd_struct_size`, and add it to the notifiers **asd_list**. The function also gets a reference of the fwnode which is released later at notifier cleanup time.

v4l2_async_nf_add_fwnode_remote(*notifier, ep, type*)

Allocate and add a fwnode remote async subdev to the notifier's master `asd_list`.

Parameters

notifier pointer to *struct v4l2_async_notifier*

ep local endpoint pointing to the remote sub-device to be matched, pointer to *struct fwnode_handle*

type Type of the driver's async sub-device struct. The *struct v4l2_async_subdev* shall be the first member of the driver's async sub-device struct, i.e. both begin at the same memory address.

Description

Gets the remote endpoint of a given local endpoint, set it up for fwnode matching and adds the async sub-device to the notifier's **asd_list**. The function also gets a reference of the fwnode which is released later at notifier cleanup time.

This is just like *v4l2_async_nf_add_fwnode()*, but with the exception that the fwnode refers to a local endpoint, not the remote one.

v4l2_async_nf_add_i2c(*notifier, adapter, address, type*)

Allocate and add an i2c async subdev to the notifier's master `asd_list`.

Parameters

notifier pointer to *struct v4l2_async_notifier*

adapter I2C adapter ID to be matched

address I2C address of sub-device to be matched

type Type of the driver's async sub-device struct. The *struct v4l2_async_subdev* shall be the first member of the driver's async sub-device struct, i.e. both begin at the same memory address.

Description

Same as *v4l2_async_nf_add_fwnode()* but for I2C matched sub-devices.

```
int v4l2_async_nf_register(struct v4l2_device *v4l2_dev, struct
                          v4l2_async_notifier *notifier)
registers a subdevice asynchronous notifier
```

Parameters

struct v4l2_device *v4l2_dev pointer to *struct v4l2_device*

struct v4l2_async_notifier *notifier pointer to *struct v4l2_async_notifier*

```
int v4l2_async_subdev_nf_register(struct v4l2_subdev *sd, struct
                                 v4l2_async_notifier *notifier)
registers a subdevice asynchronous notifier for a sub-device
```

Parameters

struct v4l2_subdev *sd pointer to *struct v4l2_subdev*

struct v4l2_async_notifier *notifier pointer to *struct v4l2_async_notifier*

void **v4l2_async_nf_unregister**(*struct v4l2_async_notifier *notifier*)
unregisters a subdevice asynchronous notifier

Parameters

struct v4l2_async_notifier *notifier pointer to *struct v4l2_async_notifier*

void **v4l2_async_nf_cleanup**(*struct v4l2_async_notifier *notifier*)
clean up notifier resources

Parameters

struct v4l2_async_notifier *notifier the notifier the resources of which are to be cleaned up

Description

Release memory resources related to a notifier, including the async sub-devices allocated for the purposes of the notifier but not the notifier itself. The user is responsible for calling this function to clean up the notifier after calling *v4l2_async_nf_add_fwnode_remote()*, *v4l2_async_nf_add_fwnode()*, *v4l2_async_nf_add_i2c()*, *__v4l2_async_nf_add_subdev()* or *v4l2_async_nf_parse_fwnode_endpoints()*.

There is no harm from calling *v4l2_async_nf_cleanup()* in other cases as long as its memory has been zeroed after it has been allocated.

int **v4l2_async_register_subdev**(*struct v4l2_subdev *sd*)
registers a sub-device to the asynchronous subdevice framework

Parameters

struct v4l2_subdev *sd pointer to *struct v4l2_subdev*

int **v4l2_async_register_subdev_sensor**(*struct v4l2_subdev *sd*)
registers a sensor sub-device to the asynchronous sub-device framework and parse set up common sensor related devices

Parameters

struct v4l2_subdev *sd pointer to *struct v4l2_subdev*

Description

This function is just like *v4l2_async_register_subdev()* with the exception that calling it will also parse firmware interfaces for remote references using *v4l2_async_nf_parse_fwnode_sensor()* and registers the async sub-devices. The sub-device is similarly unregistered by calling *v4l2_async_unregister_subdev()*.

While registered, the subdev module is marked as in-use.

An error is returned if the module is no longer loaded on any attempts to register it.

void **v4l2_async_unregister_subdev**(*struct v4l2_subdev *sd*)
unregisters a sub-device to the asynchronous subdevice framework

Parameters

struct v4l2_subdev *sd pointer to *struct v4l2_subdev*

2.2.22 V4L2 fwnode kAPI

struct **v4l2_fwnode_bus_mipi_csi2**
MIPI CSI-2 bus data structure

Definition

```
struct v4l2_fwnode_bus_mipi_csi2 {
    unsigned int flags;
    unsigned char data_lanes[V4L2_FWNODE_CSI2_MAX_DATA_LANES];
    unsigned char clock_lane;
    unsigned char num_data_lanes;
    bool lane_polarities[1 + V4L2_FWNODE_CSI2_MAX_DATA_LANES];
};
```

Members

flags media bus (V4L2_MBUS_*) flags

data_lanes an array of physical data lane indexes

clock_lane physical lane index of the clock lane

num_data_lanes number of data lanes

lane_polarities polarity of the lanes. The order is the same of the physical lanes.

struct **v4l2_fwnode_bus_parallel**
parallel data bus data structure

Definition

```
struct v4l2_fwnode_bus_parallel {
    unsigned int flags;
    unsigned char bus_width;
    unsigned char data_shift;
};
```

Members

flags media bus (V4L2_MBUS_*) flags

bus_width bus width in bits

data_shift data shift in bits

struct **v4l2_fwnode_bus_mipi_csi1**
CSI-1/CCP2 data bus structure

Definition

```
struct v4l2_fwnode_bus_mipi_csi1 {
    unsigned char clock_inv:1;
    unsigned char strobe:1;
    bool lane_polarity[2];
    unsigned char data_lane;
    unsigned char clock_lane;
};
```

Members

clock_inv polarity of clock/strobe signal false - not inverted, true - inverted

strobe false - data/clock, true - data/strobe

lane_polarity the polarities of the clock (index 0) and data lanes index (1)

data_lane the number of the data lane

clock_lane the number of the clock lane

struct **v4l2_fwnode_endpoint**
the endpoint data structure

Definition

```
struct v4l2_fwnode_endpoint {
    struct fwnode_endpoint base;
    enum v4l2_mbus_type bus_type;
    struct {
        struct v4l2_fwnode_bus_parallel parallel;
        struct v4l2_fwnode_bus_mipi_csi1 mipi_csi1;
        struct v4l2_fwnode_bus_mipi_csi2 mipi_csi2;
    } bus;
    u64 *link_frequencies;
    unsigned int nr_of_link_frequencies;
};
```

Members

base fwnode endpoint of the v4l2_fwnode

bus_type bus type

bus bus configuration data structure

bus.parallel embedded *struct v4l2_fwnode_bus_parallel*. Used if the bus is parallel.

bus.mipi_csi1 embedded *struct v4l2_fwnode_bus_mipi_csi1*. Used if the bus is MIPI Alliance's Camera Serial Interface version 1 (MIPI CSI1) or Standard Mobile Imaging Architecture's Compact Camera Port 2 (SMIA CCP2).

bus.mipi_csi2 embedded *struct v4l2_fwnode_bus_mipi_csi2*. Used if the bus is MIPI Alliance's Camera Serial Interface version 2 (MIPI CSI2).

link_frequencies array of supported link frequencies

nr_of_link_frequencies number of elements in link_frequencies array

V4L2_FWNODE_PROPERTY_UNSET()
identify a non initialized property

Parameters

Description

All properties in *struct v4l2_fwnode_device_properties* are initialized to this value.

enum **v4l2_fwnode_orientation**
possible device orientation

Constants

V4L2_FWNODE_ORIENTATION_FRONT device installed on the front side

V4L2_FWNODE_ORIENTATION_BACK device installed on the back side

V4L2_FWNODE_ORIENTATION_EXTERNAL device externally located

struct **v4l2_fwnode_device_properties**
fwnode device properties

Definition

```
struct v4l2_fwnode_device_properties {
    enum v4l2_fwnode_orientation orientation;
    unsigned int rotation;
};
```

Members

orientation device orientation. See *enum v4l2_fwnode_orientation*

rotation device rotation

struct **v4l2_fwnode_link**
a link between two endpoints

Definition

```
struct v4l2_fwnode_link {
    struct fwnode_handle *local_node;
    unsigned int local_port;
    unsigned int local_id;
    struct fwnode_handle *remote_node;
    unsigned int remote_port;
    unsigned int remote_id;
};
```

Members

local_node pointer to device_node of this endpoint

local_port identifier of the port this endpoint belongs to

local_id identifier of the id this endpoint belongs to

remote_node pointer to device_node of the remote endpoint

remote_port identifier of the port the remote endpoint belongs to

remote_id identifier of the id the remote endpoint belongs to

enum **v4l2_connector_type**
connector type

Constants

V4L2_CONN_UNKNOWN unknown connector type, no V4L2 connector configuration

V4L2_CONN_COMPOSITE analog composite connector

V4L2_CONN_SVIDEO analog svideo connector

struct **v4l2_connector_link**
connector link data structure

Definition

```
struct v4l2_connector_link {
    struct list_head head;
    struct v4l2_fwnode_link fwnode_link;
};
```

Members

head structure to be used to add the link to the *struct v4l2_fwnode_connector*

fwnode_link *struct v4l2_fwnode_link* link between the connector and the device the connector belongs to.

struct **v4l2_fwnode_connector_analog**
analog connector data structure

Definition

```
struct v4l2_fwnode_connector_analog {
    v4l2_std_id sdtv_stds;
};
```

Members

sdtv_stds sdtv standards this connector supports, set to V4L2_STD_ALL if no restrictions are specified.

struct **v4l2_fwnode_connector**
the connector data structure

Definition

```
struct v4l2_fwnode_connector {
    const char *name;
    const char *label;
    enum v4l2_connector_type type;
    struct list_head links;
    unsigned int nr_of_links;
    union {
        struct v4l2_fwnode_connector_analog analog;
    } connector;
};
```

Members

name the connector device name

label optional connector label

type connector type

links list of all connector *struct v4l2_connector_link* links

nr_of_links total number of links

connector connector configuration

connector.analog analog connector configuration *struct v4l2_fwnode_connector_analog*

enum **v4l2_fwnode_bus_type**

Video bus types defined by firmware properties

Constants

V4L2_FWNODE_BUS_TYPE_GUESS Default value if no bus-type fwnode property

V4L2_FWNODE_BUS_TYPE_CSI2_CPHY MIPI CSI-2 bus, C-PHY physical layer

V4L2_FWNODE_BUS_TYPE_CSI1 MIPI CSI-1 bus

V4L2_FWNODE_BUS_TYPE_CCP2 SMIA Compact Camera Port 2 bus

V4L2_FWNODE_BUS_TYPE_CSI2_DPHY MIPI CSI-2 bus, D-PHY physical layer

V4L2_FWNODE_BUS_TYPE_PARALLEL Camera Parallel Interface bus

V4L2_FWNODE_BUS_TYPE_BT656 BT.656 video format bus-type

NR_OF_V4L2_FWNODE_BUS_TYPE Number of bus-types

int **v4l2_fwnode_endpoint_parse**(struct fwnode_handle *fwnode, struct v4l2_fwnode_endpoint *vep)
parse all fwnode node properties

Parameters

struct fwnode_handle *fwnode pointer to the endpoint's fwnode handle

struct v4l2_fwnode_endpoint *vep pointer to the V4L2 fwnode data structure

Description

This function parses the V4L2 fwnode endpoint specific parameters from the firmware. There are two ways to use this function, either by letting it obtain the type of the bus (by setting the **vep.bus_type** field to **V4L2_MBUS_UNKNOWN**) or specifying the bus type explicitly to one of the *enum v4l2_mbus_type* types.

When **vep.bus_type** is **V4L2_MBUS_UNKNOWN**, the function will use the “bus-type” property to determine the type when it is available. The caller is responsible for validating the contents of **vep.bus_type** field after the call returns.

As a deprecated functionality to support older DT bindings without “bus-type” property for devices that support multiple types, if the “bus-type” property does not exist, the function will attempt to guess the type based on the endpoint properties available. **NEVER RELY ON GUESSING THE BUS TYPE IN NEW DRIVERS OR BINDINGS.**

It is also possible to set **vep.bus_type** corresponding to an actual bus. In this case the function will only attempt to parse properties related to this bus, and it will return an error if the value of the “bus-type” property corresponds to a different bus.

The caller is required to initialise all fields of **vep**, either with explicitly values, or by zeroing them.

The function does not change the V4L2 fwnode endpoint state if it fails.

NOTE

This function does not parse “link-frequencies” property as its size is not known in advance. Please use *v4l2_fwnode_endpoint_alloc_parse()* if you need properties of variable size.

Return

0 on success or a negative error code on failure: -ENOMEM on memory allocation failure
-EINVAL on parsing failure -ENXIO on mismatching bus types

void **v4l2_fwnode_endpoint_free**(struct *v4l2_fwnode_endpoint* *vep)
free the V4L2 fwnode acquired by *v4l2_fwnode_endpoint_alloc_parse()*

Parameters

struct v4l2_fwnode_endpoint *vep the V4L2 fwnode the resources of which are to be released

Description

It is safe to call this function with NULL argument or on a V4L2 fwnode the parsing of which failed.

int **v4l2_fwnode_endpoint_alloc_parse**(struct *fwnode_handle* *fwnode, struct *v4l2_fwnode_endpoint* *vep)
parse all fwnode node properties

Parameters

struct fwnode_handle *fwnode pointer to the endpoint's fwnode handle

struct v4l2_fwnode_endpoint *vep pointer to the V4L2 fwnode data structure

Description

This function parses the V4L2 fwnode endpoint specific parameters from the firmware. There are two ways to use this function, either by letting it obtain the type of the bus (by setting the **vep.bus_type** field to V4L2_MBUS_UNKNOWN) or specifying the bus type explicitly to one of the *enum v4l2_mbus_type* types.

When **vep.bus_type** is V4L2_MBUS_UNKNOWN, the function will use the “bus-type” property to determine the type when it is available. The caller is responsible for validating the contents of **vep.bus_type** field after the call returns.

As a deprecated functionality to support older DT bindings without “bus-type” property for devices that support multiple types, if the “bus-type” property does not exist, the function will attempt to guess the type based on the endpoint properties available. NEVER RELY ON GUESSING THE BUS TYPE IN NEW DRIVERS OR BINDINGS.

It is also possible to set **vep.bus_type** corresponding to an actual bus. In this case the function will only attempt to parse properties related to this bus, and it will return an error if the value of the “bus-type” property corresponds to a different bus.

The caller is required to initialise all fields of **vep**, either with explicitly values, or by zeroing them.

The function does not change the V4L2 fwnode endpoint state if it fails.

v4l2_fwnode_endpoint_alloc_parse() has two important differences to *v4l2_fwnode_endpoint_parse()*:

1. It also parses variable size data.
2. The memory it has allocated to store the variable size data must be freed using *v4l2_fwnode_endpoint_free()* when no longer needed.

Return

0 on success or a negative error code on failure: -ENOMEM on memory allocation failure
-EINVAL on parsing failure -ENXIO on mismatching bus types

```
int v4l2_fwnode_parse_link(struct fwnode_handle *fwnode, struct
                          v4l2_fwnode_link *link)
    parse a link between two endpoints
```

Parameters

struct fwnode_handle *fwnode pointer to the endpoint's fwnode at the local end of the link

struct v4l2_fwnode_link *link pointer to the V4L2 fwnode link data structure

Description

Fill the link structure with the local and remote nodes and port numbers. The `local_node` and `remote_node` fields are set to point to the local and remote port's parent nodes respectively (the port parent node being the parent node of the port node if that node isn't a 'ports' node, or the grand-parent node of the port node otherwise).

A reference is taken to both the local and remote nodes, the caller must use `v4l2_fwnode_put_link()` to drop the references when done with the link.

Return

0 on success, or `-ENOLINK` if the remote endpoint fwnode can't be found.

```
void v4l2_fwnode_put_link(struct v4l2_fwnode_link *link)
    drop references to nodes in a link
```

Parameters

struct v4l2_fwnode_link *link pointer to the V4L2 fwnode link data structure

Description

Drop references to the local and remote nodes in the link. This function must be called on every link parsed with `v4l2_fwnode_parse_link()`.

```
void v4l2_fwnode_connector_free(struct v4l2_fwnode_connector *connector)
    free the V4L2 connector acquired memory
```

Parameters

struct v4l2_fwnode_connector *connector the V4L2 connector resources of which are to be released

Description

Free all allocated memory and put all links acquired by `v4l2_fwnode_connector_parse()` and `v4l2_fwnode_connector_add_link()`.

It is safe to call this function with `NULL` argument or on a V4L2 connector the parsing of which failed.

```
int v4l2_fwnode_connector_parse(struct fwnode_handle *fwnode, struct
                               v4l2_fwnode_connector *connector)
    initialize the 'struct v4l2_fwnode_connector'
```

Parameters

struct fwnode_handle *fwnode pointer to the subdev endpoint's fwnode handle where the connector is connected to or to the connector endpoint fwnode handle.

struct v4l2_fwnode_connector *connector pointer to the V4L2 fwnode connector data structure

Description

Fill the *struct v4l2_fwnode_connector* with the connector type, label and all *enum v4l2_connector_type* specific connector data. The label is optional so it is set to NULL if no one was found. The function initialize the links to zero. Adding links to the connector is done by calling *v4l2_fwnode_connector_add_link()*.

The memory allocated for the label must be freed when no longer needed. Freeing the memory is done by *v4l2_fwnode_connector_free()*.

Return

- 0 on success or a negative error code on failure:
- -EINVAL if **fwnode** is invalid
- -ENOTCONN if connector type is unknown or connector device can't be found

int **v4l2_fwnode_connector_add_link**(struct fwnode_handle *fwnode, struct *v4l2_fwnode_connector* *connector)
add a link between a connector node and a v4l2-subdev node.

Parameters

struct fwnode_handle *fwnode pointer to the subdev endpoint's fwnode handle where the connector is connected to

struct v4l2_fwnode_connector *connector pointer to the V4L2 fwnode connector data structure

Description

Add a new *struct v4l2_connector_link* link to the *struct v4l2_fwnode_connector* connector links list. The link *local_node* points to the connector node, the *remote_node* to the host v4l2 (sub)dev.

The taken references to *remote_node* and *local_node* must be dropped and the allocated memory must be freed when no longer needed. Both is done by calling *v4l2_fwnode_connector_free()*.

Return

- 0 on success or a negative error code on failure:
- -EINVAL if **fwnode** or **connector** is invalid or **connector** type is unknown
- -ENOMEM on link memory allocation failure
- -ENOTCONN if remote connector device can't be found
- -ENOLINK if link parsing between v4l2 (sub)dev and connector fails

int **v4l2_fwnode_device_parse**(struct device *dev, struct *v4l2_fwnode_device_properties* *props)
parse fwnode device properties

Parameters

struct device *dev pointer to struct device

struct v4l2_fwnode_device_properties *props pointer to *struct v4l2_fwnode_device_properties* where to store the parsed properties values

Description

This function parses and validates the V4L2 fwnode device properties from the firmware interface, and fills the **struct** `v4l2_fwnode_device_properties` provided by the caller.

Return

% 0 on success -EINVAL if a parsed property value is not valid

parse_endpoint_func

Typedef: Driver's callback function to be called on each V4L2 fwnode endpoint.

Syntax

```
int parse_endpoint_func (struct device *dev, struct
v4l2_fwnode_endpoint *vep, struct v4l2_async_subdev *asd)
```

Parameters

struct device *dev pointer to `struct device`

struct v4l2_fwnode_endpoint *vep pointer to `struct v4l2_fwnode_endpoint`

struct v4l2_async_subdev *asd pointer to `struct v4l2_async_subdev`

Return

- 0 on success
- -ENOTCONN if the endpoint is to be skipped but this should not be considered as an error
- -EINVAL if the endpoint configuration is invalid

```
int v4l2_async_nf_parse_fwnode_endpoints(struct device *dev, struct
v4l2_async_notifier *notifier,
size_t asd_struct_size,
parse_endpoint_func parse_endpoint)
```

Parse V4L2 fwnode endpoints in a device node

Parameters

struct device *dev the device the endpoints of which are to be parsed

struct v4l2_async_notifier *notifier notifier for **dev**

size_t asd_struct_size size of the driver's async sub-device struct, including `sizeof(struct v4l2_async_subdev)`. The `struct v4l2_async_subdev` shall be the first member of the driver's async sub-device struct, i.e. both begin at the same memory address.

parse_endpoint_func parse_endpoint Driver's callback function called on each V4L2 fwnode endpoint. Optional.

Description

DEPRECATED! This function is deprecated. Don't use it in new drivers. Instead see an example in `cio2_parse_firmware()` function in `drivers/media/pci/intel/ipu3/ipu3-cio2.c`.

Parse the fwnode endpoints of the **dev** device and populate the async sub-devices list in the notifier. The **parse_endpoint** callback function is called for each endpoint with the corresponding async sub-device pointer to let the caller initialize the driver-specific part of the async sub-device structure.

The notifier memory shall be zeroed before this function is called on the notifier.

This function may not be called on a registered notifier and may be called on a notifier only once.

The `struct v4l2_fwnode_endpoint` passed to the callback function **parse_endpoint** is released once the function is finished. If there is a need to retain that configuration, the user needs to allocate memory for it.

Any notifier populated using this function must be released with a call to `v4l2_async_nf_cleanup()` after it has been unregistered and the async sub-devices are no longer in use, even if the function returned an error.

Return

0 on success, including when no async sub-devices are found -ENOMEM if memory allocation failed -EINVAL if graph or endpoint parsing failed Other error codes as returned by **parse_endpoint**

2.2.23 V4L2 rect helper functions

void **v4l2_rect_set_size_to**(struct v4l2_rect *r, const struct v4l2_rect *size)
copy the width/height values.

Parameters

struct v4l2_rect *r rect whose width and height fields will be set

const struct v4l2_rect *size rect containing the width and height fields you need.

void **v4l2_rect_set_min_size**(struct v4l2_rect *r, const struct v4l2_rect *min_size)
width and height of r should be >= min_size.

Parameters

struct v4l2_rect *r rect whose width and height will be modified

const struct v4l2_rect *min_size rect containing the minimal width and height

void **v4l2_rect_set_max_size**(struct v4l2_rect *r, const struct v4l2_rect *max_size)
width and height of r should be <= max_size

Parameters

struct v4l2_rect *r rect whose width and height will be modified

const struct v4l2_rect *max_size rect containing the maximum width and height

void **v4l2_rect_map_inside**(struct v4l2_rect *r, const struct v4l2_rect *boundary)
r should be inside boundary.

Parameters

struct v4l2_rect *r rect that will be modified

const struct v4l2_rect *boundary rect containing the boundary for r

bool **v4l2_rect_same_size**(const struct v4l2_rect *r1, const struct v4l2_rect *r2)
return true if r1 has the same size as r2

Parameters

const struct v4l2_rect *r1 rectangle.

const struct v4l2_rect *r2 rectangle.

Description

Return true if both rectangles have the same size.

bool **v4l2_rect_same_position**(const struct v4l2_rect *r1, const struct v4l2_rect *r2)
return true if r1 has the same position as r2

Parameters

const struct v4l2_rect *r1 rectangle.

const struct v4l2_rect *r2 rectangle.

Description

Return true if both rectangles have the same position

bool **v4l2_rect_equal**(const struct v4l2_rect *r1, const struct v4l2_rect *r2)
return true if r1 equals r2

Parameters

const struct v4l2_rect *r1 rectangle.

const struct v4l2_rect *r2 rectangle.

Description

Return true if both rectangles have the same size and position.

void **v4l2_rect_intersect**(struct v4l2_rect *r, const struct v4l2_rect *r1, const struct v4l2_rect *r2)
calculate the intersection of two rects.

Parameters

struct v4l2_rect *r intersection of **r1** and **r2**.

const struct v4l2_rect *r1 rectangle.

const struct v4l2_rect *r2 rectangle.

void **v4l2_rect_scale**(struct v4l2_rect *r, const struct v4l2_rect *from, const struct v4l2_rect *to)
scale rect r by to/from

Parameters

struct v4l2_rect *r rect to be scaled.

const struct v4l2_rect *from from rectangle.

const struct v4l2_rect *to to rectangle.

Description

This scales rectangle **r** horizontally by **to->width** / **from->width** and vertically by **to->height** / **from->height**.

Typically **r** is a rectangle inside **from** and you want the rectangle as it would appear after scaling **from** to **to**. So the resulting **r** will be the scaled rectangle inside **to**.

bool **v4l2_rect_overlap**(const struct v4l2_rect *r1, const struct v4l2_rect *r2)
do r1 and r2 overlap?

Parameters

const struct v4l2_rect *r1 rectangle.

const struct v4l2_rect *r2 rectangle.

Description

Returns true if **r1** and **r2** overlap.

bool **v4l2_rect_enclosed**(struct v4l2_rect *r1, struct v4l2_rect *r2)
is r1 enclosed in r2?

Parameters

struct v4l2_rect *r1 rectangle.

struct v4l2_rect *r2 rectangle.

Description

Returns true if **r1** is enclosed in **r2**.

2.2.24 Tuner functions and data structures

enum **tuner_mode**
Mode of the tuner

Constants

T_RADIO Tuner core will work in radio mode

T_ANALOG_TV Tuner core will work in analog TV mode

Description

Older boards only had a single tuner device, but some devices have a separate tuner for radio. In any case, the tuner-core needs to know if the tuner chip(s) will be used in radio mode or analog TV mode, as, on radio mode, frequencies are specified on a different range than on TV mode. This enum is used by the tuner core in order to work with the proper tuner range and eventually use a different tuner chip while in radio mode.

struct **tuner_setup**
setup the tuner chipsets

Definition

```
struct tuner_setup {
    unsigned short  addr;
    unsigned int    type;
    unsigned int    mode_mask;
    void *config;
    int (*tuner_callback)(void *dev, int component, int cmd, int arg);
};
```

Members

addr I2C address used to control the tuner device/chipset

type Type of the tuner, as defined at the TUNER_* macros. Each different tuner model should have a unique identifier.

mode_mask Mask with the allowed tuner modes: V4L2_TUNER_RADIO, V4L2_TUNER_ANALOG_TV and/or V4L2_TUNER_DIGITAL_TV, describing if the tuner should be used to support Radio, analog TV and/or digital TV.

config Used to send tuner-specific configuration for complex tuners that require extra parameters to be set. Only a very few tuners require it and its usage on newer tuners should be avoided.

tuner_callback Some tuners require to call back the bridge driver, in order to do some tasks like rising a GPIO at the bridge chipset, in order to do things like resetting the device.

Description

Older boards only had a single tuner device. Nowadays multiple tuner devices may be present on a single board. Using TUNER_SET_TYPE_ADDR to pass the tuner_setup structure it is possible to setup each tuner device in turn.

Since multiple devices may be present it is no longer sufficient to send a command to a single i2c device. Instead you should broadcast the command to all i2c devices.

By setting the mode_mask correctly you can select which commands are accepted by a specific tuner device. For example, set mode_mask to T_RADIO if the device is a radio-only tuner. That specific tuner will only accept commands when the tuner is in radio mode and ignore them when the tuner is set to TV mode.

enum **param_type**
type of the tuner parameters

Constants

TUNER_PARAM_TYPE_RADIO Tuner params are for FM and/or AM radio

TUNER_PARAM_TYPE_PAL Tuner params are for PAL color TV standard

TUNER_PARAM_TYPE_SECAM Tuner params are for SECAM color TV standard

TUNER_PARAM_TYPE_NTSC Tuner params are for NTSC color TV standard

TUNER_PARAM_TYPE_DIGITAL Tuner params are for digital TV

struct **tuner_range**
define the frequencies supported by the tuner

Definition

```
struct tuner_range {
    unsigned short limit;
    unsigned char config;
    unsigned char cb;
};
```

Members

limit Max frequency supported by that range, in 62.5 kHz (TV) or 62.5 Hz (Radio), as defined by V4L2_TUNER_CAP_LOW.

config Value of the band switch byte (BB) to setup this mode.

cb Value of the CB byte to setup this mode.

Description

Please notice that digital tuners like xc3028/xc4000/xc5000 don't use those ranges, as they're defined inside the driver. This is used by analog tuners that are compatible with the "Philips way" to setup the tuners. On those devices, the tuner set is done via 4 bytes:

- 1) divider byte1 (DB1)
- 2) divider byte 2 (DB2)
- 3) Control byte (CB)
- 4) band switch byte (BB)

Some tuners also have an additional optional Auxiliary byte (AB).

struct `tuner_params`

Parameters to be used to setup the tuner. Those are used by drivers/media/tuners/tuner-types.c in order to specify the tuner properties. Most of the parameters are for tuners based on tda9887 IF-PLL multi-standard analog TV/Radio demodulator, with is very common on legacy analog tuners.

Definition

```
struct tuner_params {
    enum param_type type;
    unsigned int cb_first_if_lower_freq:1;
    unsigned int has_tda9887:1;
    unsigned int port1_fm_high_sensitivity:1;
    unsigned int port2_fm_high_sensitivity:1;
    unsigned int fm_gain_normal:1;
    unsigned int intercarrier_mode:1;
    unsigned int port1_active:1;
    unsigned int port2_active:1;
    unsigned int port1_invert_for_secam_lc:1;
    unsigned int port2_invert_for_secam_lc:1;
    unsigned int port1_set_for_fm_mono:1;
    unsigned int default_pll_gating_l8:1;
    unsigned int radio_if:2;
    signed int default_top_low:5;
    signed int default_top_mid:5;
    signed int default_top_high:5;
    signed int default_top_secam_low:5;
    signed int default_top_secam_mid:5;
    signed int default_top_secam_high:5;
    ul6 iffreq;
    unsigned int count;
    struct tuner_range *ranges;
};
```

Members

type Type of the tuner parameters, as defined at [enum param_type](#). If the tuner supports multiple standards, an array should be used, with one row per different standard.

cb_first_if_lower_freq Many Philips-based tuners have a comment in their datasheet like "For channel selection involving band switching, and to ensure smooth tuning to the desired channel without causing unnecessary charge pump action, it is recommended to consider the difference between wanted channel frequency and the current channel fre-

quency. Unnecessary charge pump action will result in very low tuning voltage which may drive the oscillator to extreme conditions". Set `cb_first_if_lower_freq` to 1, if this check is required for this tuner. I tested this for PAL by first setting the TV frequency to 203 MHz and then switching to 96.6 MHz FM radio. The result was static unless the control byte was sent first.

has_tda9887 Set to 1 if this tuner uses a tda9887

port1_fm_high_sensitivity Many Philips tuners use tda9887 PORT1 to select the FM radio sensitivity. If this setting is 1, then set PORT1 to 1 to get proper FM reception.

port2_fm_high_sensitivity Some Philips tuners use tda9887 PORT2 to select the FM radio sensitivity. If this setting is 1, then set PORT2 to 1 to get proper FM reception.

fm_gain_normal Some Philips tuners use tda9887 `cGainNormal` to select the FM radio sensitivity. If this setting is 1, e register will use `cGainNormal` instead of `cGainLow`.

intercarrier_mode Most tuners with a tda9887 use QSS mode. Some (cheaper) tuners use Intercarrier mode. If this setting is 1, then the tuner needs to be set to intercarrier mode.

port1_active This setting sets the default value for PORT1. 0 means inactive, 1 means active. Note: the actual bit value written to the tda9887 is inverted. So a 0 here means a 1 in the B6 bit.

port2_active This setting sets the default value for PORT2. 0 means inactive, 1 means active. Note: the actual bit value written to the tda9887 is inverted. So a 0 here means a 1 in the B7 bit.

port1_invert_for_secam_lc Sometimes PORT1 is inverted when the SECAM-L' standard is selected. Set this bit to 1 if this is needed.

port2_invert_for_secam_lc Sometimes PORT2 is inverted when the SECAM-L' standard is selected. Set this bit to 1 if this is needed.

port1_set_for_fm_mono Some cards require PORT1 to be 1 for mono Radio FM and 0 for stereo.

default_pll_gating_18 Select 18% (or according to datasheet 0%) L standard PLL gating, vs the driver default of 36%.

radio_if IF to use in radio mode. Tuners with a separate radio IF filter seem to use 10.7, while those without use 33.3 for PAL/SECAM tuners and 41.3 for NTSC tuners. 0 = 10.7, 1 = 33.3, 2 = 41.3

default_top_low Default tda9887 TOP value in dB for the low band. Default is 0. Range: -16:+15

default_top_mid Default tda9887 TOP value in dB for the mid band. Default is 0. Range: -16:+15

default_top_high Default tda9887 TOP value in dB for the high band. Default is 0. Range: -16:+15

default_top_secam_low Default tda9887 TOP value in dB for SECAM-L/L' for the low band. Default is 0. Several tuners require a different TOP value for the SECAM-L/L' standards. Range: -16:+15

default_top_secam_mid Default tda9887 TOP value in dB for SECAM-L/L' for the mid band. Default is 0. Several tuners require a different TOP value for the SECAM-L/L' standards. Range: -16:+15

default_top_secam_high Default tda9887 TOP value in dB for SECAM-L/L' for the high band. Default is 0. Several tuners require a different TOP value for the SECAM-L/L' standards. Range: -16:+15

iffreq Intermediate frequency (IF) used by the tuner on digital mode.

count Size of the ranges array.

ranges Array with the frequency ranges supported by the tuner.

struct **tunertype**

describes the known tuners.

Definition

```
struct tunertype {
    char *name;
    unsigned int count;
    struct tuner_params *params;
    u16 min;
    u16 max;
    u32 stepsize;
    u8 *initdata;
    u8 *sleepdata;
};
```

Members

name string with the tuner's name.

count size of *struct tuner_params* array.

params pointer to *struct tuner_params* array.

min minimal tuner frequency, in 62.5 kHz step. should be multiplied to 16 to convert to MHz.

max minimal tuner frequency, in 62.5 kHz step. Should be multiplied to 16 to convert to MHz.

stepsize frequency step, in Hz.

initdata optional byte sequence to initialize the tuner.

sleepdata optional byte sequence to power down the tuner.

2.2.25 V4L2 common functions and data structures

int **v4l2_ctrl_query_fill**(struct *v4l2_queryctrl* *qctrl, s32 *min*, s32 *max*, s32 *step*, s32 *def*)
Fill in a *struct v4l2_queryctrl*

Parameters

struct v4l2_queryctrl *qctrl pointer to the *struct v4l2_queryctrl* to be filled

s32 min minimum value for the control

s32 max maximum value for the control

s32 step control step

s32 def default value for the control

Description

Fills the *struct v4l2_queryctrl* fields for the query control.

Note: This function assumes that the **qctrl->id** field is filled.

Returns -EINVAL if the control is not known by the V4L2 core, 0 on success.

enum v4l2_i2c_tuner_type

specifies the range of tuner address that should be used when seeking for I2C devices.

Constants

ADDRS_RADIO Radio tuner addresses. Represent the following I2C addresses: 0x10 (if compiled with tea5761 support) and 0x60.

ADDRS_DEMOD Demod tuner addresses. Represent the following I2C addresses: 0x42, 0x43, 0x4a and 0x4b.

ADDRS_TV TV tuner addresses. Represent the following I2C addresses: 0x42, 0x43, 0x4a, 0x4b, 0x60, 0x61, 0x62, 0x63 and 0x64.

ADDRS_TV_WITH_DEMOD TV tuner addresses if demod is present, this excludes addresses used by the demodulator from the list of candidates. Represent the following I2C addresses: 0x60, 0x61, 0x62, 0x63 and 0x64.

NOTE

All I2C addresses above use the 7-bit notation.

```
struct v4l2_subdev * v4l2_i2c_new_subdev(struct v4l2_device *v4l2_dev,
                                         struct i2c_adapter *adapter, const
                                         char *client_type, u8 addr, const unsigned
                                         short *probe_addrs)
```

Load an i2c module and return an initialized *struct v4l2_subdev*.

Parameters

struct v4l2_device *v4l2_dev pointer to *struct v4l2_device*

struct i2c_adapter *adapter pointer to *struct i2c_adapter*

const char *client_type name of the chip that's on the adapter.

u8 addr I2C address. If zero, it will use **probe_addrs**

const unsigned short *probe_addrs array with a list of address. The last entry at such array should be I2C_CLIENT_END.

Description

returns a *struct v4l2_subdev* pointer.

```
struct v4l2_subdev * v4l2_i2c_new_subdev_board(struct v4l2_device *v4l2_dev,
                                              struct i2c_adapter *adapter, struct
                                              i2c_board_info *info, const unsigned
                                              short *probe_addrs)
```

Load an i2c module and return an initialized *struct v4l2_subdev*.

Parameters

struct v4l2_device *v4l2_dev pointer to *struct v4l2_device*

struct i2c_adapter *adapter pointer to *struct i2c_adapter*

struct i2c_board_info *info pointer to *struct i2c_board_info* used to replace the *irq*, *platform_data* and *addr* arguments.

const unsigned short *probe_addrs array with a list of address. The last entry at such array should be *I2C_CLIENT_END*.

Description

returns a *struct v4l2_subdev* pointer.

void v4l2_i2c_subdev_set_name(*struct v4l2_subdev *sd*, *struct i2c_client *client*, *const char *devname*, *const char *postfix*)

Set name for an I²C sub-device

Parameters

struct v4l2_subdev *sd pointer to *struct v4l2_subdev*

struct i2c_client *client pointer to *struct i2c_client*

const char *devname the name of the device; if NULL, the I²C device's name will be used

const char *postfix sub-device specific string to put right after the I²C device name; may be NULL

void v4l2_i2c_subdev_init(*struct v4l2_subdev *sd*, *struct i2c_client *client*, *const struct v4l2_subdev_ops *ops*)

Initializes a *struct v4l2_subdev* with data from an *i2c_client* struct.

Parameters

struct v4l2_subdev *sd pointer to *struct v4l2_subdev*

struct i2c_client *client pointer to *struct i2c_client*

const struct v4l2_subdev_ops *ops pointer to *struct v4l2_subdev_ops*

unsigned short v4l2_i2c_subdev_addr(*struct v4l2_subdev *sd*)
returns *i2c_client* address of *struct v4l2_subdev*.

Parameters

struct v4l2_subdev *sd pointer to *struct v4l2_subdev*

Description

Returns the address of an I2C sub-device

const unsigned short *v4l2_i2c_tuner_addrs(*enum v4l2_i2c_tuner_type type*)
Return a list of I2C tuner addresses to probe.

Parameters

enum v4l2_i2c_tuner_type type type of the tuner to seek, as defined by *enum v4l2_i2c_tuner_type*.

NOTE

Use only if the tuner addresses are unknown.

void **v4l2_i2c_subdev_unregister**(struct *v4l2_subdev* *sd)
 Unregister a v4l2_subdev

Parameters

struct v4l2_subdev *sd pointer to *struct v4l2_subdev*

struct *v4l2_subdev* * **v4l2_spi_new_subdev**(struct *v4l2_device* *v4l2_dev,
 struct spi_master *master, struct
 spi_board_info *info)
 Load an spi module and return an initialized *struct v4l2_subdev*.

Parameters

struct v4l2_device *v4l2_dev pointer to *struct v4l2_device*.

struct spi_master *master pointer to struct spi_master.

struct spi_board_info *info pointer to struct spi_board_info.

Description

returns a *struct v4l2_subdev* pointer.

void **v4l2_spi_subdev_init**(struct *v4l2_subdev* *sd, struct spi_device *spi, const struct
v4l2_subdev_ops *ops)
 Initialize a v4l2_subdev with data from an spi_device struct.

Parameters

struct v4l2_subdev *sd pointer to *struct v4l2_subdev*

struct spi_device *spi pointer to struct spi_device.

const struct v4l2_subdev_ops *ops pointer to *struct v4l2_subdev_ops*

void **v4l2_spi_subdev_unregister**(struct *v4l2_subdev* *sd)
 Unregister a v4l2_subdev

Parameters

struct v4l2_subdev *sd pointer to *struct v4l2_subdev*

void **v4l_bound_align_image**(unsigned int *width, unsigned int wmin, unsigned
 int wmax, unsigned int walign, unsigned int *height,
 unsigned int hmin, unsigned int hmax, unsigned int halign,
 unsigned int salign)
 adjust video dimensions according to a given constraints.

Parameters

unsigned int *width pointer to width that will be adjusted if needed.

unsigned int wmin minimum width.

unsigned int wmax maximum width.

unsigned int walign least significant bit on width.

unsigned int *height pointer to height that will be adjusted if needed.

unsigned int hmin minimum height.

unsigned int hmax maximum height.

unsigned int halign least significant bit on height.

unsigned int salign least significant bit for the image size (e. g. *width * height*).

Description

Clip an image to have **width** between **wmin** and **wmax**, and **height** between **hmin** and **hmax**, inclusive.

Additionally, the **width** will be a multiple of 2^{walign} , the **height** will be a multiple of 2^{halign} , and the overall size *width * height* will be a multiple of 2^{salign} .

Note:

1. The clipping rectangle may be shrunk or enlarged to fit the alignment constraints.
2. **wmax** must not be smaller than **wmin**.
3. **hmax** must not be smaller than **hmin**.
4. The alignments must not be so high there are no possible image sizes within the allowed bounds.
5. **wmin** and **hmin** must be at least 1 (don't use 0).
6. For **walign**, **halign** and **salign**, if you don't care about a certain alignment, specify 0, as $2^0 = 1$ and one byte alignment is equivalent to no alignment.
7. If you only want to adjust downward, specify a maximum that's the same as the initial value.

v4l2_find_nearest_size(*array, array_size, width_field, height_field, width, height*)

Find the nearest size among a discrete set of resolutions contained in an array of a driver specific struct.

Parameters

array a driver specific array of image sizes

array_size the length of the driver specific array of image sizes

width_field the name of the width field in the driver specific struct

height_field the name of the height field in the driver specific struct

width desired width.

height desired height.

Description

Finds the closest resolution to minimize the width and height differences between what requested and the supported resolutions. The size of the width and height fields in the driver specific must equal to that of u32, i.e. four bytes.

Returns the best match or NULL if the length of the array is zero.

int **v4l2_g_parm_cap**(struct *video_device* *vdev, struct *v4l2_subdev* *sd, struct *v4l2_streamparm* *a)
helper routine for vidioc_g_parm to fill this in by calling the g_frame_interval op of the given

subdev. It only works for V4L2_BUF_TYPE_VIDEO_CAPTURE(_MPLANE), hence the `_cap` in the function name.

Parameters

struct video_device *vdev the *struct video_device* pointer. Used to determine the device caps.

struct v4l2_subdev *sd the sub-device pointer.

struct v4l2_streamparm *a the VIDIOC_G_PARM argument.

int **v4l2_s_parm_cap**(struct *video_device* *vdev, struct *v4l2_subdev* *sd, struct *v4l2_streamparm* *a)
 helper routine for `vidioc_s_parm` to fill this in by calling the `s_frame_interval` op of the given subdev. It only works for V4L2_BUF_TYPE_VIDEO_CAPTURE(_MPLANE), hence the `_cap` in the function name.

Parameters

struct video_device *vdev the *struct video_device* pointer. Used to determine the device caps.

struct v4l2_subdev *sd the sub-device pointer.

struct v4l2_streamparm *a the VIDIOC_S_PARM argument.

enum **v4l2_pixel_encoding**
 specifies the pixel encoding value

Constants

V4L2_PIXEL_ENC_UNKNOWN Pixel encoding is unknown/un-initialized

V4L2_PIXEL_ENC_YUV Pixel encoding is YUV

V4L2_PIXEL_ENC_RGB Pixel encoding is RGB

V4L2_PIXEL_ENC_BAYER Pixel encoding is Bayer

struct **v4l2_format_info**
 information about a V4L2 format

Definition

```
struct v4l2_format_info {
    u32 format;
    u8 pixel_enc;
    u8 mem_planes;
    u8 comp_planes;
    u8 bpp[4];
    u8 hdiv;
    u8 vdiv;
    u8 block_w[4];
    u8 block_h[4];
};
```

Members

format 4CC format identifier (V4L2_PIX_FMT_*)

pixel_enc Pixel encoding (see *enum v4l2_pixel_encoding* above)

mem_planes Number of memory planes, which includes the alpha plane (1 to 4).

comp_planes Number of component planes, which includes the alpha plane (1 to 4).

bpp Array of per-plane bytes per pixel

hdiv Horizontal chroma subsampling factor

vdiv Vertical chroma subsampling factor

block_w Per-plane macroblock pixel width (optional)

block_h Per-plane macroblock pixel height (optional)

s64 **v4l2_get_link_freq**(struct *v4l2_ctrl_handler* *handler, unsigned int mul, unsigned int div)

Get link rate from transmitter

Parameters

struct v4l2_ctrl_handler *handler The transmitter's control handler

unsigned int mul The multiplier between pixel rate and link frequency. Bits per pixel on D-PHY, samples per clock on parallel. 0 otherwise.

unsigned int div The divisor between pixel rate and link frequency. Number of data lanes times two on D-PHY, 1 on parallel. 0 otherwise.

Description

This function is intended for obtaining the link frequency from the transmitter sub-devices. It returns the link rate, either from the V4L2_CID_LINK_FREQ control implemented by the transmitter, or value calculated based on the V4L2_CID_PIXEL_RATE implemented by the transmitter.

Returns link frequency on success, otherwise a negative error code: -ENOENT: Link frequency or pixel rate control not found -EINVAL: Invalid link frequency value

struct **v4l2_ioctl_ops**

describe operations for each V4L2 ioctl

Definition

```
struct v4l2_ioctl_ops {
    int (*vidioc_querycap)(struct file *file, void *fh, struct v4l2_capability *cap);
    int (*vidioc_enum_fmt_vid_cap)(struct file *file, void *fh, struct v4l2_fmtdesc *f);
    int (*vidioc_enum_fmt_vid_overlay)(struct file *file, void *fh, struct v4l2_fmtdesc *f);
    int (*vidioc_enum_fmt_vid_out)(struct file *file, void *fh, struct v4l2_fmtdesc *f);
    int (*vidioc_enum_fmt_sdr_cap)(struct file *file, void *fh, struct v4l2_fmtdesc *f);
    int (*vidioc_enum_fmt_sdr_out)(struct file *file, void *fh, struct v4l2_fmtdesc *f);
    int (*vidioc_enum_fmt_meta_cap)(struct file *file, void *fh, struct v4l2_fmtdesc *f);
    int (*vidioc_enum_fmt_meta_out)(struct file *file, void *fh, struct v4l2_fmtdesc *f);
    int (*vidioc_g_fmt_vid_cap)(struct file *file, void *fh, struct v4l2_format *f);
    int (*vidioc_g_fmt_vid_overlay)(struct file *file, void *fh, struct v4l2_format *f);
    int (*vidioc_g_fmt_vid_out)(struct file *file, void *fh, struct v4l2_format *f);
    int (*vidioc_g_fmt_vid_out_overlay)(struct file *file, void *fh, struct v4l2_format *f);
    int (*vidioc_g_fmt_vbi_cap)(struct file *file, void *fh, struct v4l2_format *f);
    int (*vidioc_g_fmt_vbi_out)(struct file *file, void *fh, struct v4l2_format *f);
    int (*vidioc_g_fmt_sliced_vbi_cap)(struct file *file, void *fh, struct v4l2_format *f);
};
```

```

int (*vidioc_g_fmt_sliced_vbi_out)(struct file *file, void *fh, struct v4l2_format
↳*f);
int (*vidioc_g_fmt_vid_cap_mplane)(struct file *file, void *fh, struct v4l2_format
↳*f);
int (*vidioc_g_fmt_vid_out_mplane)(struct file *file, void *fh, struct v4l2_format
↳*f);
int (*vidioc_g_fmt_sdr_cap)(struct file *file, void *fh, struct v4l2_format *f);
int (*vidioc_g_fmt_sdr_out)(struct file *file, void *fh, struct v4l2_format *f);
int (*vidioc_g_fmt_meta_cap)(struct file *file, void *fh, struct v4l2_format *f);
int (*vidioc_g_fmt_meta_out)(struct file *file, void *fh, struct v4l2_format *f);
int (*vidioc_s_fmt_vid_cap)(struct file *file, void *fh, struct v4l2_format *f);
int (*vidioc_s_fmt_vid_overlay)(struct file *file, void *fh, struct v4l2_format *f);
int (*vidioc_s_fmt_vid_out)(struct file *file, void *fh, struct v4l2_format *f);
int (*vidioc_s_fmt_vid_out_overlay)(struct file *file, void *fh, struct v4l2_format
↳*f);
int (*vidioc_s_fmt_vbi_cap)(struct file *file, void *fh, struct v4l2_format *f);
int (*vidioc_s_fmt_vbi_out)(struct file *file, void *fh, struct v4l2_format *f);
int (*vidioc_s_fmt_sliced_vbi_cap)(struct file *file, void *fh, struct v4l2_format
↳*f);
int (*vidioc_s_fmt_sliced_vbi_out)(struct file *file, void *fh, struct v4l2_format
↳*f);
int (*vidioc_s_fmt_vid_cap_mplane)(struct file *file, void *fh, struct v4l2_format
↳*f);
int (*vidioc_s_fmt_vid_out_mplane)(struct file *file, void *fh, struct v4l2_format
↳*f);
int (*vidioc_s_fmt_sdr_cap)(struct file *file, void *fh, struct v4l2_format *f);
int (*vidioc_s_fmt_sdr_out)(struct file *file, void *fh, struct v4l2_format *f);
int (*vidioc_s_fmt_meta_cap)(struct file *file, void *fh, struct v4l2_format *f);
int (*vidioc_s_fmt_meta_out)(struct file *file, void *fh, struct v4l2_format *f);
int (*vidioc_try_fmt_vid_cap)(struct file *file, void *fh, struct v4l2_format *f);
int (*vidioc_try_fmt_vid_overlay)(struct file *file, void *fh, struct v4l2_format
↳*f);
int (*vidioc_try_fmt_vid_out)(struct file *file, void *fh, struct v4l2_format *f);
int (*vidioc_try_fmt_vid_out_overlay)(struct file *file, void *fh, struct v4l2_
↳format *f);
int (*vidioc_try_fmt_vbi_cap)(struct file *file, void *fh, struct v4l2_format *f);
int (*vidioc_try_fmt_vbi_out)(struct file *file, void *fh, struct v4l2_format *f);
int (*vidioc_try_fmt_sliced_vbi_cap)(struct file *file, void *fh, struct v4l2_format
↳*f);
int (*vidioc_try_fmt_sliced_vbi_out)(struct file *file, void *fh, struct v4l2_format
↳*f);
int (*vidioc_try_fmt_vid_cap_mplane)(struct file *file, void *fh, struct v4l2_format
↳*f);
int (*vidioc_try_fmt_vid_out_mplane)(struct file *file, void *fh, struct v4l2_format
↳*f);
int (*vidioc_try_fmt_sdr_cap)(struct file *file, void *fh, struct v4l2_format *f);
int (*vidioc_try_fmt_sdr_out)(struct file *file, void *fh, struct v4l2_format *f);
int (*vidioc_try_fmt_meta_cap)(struct file *file, void *fh, struct v4l2_format *f);
int (*vidioc_try_fmt_meta_out)(struct file *file, void *fh, struct v4l2_format *f);
int (*vidioc_reqbufs)(struct file *file, void *fh, struct v4l2_requestbuffers *b);
int (*vidioc_querybuf)(struct file *file, void *fh, struct v4l2_buffer *b);
int (*vidioc_qbuf)(struct file *file, void *fh, struct v4l2_buffer *b);
int (*vidioc_expbuf)(struct file *file, void *fh, struct v4l2_exportbuffer *e);
int (*vidioc_dqbuf)(struct file *file, void *fh, struct v4l2_buffer *b);
int (*vidioc_create_bufs)(struct file *file, void *fh, struct v4l2_create_buffers
↳*b);
int (*vidioc_prepare_buf)(struct file *file, void *fh, struct v4l2_buffer *b);

```

```

int (*vidioc_overlay)(struct file *file, void *fh, unsigned int i);
int (*vidioc_g_fbbuf)(struct file *file, void *fh, struct v4l2_framebuffer *a);
int (*vidioc_s_fbbuf)(struct file *file, void *fh, const struct v4l2_framebuffer *a);
int (*vidioc_streamon)(struct file *file, void *fh, enum v4l2_buf_type i);
int (*vidioc_streamoff)(struct file *file, void *fh, enum v4l2_buf_type i);
int (*vidioc_g_std)(struct file *file, void *fh, v4l2_std_id *norm);
int (*vidioc_s_std)(struct file *file, void *fh, v4l2_std_id norm);
int (*vidioc_querystd)(struct file *file, void *fh, v4l2_std_id *a);
int (*vidioc_enum_input)(struct file *file, void *fh, struct v4l2_input *inp);
int (*vidioc_g_input)(struct file *file, void *fh, unsigned int *i);
int (*vidioc_s_input)(struct file *file, void *fh, unsigned int i);
int (*vidioc_enum_output)(struct file *file, void *fh, struct v4l2_output *a);
int (*vidioc_g_output)(struct file *file, void *fh, unsigned int *i);
int (*vidioc_s_output)(struct file *file, void *fh, unsigned int i);
int (*vidioc_queryctrl)(struct file *file, void *fh, struct v4l2_queryctrl *a);
int (*vidioc_query_ext_ctrl)(struct file *file, void *fh, struct v4l2_query_ext_ctrl
↪ *a);
int (*vidioc_g_ctrl)(struct file *file, void *fh, struct v4l2_control *a);
int (*vidioc_s_ctrl)(struct file *file, void *fh, struct v4l2_control *a);
int (*vidioc_g_ext_ctrls)(struct file *file, void *fh, struct v4l2_ext_controls *a);
int (*vidioc_s_ext_ctrls)(struct file *file, void *fh, struct v4l2_ext_controls *a);
int (*vidioc_try_ext_ctrls)(struct file *file, void *fh, struct v4l2_ext_controls
↪ *a);
int (*vidioc_querymenu)(struct file *file, void *fh, struct v4l2_querymenu *a);
int (*vidioc_enumaudio)(struct file *file, void *fh, struct v4l2_audio *a);
int (*vidioc_g_audio)(struct file *file, void *fh, struct v4l2_audio *a);
int (*vidioc_s_audio)(struct file *file, void *fh, const struct v4l2_audio *a);
int (*vidioc_enumaudout)(struct file *file, void *fh, struct v4l2_audioout *a);
int (*vidioc_g_audout)(struct file *file, void *fh, struct v4l2_audioout *a);
int (*vidioc_s_audout)(struct file *file, void *fh, const struct v4l2_audioout *a);
int (*vidioc_g_modulator)(struct file *file, void *fh, struct v4l2_modulator *a);
int (*vidioc_s_modulator)(struct file *file, void *fh, const struct v4l2_modulator
↪ *a);
int (*vidioc_g_pixelaspect)(struct file *file, void *fh, int buf_type, struct v4l2_
↪ fract *aspect);
int (*vidioc_g_selection)(struct file *file, void *fh, struct v4l2_selection *s);
int (*vidioc_s_selection)(struct file *file, void *fh, struct v4l2_selection *s);
int (*vidioc_g_jpegcomp)(struct file *file, void *fh, struct v4l2_jpegcompression
↪ *a);
int (*vidioc_s_jpegcomp)(struct file *file, void *fh, const struct v4l2_
↪ jpegcompression *a);
int (*vidioc_g_enc_index)(struct file *file, void *fh, struct v4l2_enc_idx *a);
int (*vidioc_encoder_cmd)(struct file *file, void *fh, struct v4l2_encoder_cmd *a);
int (*vidioc_try_encoder_cmd)(struct file *file, void *fh, struct v4l2_encoder_cmd
↪ *a);
int (*vidioc_decoder_cmd)(struct file *file, void *fh, struct v4l2_decoder_cmd *a);
int (*vidioc_try_decoder_cmd)(struct file *file, void *fh, struct v4l2_decoder_cmd
↪ *a);
int (*vidioc_g_parm)(struct file *file, void *fh, struct v4l2_streamparm *a);
int (*vidioc_s_parm)(struct file *file, void *fh, struct v4l2_streamparm *a);
int (*vidioc_g_tuner)(struct file *file, void *fh, struct v4l2_tuner *a);
int (*vidioc_s_tuner)(struct file *file, void *fh, const struct v4l2_tuner *a);
int (*vidioc_g_frequency)(struct file *file, void *fh, struct v4l2_frequency *a);
int (*vidioc_s_frequency)(struct file *file, void *fh, const struct v4l2_frequency
↪ *a);
int (*vidioc_enum_freq_bands)(struct file *file, void *fh, struct v4l2_frequency_
↪ band *band);

```

```

    int (*vidioc_g_sliced_vbi_cap)(struct file *file, void *fh, struct v4l2_sliced_vbi_
↳cap *a);
    int (*vidioc_log_status)(struct file *file, void *fh);
    int (*vidioc_s_hw_freq_seek)(struct file *file, void *fh, const struct v4l2_hw_freq_
↳seek *a);
#ifdef CONFIG_VIDEO_ADV_DEBUG;
    int (*vidioc_g_register)(struct file *file, void *fh, struct v4l2_dbg_register *reg);
    int (*vidioc_s_register)(struct file *file, void *fh, const struct v4l2_dbg_register_
↳*reg);
    int (*vidioc_g_chip_info)(struct file *file, void *fh, struct v4l2_dbg_chip_info_
↳*chip);
#endif;
    int (*vidioc_enum_framesizes)(struct file *file, void *fh, struct v4l2_frmsizeenum_
↳*fsize);
    int (*vidioc_enum_frameintervals)(struct file *file, void *fh, struct v4l2_
↳frmivalenum *fival);
    int (*vidioc_s_dv_timings)(struct file *file, void *fh, struct v4l2_dv_timings_
↳*timings);
    int (*vidioc_g_dv_timings)(struct file *file, void *fh, struct v4l2_dv_timings_
↳*timings);
    int (*vidioc_query_dv_timings)(struct file *file, void *fh, struct v4l2_dv_timings_
↳*timings);
    int (*vidioc_enum_dv_timings)(struct file *file, void *fh, struct v4l2_enum_dv_
↳timings *timings);
    int (*vidioc_dv_timings_cap)(struct file *file, void *fh, struct v4l2_dv_timings_cap_
↳*cap);
    int (*vidioc_g_edid)(struct file *file, void *fh, struct v4l2_edid *edid);
    int (*vidioc_s_edid)(struct file *file, void *fh, struct v4l2_edid *edid);
    int (*vidioc_subscribe_event)(struct v4l2_fh *fh, const struct v4l2_event_
↳subscription *sub);
    int (*vidioc_unsubscribe_event)(struct v4l2_fh *fh, const struct v4l2_event_
↳subscription *sub);
    long (*vidioc_default)(struct file *file, void *fh, bool valid_prio, unsigned int_
↳cmd, void *arg);
};

```

Members

vidioc_querycap pointer to the function that implements *VIDIOC_QUERYCAP* ioctl

vidioc_enum_fmt_vid_cap pointer to the function that implements *VIDIOC_ENUM_FMT* ioctl logic for video capture in single and multi plane mode

vidioc_enum_fmt_vid_overlay pointer to the function that implements *VIDIOC_ENUM_FMT* ioctl logic for video overlay

vidioc_enum_fmt_vid_out pointer to the function that implements *VIDIOC_ENUM_FMT* ioctl logic for video output in single and multi plane mode

vidioc_enum_fmt_sdr_cap pointer to the function that implements *VIDIOC_ENUM_FMT* ioctl logic for Software Defined Radio capture

vidioc_enum_fmt_sdr_out pointer to the function that implements *VIDIOC_ENUM_FMT* ioctl logic for Software Defined Radio output

vidioc_enum_fmt_meta_cap pointer to the function that implements *VIDIOC_ENUM_FMT* ioctl logic for metadata capture

vidioc_enum_fmt_meta_out pointer to the function that implements *VIDIOC_ENUM_FMT* ioctl logic for metadata output

vidioc_g_fmt_vid_cap pointer to the function that implements *VIDIOC_G_FMT* ioctl logic for video capture in single plane mode

vidioc_g_fmt_vid_overlay pointer to the function that implements *VIDIOC_G_FMT* ioctl logic for video overlay

vidioc_g_fmt_vid_out pointer to the function that implements *VIDIOC_G_FMT* ioctl logic for video out in single plane mode

vidioc_g_fmt_vid_out_overlay pointer to the function that implements *VIDIOC_G_FMT* ioctl logic for video overlay output

vidioc_g_fmt_vbi_cap pointer to the function that implements *VIDIOC_G_FMT* ioctl logic for raw VBI capture

vidioc_g_fmt_vbi_out pointer to the function that implements *VIDIOC_G_FMT* ioctl logic for raw VBI output

vidioc_g_fmt_sliced_vbi_cap pointer to the function that implements *VIDIOC_G_FMT* ioctl logic for sliced VBI capture

vidioc_g_fmt_sliced_vbi_out pointer to the function that implements *VIDIOC_G_FMT* ioctl logic for sliced VBI output

vidioc_g_fmt_vid_cap_mplane pointer to the function that implements *VIDIOC_G_FMT* ioctl logic for video capture in multiple plane mode

vidioc_g_fmt_vid_out_mplane pointer to the function that implements *VIDIOC_G_FMT* ioctl logic for video out in multiplane plane mode

vidioc_g_fmt_sdr_cap pointer to the function that implements *VIDIOC_G_FMT* ioctl logic for Software Defined Radio capture

vidioc_g_fmt_sdr_out pointer to the function that implements *VIDIOC_G_FMT* ioctl logic for Software Defined Radio output

vidioc_g_fmt_meta_cap pointer to the function that implements *VIDIOC_G_FMT* ioctl logic for metadata capture

vidioc_g_fmt_meta_out pointer to the function that implements *VIDIOC_G_FMT* ioctl logic for metadata output

vidioc_s_fmt_vid_cap pointer to the function that implements *VIDIOC_S_FMT* ioctl logic for video capture in single plane mode

vidioc_s_fmt_vid_overlay pointer to the function that implements *VIDIOC_S_FMT* ioctl logic for video overlay

vidioc_s_fmt_vid_out pointer to the function that implements *VIDIOC_S_FMT* ioctl logic for video out in single plane mode

vidioc_s_fmt_vid_out_overlay pointer to the function that implements *VIDIOC_S_FMT* ioctl logic for video overlay output

vidioc_s_fmt_vbi_cap pointer to the function that implements *VIDIOC_S_FMT* ioctl logic for raw VBI capture

vidioc_s_fmt_vbi_out pointer to the function that implements *VIDIOC_S_FMT* ioctl logic for raw VBI output

vidioc_s_fmt_sliced_vbi_cap pointer to the function that implements *VIDIOC_S_FMT* ioctl logic for sliced VBI capture

vidioc_s_fmt_sliced_vbi_out pointer to the function that implements *VIDIOC_S_FMT* ioctl logic for sliced VBI output

vidioc_s_fmt_vid_cap_mplane pointer to the function that implements *VIDIOC_S_FMT* ioctl logic for video capture in multiple plane mode

vidioc_s_fmt_vid_out_mplane pointer to the function that implements *VIDIOC_S_FMT* ioctl logic for video out in multiplane plane mode

vidioc_s_fmt_sdr_cap pointer to the function that implements *VIDIOC_S_FMT* ioctl logic for Software Defined Radio capture

vidioc_s_fmt_sdr_out pointer to the function that implements *VIDIOC_S_FMT* ioctl logic for Software Defined Radio output

vidioc_s_fmt_meta_cap pointer to the function that implements *VIDIOC_S_FMT* ioctl logic for metadata capture

vidioc_s_fmt_meta_out pointer to the function that implements *VIDIOC_S_FMT* ioctl logic for metadata output

vidioc_try_fmt_vid_cap pointer to the function that implements *VIDIOC_TRY_FMT* ioctl logic for video capture in single plane mode

vidioc_try_fmt_vid_overlay pointer to the function that implements *VIDIOC_TRY_FMT* ioctl logic for video overlay

vidioc_try_fmt_vid_out pointer to the function that implements *VIDIOC_TRY_FMT* ioctl logic for video out in single plane mode

vidioc_try_fmt_vid_out_overlay pointer to the function that implements *VIDIOC_TRY_FMT* ioctl logic for video overlay output

vidioc_try_fmt_vbi_cap pointer to the function that implements *VIDIOC_TRY_FMT* ioctl logic for raw VBI capture

vidioc_try_fmt_vbi_out pointer to the function that implements *VIDIOC_TRY_FMT* ioctl logic for raw VBI output

vidioc_try_fmt_sliced_vbi_cap pointer to the function that implements *VIDIOC_TRY_FMT* ioctl logic for sliced VBI capture

vidioc_try_fmt_sliced_vbi_out pointer to the function that implements *VIDIOC_TRY_FMT* ioctl logic for sliced VBI output

vidioc_try_fmt_vid_cap_mplane pointer to the function that implements *VIDIOC_TRY_FMT* ioctl logic for video capture in multiple plane mode

vidioc_try_fmt_vid_out_mplane pointer to the function that implements *VIDIOC_TRY_FMT* ioctl logic for video out in multiplane plane mode

vidioc_try_fmt_sdr_cap pointer to the function that implements *VIDIOC_TRY_FMT* ioctl logic for Software Defined Radio capture

vidioc_try_fmt_sdr_out pointer to the function that implements *VIDIOC_TRY_FMT* ioctl logic for Software Defined Radio output

vidioc_try_fmt_meta_cap pointer to the function that implements *VIDIOC_TRY_FMT* ioctl logic for metadata capture

vidioc_try_fmt_meta_out pointer to the function that implements *VIDIOC_TRY_FMT* ioctl logic for metadata output

vidioc_reqbufs pointer to the function that implements *VIDIOC_REQBUFS* ioctl

vidioc_querybuf pointer to the function that implements *VIDIOC_QUERYBUF* ioctl

vidioc_qbuf pointer to the function that implements *VIDIOC_QBUF* ioctl

vidioc_expbuf pointer to the function that implements *VIDIOC_EXPBUF* ioctl

vidioc_dqbuf pointer to the function that implements *VIDIOC_DQBUF* ioctl

vidioc_create_bufs pointer to the function that implements *VIDIOC_CREATE_BUFS* ioctl

vidioc_prepare_buf pointer to the function that implements *VIDIOC_PREPARE_BUF* ioctl

vidioc_overlay pointer to the function that implements *VIDIOC_OVERLAY* ioctl

vidioc_g_fbuf pointer to the function that implements *VIDIOC_G_FBUF* ioctl

vidioc_s_fbuf pointer to the function that implements *VIDIOC_S_FBUF* ioctl

vidioc_streamon pointer to the function that implements *VIDIOC_STREAMON* ioctl

vidioc_streamoff pointer to the function that implements *VIDIOC_STREAMOFF* ioctl

vidioc_g_std pointer to the function that implements *VIDIOC_G_STD* ioctl

vidioc_s_std pointer to the function that implements *VIDIOC_S_STD* ioctl

vidioc_querystd pointer to the function that implements *VIDIOC_QUERYSTD* ioctl

vidioc_enum_input pointer to the function that implements *VIDIOC_ENUM_INPUT* ioctl

vidioc_g_input pointer to the function that implements *VIDIOC_G_INPUT* ioctl

vidioc_s_input pointer to the function that implements *VIDIOC_S_INPUT* ioctl

vidioc_enum_output pointer to the function that implements *VIDIOC_ENUM_OUTPUT* ioctl

vidioc_g_output pointer to the function that implements *VIDIOC_G_OUTPUT* ioctl

vidioc_s_output pointer to the function that implements *VIDIOC_S_OUTPUT* ioctl

vidioc_queryctrl pointer to the function that implements *VIDIOC_QUERYCTRL* ioctl

vidioc_query_ext_ctrl pointer to the function that implements *VIDIOC_QUERY_EXT_CTRL* ioctl

vidioc_g_ctrl pointer to the function that implements *VIDIOC_G_CTRL* ioctl

vidioc_s_ctrl pointer to the function that implements *VIDIOC_S_CTRL* ioctl

vidioc_g_ext_ctrls pointer to the function that implements *VIDIOC_G_EXT_CTRL* ioctl

vidioc_s_ext_ctrls pointer to the function that implements *VIDIOC_S_EXT_CTRL* ioctl

vidioc_try_ext_ctrls pointer to the function that implements *VIDIOC_TRY_EXT_CTRL* ioctl

vidioc_querymenu pointer to the function that implements *VIDIOC_QUERYMENU* ioctl

vidioc_enumaudio pointer to the function that implements *VIDIOC_ENUMAUDIO* ioctl

vidioc_g_audio pointer to the function that implements *VIDIOC_G_AUDIO* ioctl

vidioc_s_audio pointer to the function that implements *VIDIOC_S_AUDIO* ioctl

vidioc_enumaudout pointer to the function that implements *VIDIOC_ENUMAUDOUT* ioctl

vidioc_g_audout pointer to the function that implements *VIDIOC_G_AUDOUT* ioctl

vidioc_s_audout pointer to the function that implements *VIDIOC_S_AUDOUT* ioctl

vidioc_g_modulator pointer to the function that implements *VIDIOC_G_MODULATOR* ioctl

vidioc_s_modulator pointer to the function that implements *VIDIOC_S_MODULATOR* ioctl

vidioc_g_pixelaspect pointer to the function that implements the pixelaspect part of the *VIDIOC_CROPCAP* ioctl

vidioc_g_selection pointer to the function that implements *VIDIOC_G_SELECTION* ioctl

vidioc_s_selection pointer to the function that implements *VIDIOC_S_SELECTION* ioctl

vidioc_g_jpegcomp pointer to the function that implements *VIDIOC_G_JPEGCOMP* ioctl

vidioc_s_jpegcomp pointer to the function that implements *VIDIOC_S_JPEGCOMP* ioctl

vidioc_g_enc_index pointer to the function that implements *VIDIOC_G_ENC_INDEX* ioctl

vidioc_encoder_cmd pointer to the function that implements *VIDIOC_ENCODER_CMD* ioctl

vidioc_try_encoder_cmd pointer to the function that implements *VIDIOC_TRY_ENCODER_CMD* ioctl

vidioc_decoder_cmd pointer to the function that implements *VIDIOC_DECODER_CMD* ioctl

vidioc_try_decoder_cmd pointer to the function that implements *VIDIOC_TRY_DECODER_CMD* ioctl

vidioc_g_parm pointer to the function that implements *VIDIOC_G_PARM* ioctl

vidioc_s_parm pointer to the function that implements *VIDIOC_S_PARM* ioctl

vidioc_g_tuner pointer to the function that implements *VIDIOC_G_TUNER* ioctl

vidioc_s_tuner pointer to the function that implements *VIDIOC_S_TUNER* ioctl

vidioc_g_frequency pointer to the function that implements *VIDIOC_G_FREQUENCY* ioctl

vidioc_s_frequency pointer to the function that implements *VIDIOC_S_FREQUENCY* ioctl

vidioc_enum_freq_bands pointer to the function that implements *VIDIOC_ENUM_FREQ_BANDS* ioctl

vidioc_g_sliced_vbi_cap pointer to the function that implements *VIDIOC_G_SLICED_VBI_CAP* ioctl

vidioc_log_status pointer to the function that implements *VIDIOC_LOG_STATUS* ioctl

vidioc_s_hw_freq_seek pointer to the function that implements *VIDIOC_S_HW_FREQ_SEEK* ioctl

vidioc_g_register pointer to the function that implements *VIDIOC_DBG_G_REGISTER* ioctl

vidioc_s_register pointer to the function that implements *VIDIOC_DBG_S_REGISTER* ioctl

vidioc_g_chip_info pointer to the function that implements *VIDIOC_DBG_G_CHIP_INFO* ioctl

vidioc_enum_framesizes pointer to the function that implements *VIDIOC_ENUM_FRAMESIZES* ioctl

vidioc_enum_frameintervals pointer to the function that implements *VIDIOC_ENUM_FRAMEINTERVALS* ioctl

vidioc_s_dv_timings pointer to the function that implements *VIDIOC_S_DV_TIMINGS* ioctl

vidioc_g_dv_timings pointer to the function that implements *VIDIOC_G_DV_TIMINGS* ioctl

vidioc_query_dv_timings pointer to the function that implements *VIDIOC_QUERY_DV_TIMINGS* ioctl

vidioc_enum_dv_timings pointer to the function that implements *VIDIOC_ENUM_DV_TIMINGS* ioctl

vidioc_dv_timings_cap pointer to the function that implements *VIDIOC_DV_TIMINGS_CAP* ioctl

vidioc_g_edid pointer to the function that implements *VIDIOC_G_EDID* ioctl

vidioc_s_edid pointer to the function that implements *VIDIOC_S_EDID* ioctl

vidioc_subscribe_event pointer to the function that implements *VIDIOC_SUBSCRIBE_EVENT* ioctl

vidioc_unsubscribe_event pointer to the function that implements *VIDIOC_UNSUBSCRIBE_EVENT* ioctl

vidioc_default pointer used to allow other ioctls

const char * **v4l2_norm_to_name**(v4l2_std_id id)

Ancillary routine to analog TV standard name from its ID.

Parameters

v4l2_std_id id analog TV standard ID.

Return

returns a string with the name of the analog TV standard. If the standard is not found or if **id** points to multiple standard, it returns "Unknown".

void **v4l2_video_std_frame_period**(int id, struct v4l2_fract *frameperiod)

Ancillary routine that fills a struct v4l2_fract pointer with the default framerate fraction.

Parameters

int id analog TV standard ID.

struct v4l2_fract *frameperiod struct v4l2_fract pointer to be filled

int **v4l2_video_std_construct**(struct v4l2_standard *vs, int id, const char *name)

Ancillary routine that fills in the fields of a v4l2_standard structure according to the **id** parameter.

Parameters

struct v4l2_standard *vs struct v4l2_standard pointer to be filled

int id analog TV standard ID.

const char *name name of the standard to be used

Description

Note: This ancillary routine is obsolete. Shouldn't be used on newer drivers.

int **v4l_video_std_enumstd**(struct v4l2_standard *vs, v4l2_std_id id)

Ancillary routine that fills in the fields of a v4l2_standard structure according to the **id** and **vs->index** parameters.

Parameters

struct v4l2_standard *vs struct v4l2_standard pointer to be filled.

v4l2_std_id id analog TV standard ID.

void **v4l_printk_ioctl**(const char *prefix, unsigned int cmd)

Ancillary routine that prints the ioctl in a human-readable format.

Parameters

const char *prefix prefix to be added at the ioctl prints.

unsigned int cmd ioctl name

Description

Note: If prefix != NULL, then it will issue a `printk(KERN_DEBUG "` `s: ", prefix)` `` first.

long int **v4l2_compat_ioctl32**(struct file *file, unsigned int cmd, unsigned long arg)

32 Bits compatibility layer for 64 bits processors

Parameters

struct file *file Pointer to struct file.

unsigned int cmd Ioctl name.

unsigned long arg Ioctl argument.

v4l2_kioctl

Typedef: Typedef used to pass an ioctl handler.

Syntax

```
long v4l2_kioctl (struct file *file, unsigned int cmd, void *arg)
```

Parameters

struct file *file Pointer to struct file.

unsigned int cmd Ioctl name.

void *arg Ioctl argument.

long int **video_usercopy**(struct file *file, unsigned int cmd, unsigned long int arg, [v4l2_kioctl func](#))

copies data from/to userspace memory when an ioctl is issued.

Parameters

struct file *file Pointer to struct file.

unsigned int cmd Ioctl name.

unsigned long int arg Ioctl argument.

v4l2_kioctl func function that will handle the ioctl

Description

Note: This routine should be used only inside the V4L2 core.

long int **video_ioctl2**(struct file *file, unsigned int cmd, unsigned long int arg)
Handles a V4L2 ioctl.

Parameters

struct file *file Pointer to struct file.

unsigned int cmd Ioctl name.

unsigned long int arg Ioctl argument.

Description

Method used to handle an ioctl. Should be used to fill the `v4l2_ioctl_ops.unlocked_ioctl` on all V4L2 drivers.

2.2.26 Hauppauge TV EEPROM functions and data structures

enum **tveeprom_audio_processor**

Specifies the type of audio processor used on a Hauppauge device.

Constants

TVEEPROM_AUDPROC_NONE No audio processor present

TVEEPROM_AUDPROC_INTERNAL The audio processor is internal to the video processor

TVEEPROM_AUDPROC_MSP The audio processor is a MSPXXXX device

TVEEPROM_AUDPROC_OTHER The audio processor is another device

struct **tveeprom**

Contains the fields parsed from Hauppauge eeproms

Definition

```
struct tveeprom {
    u32 has_radio;
    u32 has_ir;
    u32 has_MAC_address;
    u32 tuner_type;
    u32 tuner_formats;
    u32 tuner_hauppauge_model;
    u32 tuner2_type;
    u32 tuner2_formats;
    u32 tuner2_hauppauge_model;
    u32 audio_processor;
```

```

u32 decoder_processor;
u32 model;
u32 revision;
u32 serial_number;
char rev_str[5];
u8 MAC_address[ETH_ALEN];
};

```

Members

has_radio 1 if the device has radio; 0 otherwise.

has_ir If `has_ir == 0`, then it is unknown what the IR capabilities are. Otherwise: bit 0) 1 (= IR capabilities are known); bit 1) IR receiver present; bit 2) IR transmitter (blaster) present.

has_MAC_address 0: no MAC, 1: MAC present, 2: unknown.

tuner_type type of the tuner (TUNER_*, as defined at include/media/tuner.h).

tuner_formats Supported analog TV standards (V4L2_STD_*).

tuner_hauppauge_model Hauppauge's code for the device model number.

tuner2_type type of the second tuner (TUNER_*, as defined at include/media/tuner.h).

tuner2_formats Tuner 2 supported analog TV standards (V4L2_STD_*).

tuner2_hauppauge_model tuner 2 Hauppauge's code for the device model number.

audio_processor analog audio decoder, as defined by *enum tveeprom_audio_processor*.

decoder_processor Hauppauge's code for the decoder chipset. Unused by the drivers, as they probe the decoder based on the PCI or USB ID.

model Hauppauge's model number

revision Card revision number

serial_number Card's serial number

rev_str Card revision converted to number

MAC_address MAC address for the network interface

void **tveeprom_hauppauge_analog**(struct *tveeprom* *tvee, unsigned char *eeprom_data)
Fill *struct tveeprom* using the contents of the eeprom previously filled at **eeprom_data** field.

Parameters

struct tveeprom *tvee Struct to where the eeprom parsed data will be filled;

unsigned char *eeprom_data Array with the contents of the eeprom_data. It should contain 256 bytes filled with the contents of the eeprom read from the Hauppauge device.

int **tveeprom_read**(struct i2c_client *c, unsigned char *eedata, int len)
Reads the contents of the eeprom found at the Hauppauge devices.

Parameters

struct i2c_client *c I2C client struct

unsigned char *eedata Array where the eeprom content will be stored.

int len Size of **eedata** array. If the eeprom content will be later be parsed by `tveeprom_hauppage_analog()`, len should be, at least, 256.

2.3 Digital TV (DVB) devices

Digital TV devices are implemented by several different drivers:

- A bridge driver that is responsible to talk with the bus where the other devices are connected (PCI, USB, SPI), bind to the other drivers and implement the digital demux logic (either in software or in hardware);
- Frontend drivers that are usually implemented as two separate drivers:
 - A tuner driver that implements the logic which commands the part of the hardware responsible for tuning into a digital TV transponder or physical channel. The output of a tuner is usually a baseband or Intermediate Frequency (IF) signal;
 - A demodulator driver (a.k.a “demod”) that implements the logic which commands the digital TV decoding hardware. The output of a demod is a digital stream, with multiple audio, video and data channels typically multiplexed using MPEG Transport Stream¹.

On most hardware, the frontend drivers talk with the bridge driver using an I2C bus.

2.3.1 Digital TV Common functions

2.3.1.1 Math functions

Provide some commonly-used math functions, usually required in order to estimate signal strength and signal to noise measurements in dB.

unsigned int **intlog2**(u32 *value*)
computes log2 of a value; the result is shifted left by 24 bits

Parameters

u32 value The value (must be != 0)

Description

to use rational values you can use the following method:

$$\text{intlog2}(\text{value}) = \text{intlog2}(\text{value} * 2^x) - x * 2^{24}$$

Some usecase examples:

$$\text{intlog2}(8) \text{ will give } 3 \ll 24 = 3 * 2^{24}$$

$$\text{intlog2}(9) \text{ will give } 3 \ll 24 + \dots = 3.16\dots * 2^{24}$$

$$\text{intlog2}(1.5) = \text{intlog2}(3) - 2^{24} = 0.584\dots * 2^{24}$$

Return

$\text{log2}(\text{value}) * 2^{24}$

¹ Some standards use TCP/IP for multiplexing data, like DVB-H (an abandoned standard, not used anymore) and ATSC version 3.0 current proposals. Currently, the DVB subsystem doesn't implement those standards.

unsigned int **intlog10**(u32 *value*)
 computes log10 of a value; the result is shifted left by 24 bits

Parameters

u32 value The value (must be != 0)

Description

to use rational values you can use the following method:

$$\text{intlog10}(\text{value}) = \text{intlog10}(\text{value} * 10^x) - x * 2^{24}$$

An usecase example:

$$\text{intlog10}(1000) \text{ will give } 3 \ll 24 = 3 * 2^{24}$$

due to the implementation $\text{intlog10}(1000)$ might be not exactly $3 * 2^{24}$

look at `intlog2` for similar examples

Return

$\log_{10}(\text{value}) * 2^{24}$

2.3.1.2 DVB devices

Those functions are responsible for handling the DVB device nodes.

enum **dvb_device_type**
 type of the Digital TV device

Constants

DVB_DEVICE_SEC Digital TV standalone Common Interface (CI)

DVB_DEVICE_FRONTEND Digital TV frontend.

DVB_DEVICE_DEMUX Digital TV demux.

DVB_DEVICE_DVR Digital TV digital video record (DVR).

DVB_DEVICE_CA Digital TV Conditional Access (CA).

DVB_DEVICE_NET Digital TV network.

DVB_DEVICE_VIDEO Digital TV video decoder. Deprecated. Used only on av7110-av.

DVB_DEVICE_AUDIO Digital TV audio decoder. Deprecated. Used only on av7110-av.

DVB_DEVICE OSD Digital TV On Screen Display (OSD). Deprecated. Used only on av7110.

struct **dvb_adapter**
 represents a Digital TV adapter using Linux DVB API

Definition

```
struct dvb_adapter {
    int num;
    struct list_head list_head;
    struct list_head device_list;
    const char *name;
    u8 proposed_mac [6];
};
```

```
void* priv;
struct device *device;
struct module *module;
int mfe_shared;
struct dvb_device *mfe_dvbdev;
struct mutex mfe_lock;
#ifdef CONFIG_MEDIA_CONTROLLER_DVB;
struct mutex mdev_lock;
struct media_device *mdev;
struct media_entity *conn;
struct media_pad *conn_pads;
#endif;
};
```

Members

num Number of the adapter

list_head List with the DVB adapters

device_list List with the DVB devices

name Name of the adapter

proposed_mac proposed MAC address for the adapter

priv private data

device pointer to struct device

module pointer to struct module

mfe_shared indicates mutually exclusive frontends. Use of this flag is currently deprecated.

mfe_dvbdev Frontend device in use, in the case of MFE

mfe_lock Lock to prevent using the other frontends when MFE is used.

mdev_lock Protect access to the mdev pointer.

mdev pointer to *struct media_device*, used when the media controller is used.

conn RF connector. Used only if the device has no separate tuner.

conn_pads pointer to *struct media_pad* associated with **conn**;

struct **dvb_device**

represents a DVB device node

Definition

```
struct dvb_device {
    struct list_head list_head;
    const struct file_operations *fops;
    struct dvb_adapter *adapter;
    enum dvb_device_type type;
    int minor;
    u32 id;
    int readers;
    int writers;
    int users;
    wait_queue_head_t wait_queue;
};
```



```

int (*kernel_ioctl)(struct file *file, unsigned int cmd, void *arg);
#ifdef CONFIG_MEDIA_CONTROLLER_DVB;
    const char *name;
    struct media_intf_devnode *intf_devnode;
    unsigned tsout_num_entities;
    struct media_entity *entity, *tsout_entity;
    struct media_pad *pads, *tsout_pads;
#endif;
    void *priv;
};

```

Members

list_head List head with all DVB devices

fops pointer to struct file_operations

adapter pointer to the adapter that holds this device node

type type of the device, as defined by *enum dvb_device_type*.

minor devnode minor number. Major number is always DVB_MAJOR.

id device ID number, inside the adapter

readers Initialized by the caller. Each call to open() in Read Only mode decreases this counter by one.

writers Initialized by the caller. Each call to open() in Read/Write mode decreases this counter by one.

users Initialized by the caller. Each call to open() in any mode decreases this counter by one.

wait_queue wait queue, used to wait for certain events inside one of the DVB API callers

kernel_ioctl callback function used to handle ioctl calls from userspace.

name Name to be used for the device at the Media Controller

intf_devnode Pointer to media_intf_devnode. Used by the dvbdev core to store the MC device node interface

tsout_num_entities Number of Transport Stream output entities

entity pointer to *struct media_entity* associated with the device node

tsout_entity array with MC entities associated to each TS output node

pads pointer to *struct media_pad* associated with **entity**;

tsout_pads array with the source pads for each **tsout_entity**

priv private data

Description

This structure is used by the DVB core (frontend, CA, net, demux) in order to create the device nodes. Usually, driver should not initialize this struct directly.

```

int dvb_register_adapter(struct dvb_adapter *adap, const char *name, struct module *module, struct device *device, short *adapter_nums)
    Registers a new DVB adapter

```

Parameters

struct dvb_adapter *adap pointer to *struct dvb_adapter*

const char *name Adapter's name

struct module *module initialized with THIS_MODULE at the caller

struct device *device pointer to struct device that corresponds to the device driver

short *adapter_nums Array with a list of the numbers for **dvb_register_adapter**; to select among them. Typically, initialized with: DVB_DEFINE_MOD_OPT_ADAPTER_NR(adapter_nums)

int **dvb_unregister_adapter**(struct *dvb_adapter* *adap)
Unregisters a DVB adapter

Parameters

struct dvb_adapter *adap pointer to *struct dvb_adapter*

int **dvb_register_device**(struct *dvb_adapter* *adap, struct *dvb_device* **pdvbdev,
const struct *dvb_device* *template, void *priv, enum
dvb_device_type type, int demux_sink_pads)
Registers a new DVB device

Parameters

struct dvb_adapter *adap pointer to *struct dvb_adapter*

struct dvb_device **pdvbdev pointer to the place where the new *struct dvb_device* will be stored

const struct dvb_device *template Template used to create pdvbdev;

void *priv private data

enum dvb_device_type type type of the device, as defined by *enum dvb_device_type*.

int demux_sink_pads Number of demux outputs, to be used to create the TS outputs via the Media Controller.

void **dvb_remove_device**(struct *dvb_device* *dvbdev)
Remove a registered DVB device

Parameters

struct dvb_device *dvbdev pointer to *struct dvb_device*

Description

This does not free memory. To do that, call *dvb_free_device()*.

void **dvb_free_device**(struct *dvb_device* *dvbdev)
Free memory occupied by a DVB device.

Parameters

struct dvb_device *dvbdev pointer to *struct dvb_device*

Description

Call *dvb_unregister_device()* before calling this function.

void **dvb_unregister_device**(struct *dvb_device* *dvbdev)
Unregisters a DVB device

Parameters

struct dvb_device *dvbdev pointer to *struct dvb_device*

Description

This is a combination of *dvb_remove_device()* and *dvb_free_device()*. Using this function is usually a mistake, and is often an indicator for a use-after-free bug (when a userspace process keeps a file handle to a detached device).

int **dvb_create_media_graph**(struct *dvb_adapter* *adap, bool *create_rf_connector*)
Creates media graph for the Digital TV part of the device.

Parameters

struct dvb_adapter *adap pointer to *struct dvb_adapter*

bool create_rf_connector if true, it creates the RF connector too

Description

This function checks all DVB-related functions at the media controller entities and creates the needed links for the media graph. It is capable of working with multiple tuners or multiple frontends, but it won't create links if the device has multiple tuners and multiple frontends or if the device has multiple muxes. In such case, the caller driver should manually create the remaining links.

void **dvb_register_media_controller**(struct *dvb_adapter* *adap, struct *media_device* *mdev)
registers a media controller at DVB adapter

Parameters

struct dvb_adapter *adap pointer to *struct dvb_adapter*

struct media_device *mdev pointer to *struct media_device*

struct *media_device* * **dvb_get_media_controller**(struct *dvb_adapter* *adap)
gets the associated media controller

Parameters

struct dvb_adapter *adap pointer to *struct dvb_adapter*

int **dvb_generic_open**(struct inode *inode, struct file *file)
Digital TV open function, used by DVB devices

Parameters

struct inode *inode pointer to struct inode.

struct file *file pointer to struct file.

Description

Checks if a DVB devnode is still valid, and if the permissions are OK and increment negative use count.

int **dvb_generic_release**(struct inode *inode, struct file *file)
Digital TV close function, used by DVB devices

Parameters

struct inode *inode pointer to struct inode.

struct file *file pointer to struct file.

Description

Checks if a DVB devnode is still valid, and if the permissions are OK and decrement negative use count.

long **dvb_generic_ioctl**(struct file *file, unsigned int cmd, unsigned long arg)
Digital TV close function, used by DVB devices

Parameters

struct file *file pointer to struct file.

unsigned int cmd Ioctl name.

unsigned long arg Ioctl argument.

Description

Checks if a DVB devnode and struct dvbdev.kernel_ioctl is still valid. If so, calls [dvb_usercopy\(\)](#).

int **dvb_usercopy**(struct file *file, unsigned int cmd, unsigned long arg, int (*func)(struct file *file, unsigned int cmd, void *arg))
copies data from/to userspace memory when an ioctl is issued.

Parameters

struct file *file Pointer to struct file.

unsigned int cmd Ioctl name.

unsigned long arg Ioctl argument.

int (*func)(struct file *file, unsigned int cmd, void *arg) function that will actually handle the ioctl

Description

Ancillary function that uses ioctl direction and size to copy from userspace. Then, it calls **func**, and, if needed, data is copied back to userspace.

struct i2c_client * **dvb_module_probe**(const char *module_name, const char *name, struct i2c_adapter *adap, unsigned char addr, void *platform_data)
helper routine to probe an I2C module

Parameters

const char *module_name

Name of the I2C module to be probed

const char *name

Optional name for the I2C module. Used for debug purposes. If NULL, defaults to **module_name**.

struct i2c_adapter *adap

pointer to struct i2c_adapter that describes the I2C adapter where the module will be bound.

unsigned char addr

I2C address of the adapter, in 7-bit notation.

void *platform_data

Platform data to be passed to the I2C module probed.

Description

This function binds an I2C device into the DVB core. Should be used by all drivers that use I2C bus to control the hardware. A module bound with *dvb_module_probe()* should use *dvb_module_release()* to unbind.

Note: In the past, DVB modules (mainly, frontends) were bound via *dvb_attach()* macro, with does an ugly hack, using I2C low level functions. Such usage is deprecated and will be removed soon. Instead, use this routine.

Return

On success, return an struct *i2c_client*, pointing to the bound I2C device. NULL otherwise.

void **dvb_module_release**(struct *i2c_client* **client*)
 releases an I2C device allocated with *dvb_module_probe()*.

Parameters

struct i2c_client **client* pointer to struct *i2c_client* with the I2C client to be released. can be NULL.

Description

This function should be used to free all resources reserved by *dvb_module_probe()* and unbinding the I2C hardware.

dvb_attach(*FUNCTION*, *ARGS*...)
 attaches a DVB frontend into the DVB core.

Parameters

FUNCTION function on a frontend module to be called.

ARGS... **FUNCTION** arguments.

Description

This ancillary function loads a frontend module in runtime and runs the **FUNCTION** function there, with **ARGS**. As it increments symbol usage cont, at unregister, *dvb_detach()* should be called.

Note: In the past, DVB modules (mainly, frontends) were bound via *dvb_attach()* macro, with does an ugly hack, using I2C low level functions. Such usage is deprecated and will be removed soon. Instead, you should use *dvb_module_probe()*.

dvb_detach(*FUNC*)
 detaches a DVB frontend loaded via *dvb_attach()*

Parameters

FUNC attach function

Description

Decrements usage count for a function previously called via *dvb_attach()*.

2.3.1.3 Digital TV Ring buffer

Those routines implement ring buffers used to handle digital TV data and copy it from/to userspace.

Note:

- 1) For performance reasons read and write routines don't check buffer sizes and/or number of bytes free/available. This has to be done before these routines are called. For example:

```
/* write @buflen: bytes */
free = dvb_ringbuffer_free(rbuf);
if (free >= buflen)
    count = dvb_ringbuffer_write(rbuf, buffer, buflen);
else
    /* do something */

/* read min. 1000, max. @bufsize: bytes */
avail = dvb_ringbuffer_avail(rbuf);
if (avail >= 1000)
    count = dvb_ringbuffer_read(rbuf, buffer, min(avail, bufsize));
else
    /* do something */
```

- 2) If there is exactly one reader and one writer, there is no need to lock read or write operations. Two or more readers must be locked against each other. Flushing the buffer counts as a read operation. Resetting the buffer counts as a read and write operation. Two or more writers must be locked against each other.

struct **dvb_ringbuffer**

Describes a ring buffer used at DVB framework

Definition

```
struct dvb_ringbuffer {
    u8 *data;
    ssize_t size;
    ssize_t pread;
    ssize_t pwrite;
    int error;
    wait_queue_head_t queue;
    spinlock_t lock;
};
```

Members

data Area were the ringbuffer data is written

size size of the ringbuffer

pread next position to read

pwrite next position to write

error used by ringbuffer clients to indicate that an error happened.

queue Wait queue used by ringbuffer clients to indicate when buffer was filled

lock Spinlock used to protect the ringbuffer

void **dvb_ringbuffer_init**(struct *dvb_ringbuffer* *rbuf, void *data, size_t len)
initialize ring buffer, lock and queue

Parameters

struct dvb_ringbuffer *rbuf pointer to *struct dvb_ringbuffer*

void *data pointer to the buffer where the data will be stored

size_t len bytes from ring buffer into **buf**

int **dvb_ringbuffer_empty**(struct *dvb_ringbuffer* *rbuf)
test whether buffer is empty

Parameters

struct dvb_ringbuffer *rbuf pointer to *struct dvb_ringbuffer*

ssize_t **dvb_ringbuffer_free**(struct *dvb_ringbuffer* *rbuf)
returns the number of free bytes in the buffer

Parameters

struct dvb_ringbuffer *rbuf pointer to *struct dvb_ringbuffer*

Return

number of free bytes in the buffer

ssize_t **dvb_ringbuffer_avail**(struct *dvb_ringbuffer* *rbuf)
returns the number of bytes waiting in the buffer

Parameters

struct dvb_ringbuffer *rbuf pointer to *struct dvb_ringbuffer*

Return

number of bytes waiting in the buffer

void **dvb_ringbuffer_reset**(struct *dvb_ringbuffer* *rbuf)
resets the ringbuffer to initial state

Parameters

struct dvb_ringbuffer *rbuf pointer to *struct dvb_ringbuffer*

Description

Resets the read and write pointers to zero and flush the buffer.

This counts as a read and write operation

void **dvb_ringbuffer_flush**(struct *dvb_ringbuffer* *rbuf)
flush buffer

Parameters

struct dvb_ringbuffer *rbuf pointer to *struct dvb_ringbuffer*

void **dvb_ringbuffer_flush_spinlock_wakeup**(struct *dvb_ringbuffer* *rbuf)
flush buffer protected by spinlock and wake-up waiting task(s)

Parameters

struct dvb_ringbuffer *rbuf pointer to *struct dvb_ringbuffer*

DVB_RINGBUFFER_PEEK(*rbuf*, *offs*)
peek at byte **offs** in the buffer

Parameters

rbuf pointer to *struct dvb_ringbuffer*

offs offset inside the ringbuffer

DVB_RINGBUFFER_SKIP(*rbuf*, *num*)
advance read ptr by **num** bytes

Parameters

rbuf pointer to *struct dvb_ringbuffer*

num number of bytes to advance

ssize_t dvb_ringbuffer_read_user(struct *dvb_ringbuffer* *rbuf, u8 __user *buf,
size_t len)
Reads a buffer into a user pointer

Parameters

struct dvb_ringbuffer *rbuf pointer to *struct dvb_ringbuffer*

u8 __user *buf pointer to the buffer where the data will be stored

size_t len bytes from ring buffer into **buf**

Description

This variant assumes that the buffer is a memory at the userspace. So, it will internally call `copy_to_user()`.

Return

number of bytes transferred or -EFAULT

void **dvb_ringbuffer_read**(struct *dvb_ringbuffer* *rbuf, u8 *buf, size_t len)
Reads a buffer into a pointer

Parameters

struct dvb_ringbuffer *rbuf pointer to *struct dvb_ringbuffer*

u8 *buf pointer to the buffer where the data will be stored

size_t len bytes from ring buffer into **buf**

Description

This variant assumes that the buffer is a memory at the Kernel space

Return

number of bytes transferred or -EFAULT

DVB_RINGBUFFER_WRITE_BYTE(*rbuf*, *byte*)
write single byte to ring buffer

Parameters

rbuf pointer to *struct dvb_ringbuffer*

byte byte to write

`ssize_t dvb_ringbuffer_write(struct dvb_ringbuffer *rbuf, const u8 *buf, size_t len)`
Writes a buffer into the ringbuffer

Parameters

struct dvb_ringbuffer *rbuf pointer to *struct dvb_ringbuffer*

const u8 *buf pointer to the buffer where the data will be read

size_t len bytes from ring buffer into **buf**

Description

This variant assumes that the buffer is a memory at the Kernel space

Return

number of bytes transferred or -EFAULT

`ssize_t dvb_ringbuffer_write_user(struct dvb_ringbuffer *rbuf, const u8 __user *buf, size_t len)`
Writes a buffer received via a user pointer

Parameters

struct dvb_ringbuffer *rbuf pointer to *struct dvb_ringbuffer*

const u8 __user *buf pointer to the buffer where the data will be read

size_t len bytes from ring buffer into **buf**

Description

This variant assumes that the buffer is a memory at the userspace. So, it will internally call `copy_from_user()`.

Return

number of bytes transferred or -EFAULT

`ssize_t dvb_ringbuffer_pkt_write(struct dvb_ringbuffer *rbuf, u8 *buf, size_t len)`
Write a packet into the ringbuffer.

Parameters

struct dvb_ringbuffer *rbuf Ringbuffer to write to.

u8 *buf Buffer to write.

size_t len Length of buffer (currently limited to 65535 bytes max).

Return

Number of bytes written, or -EFAULT, -ENOMEM, -EINVAL.

ssize_t **dvb_ringbuffer_pkt_read_user**(struct *dvb_ringbuffer* *rbuf, size_t idx, int offset,
u8 __user *buf, size_t len)

Read from a packet in the ringbuffer.

Parameters

struct dvb_ringbuffer *rbuf Ringbuffer concerned.

size_t idx Packet index as returned by *dvb_ringbuffer_pkt_next()*.

int offset Offset into packet to read from.

u8 __user *buf Destination buffer for data.

size_t len Size of destination buffer.

Return

Number of bytes read, or -EFAULT.

Description

Note: unlike *dvb_ringbuffer_read()*, this does **NOT** update the read pointer in the ringbuffer. You must use *dvb_ringbuffer_pkt_dispose()* to mark a packet as no longer required.

ssize_t **dvb_ringbuffer_pkt_read**(struct *dvb_ringbuffer* *rbuf, size_t idx, int offset,
u8 *buf, size_t len)

Read from a packet in the ringbuffer.

Parameters

struct dvb_ringbuffer *rbuf Ringbuffer concerned.

size_t idx Packet index as returned by *dvb_ringbuffer_pkt_next()*.

int offset Offset into packet to read from.

u8 *buf Destination buffer for data.

size_t len Size of destination buffer.

Note

unlike *dvb_ringbuffer_read_user()*, this DOES update the read pointer in the ringbuffer.

Return

Number of bytes read, or -EFAULT.

void **dvb_ringbuffer_pkt_dispose**(struct *dvb_ringbuffer* *rbuf, size_t idx)
Dispose of a packet in the ring buffer.

Parameters

struct dvb_ringbuffer *rbuf Ring buffer concerned.

size_t idx Packet index as returned by *dvb_ringbuffer_pkt_next()*.

`ssize_t dvb_ringbuffer_pkt_next(struct dvb_ringbuffer *rbuf, size_t idx, size_t *pktlen)`
 Get the index of the next packet in a ringbuffer.

Parameters

struct *dvb_ringbuffer* *rbuf Ringbuffer concerned.

size_t *idx* Previous packet index, or -1 to return the first packet index.

size_t **pktlen* On success, will be updated to contain the length of the packet in bytes. returns Packet index (if ≥ 0), or -1 if no packets available.

2.3.1.4 Digital TV VB2 handler

enum **dvb_buf_type**
 types of Digital TV memory-mapped buffers

Constants

DVB_BUF_TYPE_CAPTURE buffer is filled by the Kernel, with a received Digital TV stream

enum **dvb_vb2_states**
 states to control VB2 state machine

Constants

DVB_VB2_STATE_NONE

VB2 engine not initialized yet, init failed or VB2 was released.

DVB_VB2_STATE_INIT

VB2 engine initialized.

DVB_VB2_STATE_REQBUFS

Buffers were requested

DVB_VB2_STATE_STREAMON

VB2 is streaming. Callers should not check it directly. Instead, they should use *dvb_vb2_is_streaming()*.

Note

Description

Callers should not touch at the state machine directly. This is handled inside *dvb_vb2.c*.

struct **dvb_buffer**
 video buffer information for v4l2.

Definition

```
struct dvb_buffer {
    struct vb2_buffer      vb;
    struct list_head      list;
};
```

Members

vb embedded struct *vb2_buffer*.

list list of *struct dvb_buffer*.

struct **dvb_vb2_ctx**
control struct for VB2 handler

Definition

```
struct dvb_vb2_ctx {
    struct vb2_queue      vb_q;
    struct mutex         mutex;
    spinlock_t          slock;
    struct list_head     dvb_q;
    struct dvb_buffer    *buf;
    int offset;
    int remain;
    int state;
    int buf_siz;
    int buf_cnt;
    int nonblocking;
    enum dmx_buffer_flags flags;
    u32 count;
    char name[DVB_VB2_NAME_MAX + 1];
};
```

Members

vb_q pointer to *struct vb2_queue* with videobuf2 queue.

mutex mutex to serialize vb2 operations. Used by vb2 core `wait_prepare` and `wait_finish` operations.

slock spin lock used to protect buffer filling at `dvb_vb2.c`.

dvb_q List of buffers that are not filled yet.

buf Pointer to the buffer that are currently being filled.

offset index to the next position at the **buf** to be filled.

remain How many bytes are left to be filled at **buf**.

state bitmask of buffer states as defined by *enum dvb_vb2_states*.

buf_siz size of each VB2 buffer.

buf_cnt number of VB2 buffers.

nonblocking

If different than zero, device is operating on non-blocking mode.

flags buffer flags as defined by *enum dmx_buffer_flags*. Filled only at `DMX_DQBUF`. `DMX_QBUF` should zero this field.

count monotonic counter for filled buffers. Helps to identify data stream loses. Filled only at `DMX_DQBUF`. `DMX_QBUF` should zero this field.

name name of the device type. Currently, it can either be "dvr" or "demux_filter".

int **dvb_vb2_init**(struct *dvb_vb2_ctx* *ctx, const char *name, int non_blocking)
initializes VB2 handler

Parameters

struct dvb_vb2_ctx *ctx control struct for VB2 handler

const char *name name for the VB2 handler

int non_blocking

if not zero, it means that the device is at non-blocking mode

int dvb_vb2_release(struct *dvb_vb2_ctx* *ctx)

Releases the VB2 handler allocated resources and put **ctx** at DVB_VB2_STATE_NONE state.

Parameters

struct dvb_vb2_ctx *ctx control struct for VB2 handler

int dvb_vb2_is_streaming(struct *dvb_vb2_ctx* *ctx)

checks if the VB2 handler is streaming

Parameters

struct dvb_vb2_ctx *ctx control struct for VB2 handler

Return

0 if not streaming, 1 otherwise.

int dvb_vb2_fill_buffer(struct *dvb_vb2_ctx* *ctx, const unsigned char *src, int len, enum *dmx_buffer_flags* *buffer_flags)

fills a VB2 buffer

Parameters

struct dvb_vb2_ctx *ctx control struct for VB2 handler

const unsigned char *src place where the data is stored

int len number of bytes to be copied from **src**

enum *dmx_buffer_flags* *buffer_flags

pointer to buffer flags as defined by *enum dmx_buffer_flags*. can be NULL.

__poll_t dvb_vb2_poll(struct *dvb_vb2_ctx* *ctx, struct file *file, poll_table *wait)

Wrapper to *vb2_core_streamon()* for Digital TV buffer handling.

Parameters

struct dvb_vb2_ctx *ctx control struct for VB2 handler

struct file *file struct file argument passed to the poll file operation handler.

poll_table *wait poll_table wait argument passed to the poll file operation handler.

Description

Implements poll syscall() logic.

int dvb_vb2_stream_on(struct *dvb_vb2_ctx* *ctx)

Wrapper to *vb2_core_streamon()* for Digital TV buffer handling.

Parameters

struct dvb_vb2_ctx *ctx control struct for VB2 handler

Description

Starts dvb streaming

int **dvb_vb2_stream_off**(struct *dvb_vb2_ctx* *ctx)
Wrapper to *vb2_core_streamoff()* for Digital TV buffer handling.

Parameters

struct dvb_vb2_ctx *ctx control struct for VB2 handler

Description

Stops dvb streaming

int **dvb_vb2_reqbufs**(struct *dvb_vb2_ctx* *ctx, struct *dmx_requestbuffers* *req)
Wrapper to *vb2_core_reqbufs()* for Digital TV buffer handling.

Parameters

struct dvb_vb2_ctx *ctx control struct for VB2 handler

struct dmx_requestbuffers *req *struct dmx_requestbuffers* passed from userspace in order to handle DMX_REQBUFS.

Description

Initiate streaming by requesting a number of buffers. Also used to free previously requested buffers, is req->count is zero.

int **dvb_vb2_querybuf**(struct *dvb_vb2_ctx* *ctx, struct *dmx_buffer* *b)
Wrapper to *vb2_core_querybuf()* for Digital TV buffer handling.

Parameters

struct dvb_vb2_ctx *ctx control struct for VB2 handler

struct dmx_buffer *b *struct dmx_buffer* passed from userspace in order to handle DMX_QUERYBUF.

int **dvb_vb2_expbuf**(struct *dvb_vb2_ctx* *ctx, struct *dmx_exportbuffer* *exp)
Wrapper to *vb2_core_expbuf()* for Digital TV buffer handling.

Parameters

struct dvb_vb2_ctx *ctx control struct for VB2 handler

struct dmx_exportbuffer *exp *struct dmx_exportbuffer* passed from userspace in order to handle DMX_EXPBUF.

Description

Export a buffer as a file descriptor.

int **dvb_vb2_qbuf**(struct *dvb_vb2_ctx* *ctx, struct *dmx_buffer* *b)
Wrapper to *vb2_core_qbuf()* for Digital TV buffer handling.

Parameters

struct dvb_vb2_ctx *ctx control struct for VB2 handler

struct dmx_buffer *b *struct dmx_buffer* passed from userspace in order to handle DMX_QBUF.

Description

Queue a Digital TV buffer as requested by userspace

```
int dvb_vb2_dqbuf(struct dvb_vb2_ctx *ctx, struct dmx_buffer *b)
    Wrapper to vb2_core_dqbuf() for Digital TV buffer handling.
```

Parameters

struct *dvb_vb2_ctx* *ctx control struct for VB2 handler

struct *dmx_buffer* *b *struct *dmx_buffer** passed from userspace in order to handle DMX_DQBUF.

Description

Dequeue a Digital TV buffer to the userspace

```
int dvb_vb2_mmap(struct dvb_vb2_ctx *ctx, struct vm_area_struct *vma)
    Wrapper to vb2_mmap() for Digital TV buffer handling.
```

Parameters

struct *dvb_vb2_ctx* *ctx control struct for VB2 handler

struct *vm_area_struct* *vma pointer to struct *vm_area_struct* with the vma passed to the mmap file operation handler in the driver.

Description

map Digital TV video buffers into application address space.

2.3.2 Digital TV Frontend kABI

2.3.2.1 Digital TV Frontend

The Digital TV Frontend kABI defines a driver-internal interface for registering low-level, hardware specific driver to a hardware independent frontend layer. It is only of interest for Digital TV device driver writers. The header file for this API is named *dvb_frontend.h* and located in *include/media/*.

Demodulator driver

The demodulator driver is responsible for talking with the decoding part of the hardware. Such driver should implement *dvb_frontend_ops*, which tells what type of digital TV standards are supported, and points to a series of functions that allow the DVB core to command the hardware via the code under *include/media/dvb_frontend.c*.

A typical example of such struct in a driver foo is:

```
static struct dvb_frontend_ops foo_ops = {
    .delsys = { SYS_DVBT, SYS_DVBT2, SYS_DVBC_ANNEX_A },
    .info = {
        .name = "foo DVB-T/T2/C driver",
        .caps = FE_CAN_FEC_1_2 |
                FE_CAN_FEC_2_3 |
                FE_CAN_FEC_3_4 |
```

```

        FE_CAN_FEC_5_6 |
        FE_CAN_FEC_7_8 |
        FE_CAN_FEC_AUTO |
        FE_CAN_QPSK |
        FE_CAN_QAM_16 |
        FE_CAN_QAM_32 |
        FE_CAN_QAM_64 |
        FE_CAN_QAM_128 |
        FE_CAN_QAM_256 |
        FE_CAN_QAM_AUTO |
        FE_CAN_TRANSMISSION_MODE_AUTO |
        FE_CAN_GUARD_INTERVAL_AUTO |
        FE_CAN_HIERARCHY_AUTO |
        FE_CAN_MUTE_TS |
        FE_CAN_2G_MODULATION,
        .frequency_min = 42000000, /* Hz */
        .frequency_max = 1002000000, /* Hz */
        .symbol_rate_min = 870000,
        .symbol_rate_max = 11700000
    },
    .init = foo_init,
    .sleep = foo_sleep,
    .release = foo_release,
    .set_frontend = foo_set_frontend,
    .get_frontend = foo_get_frontend,
    .read_status = foo_get_status_and_stats,
    .tune = foo_tune,
    .i2c_gate_ctrl = foo_i2c_gate_ctrl,
    .get_frontend_algo = foo_get_algo,
};

```

A typical example of such struct in a driver bar meant to be used on Satellite TV reception is:

```

static const struct dvb_frontend_ops bar_ops = {
    .delsys = { SYS_DVBS, SYS_DVBS2 },
    .info = {
        .name = "Bar DVB-S/S2 demodulator",
        .frequency_min = 500000, /* KHz */
        .frequency_max = 2500000, /* KHz */
        .frequency_stepsize = 0,
        .symbol_rate_min = 1000000,
        .symbol_rate_max = 45000000,
        .symbol_rate_tolerance = 500,
        .caps = FE_CAN_INVERSION_AUTO |
                FE_CAN_FEC_AUTO |
                FE_CAN_QPSK,
    },
    .init = bar_init,
    .sleep = bar_sleep,
    .release = bar_release,
    .set_frontend = bar_set_frontend,
    .get_frontend = bar_get_frontend,
    .read_status = bar_get_status_and_stats,
    .i2c_gate_ctrl = bar_i2c_gate_ctrl,
    .get_frontend_algo = bar_get_algo,
    .tune = bar_tune,
};

```



```

/* Satellite-specific */
.diseqc_send_master_cmd = bar_send_diseqc_msg,
.diseqc_send_burst = bar_send_burst,
.set_tone = bar_set_tone,
.set_voltage = bar_set_voltage,
};

```

Note:

- 1) For satellite digital TV standards (DVB-S, DVB-S2, ISDB-S), the frequencies are specified in kHz, while, for terrestrial and cable standards, they're specified in Hz. Due to that, if the same frontend supports both types, you'll need to have two separate *dvb_frontend_ops* structures, one for each standard.
- 2) The `.i2c_gate_ctrl` field is present only when the hardware has allows controlling an I2C gate (either directly or via some GPIO pin), in order to remove the tuner from the I2C bus after a channel is tuned.
- 3) All new drivers should implement the *DVBv5 statistics* via `.read_status`. Yet, there are a number of callbacks meant to get statistics for signal strength, S/N and UCB. Those are there to provide backward compatibility with legacy applications that don't support the DVBv5 API. Implementing those callbacks are optional. Those callbacks may be removed in the future, after we have all existing drivers supporting DVBv5 stats.
- 4) Other callbacks are required for satellite TV standards, in order to control LNBf and DiSEqC: `.diseqc_send_master_cmd`, `.diseqc_send_burst`, `.set_tone`, `.set_voltage`.

The `include/media/dvb_frontend.c` has a kernel thread which is responsible for tuning the device. It supports multiple algorithms to detect a channel, as defined at enum *dvbfe_algo()*.

The algorithm to be used is obtained via `.get_frontend_algo`. If the driver doesn't fill its field at *struct dvb_frontend_ops*, it will default to `DVBFE_ALGO_SW`, meaning that the dvb-core will do a zigzag when tuning, e. g. it will try first to use the specified center frequency f , then, it will do $f + \Delta$, $f - \Delta$, $f + 2 \times \Delta$, $f - 2 \times \Delta$ and so on.

If the hardware has internally a some sort of zigzag algorithm, you should define a `.get_frontend_algo` function that would return `DVBFE_ALGO_HW`.

Note: The core frontend support also supports a third type (`DVBFE_ALGO_CUSTOM`), in order to allow the driver to define its own hardware-assisted algorithm. Very few hardware need to use it nowadays. Using `DVBFE_ALGO_CUSTOM` require to provide other function callbacks at *struct dvb_frontend_ops*.

Attaching frontend driver to the bridge driver

Before using the Digital TV frontend core, the bridge driver should attach the frontend demod, tuner and SEC devices and call `dvb_register_frontend()`, in order to register the new frontend at the subsystem. At device detach/removal, the bridge driver should call `dvb_unregister_frontend()` to remove the frontend from the core and then `dvb_frontend_detach()` to free the memory allocated by the frontend drivers.

The drivers should also call `dvb_frontend_suspend()` as part of their handler for `device_driver.suspend()`, and `dvb_frontend_resume()` as part of their handler for `device_driver.resume()`.

A few other optional functions are provided to handle some special cases.

2.3.2.2 Digital TV Frontend statistics

Introduction

Digital TV frontends provide a range of *statistics* meant to help tuning the device and measuring the quality of service.

For each statistics measurement, the driver should set the type of scale used, or `FE_SCALE_NOT_AVAILABLE` if the statistics is not available on a given time. Drivers should also provide the number of statistics for each type. that's usually 1 for most video standards¹.

Drivers should initialize each statistic counters with length and scale at its init code. For example, if the frontend provides signal strength, it should have, on its init code:

```
struct dtv_frontend_properties *c = &state->fe.dtv_property_cache;

c->strength.len = 1;
c->strength.stat[0].scale = FE_SCALE_NOT_AVAILABLE;
```

And, when the statistics got updated, set the scale:

```
c->strength.stat[0].scale = FE_SCALE_DECIBEL;
c->strength.stat[0].uvalue = strength;
```

Note: Please prefer to use `FE_SCALE_DECIBEL` instead of `FE_SCALE_RELATIVE` for signal strength and CNR measurements.

¹ For ISDB-T, it may provide both a global statistics and a per-layer set of statistics. On such cases, `len` should be equal to 4. The first value corresponds to the global stat; the other ones to each layer, e. g.:

- `c->cnr.stat[0]` for global S/N carrier ratio,
- `c->cnr.stat[1]` for Layer A S/N carrier ratio,
- `c->cnr.stat[2]` for layer B S/N carrier ratio,
- `c->cnr.stat[3]` for layer C S/N carrier ratio.

Groups of statistics

There are several groups of statistics currently supported:

Signal strength (*DTV_STAT_SIGNAL_STRENGTH*)

- Measures the signal strength level at the analog part of the tuner or demod.
- Typically obtained from the gain applied to the tuner and/or frontend in order to detect the carrier. When no carrier is detected, the gain is at the maximum value (so, strength is on its minimal).
- As the gain is visible through the set of registers that adjust the gain, typically, this statistics is always available².
- Drivers should try to make it available all the times, as these statistics can be used when adjusting an antenna position and to check for troubles at the cabling.

Carrier Signal to Noise ratio (*DTV_STAT_CNR*)

- Signal to Noise ratio for the main carrier.
- Signal to Noise measurement depends on the device. On some hardware, it is available when the main carrier is detected. On those hardware, CNR measurement usually comes from the tuner (e. g. after `FE_HAS_CARRIER`, see `fe_status`).

On other devices, it requires inner FEC decoding, as the frontend measures it indirectly from other parameters (e. g. after `FE_HAS_VITERBI`, see `fe_status`).

Having it available after inner FEC is more common.

Bit counts post-FEC (*DTV_STAT_POST_ERROR_BIT_COUNT* and *DTV_STAT_POST_TOTAL_BIT_C*)

- Those counters measure the number of bits and bit errors after the forward error correction (FEC) on the inner coding block (after Viterbi, LDPC or other inner code).
- Due to its nature, those statistics depend on full coding lock (e. g. after `FE_HAS_SYNC` or after `FE_HAS_LOCK`, see `fe_status`).

Bit counts pre-FEC (*DTV_STAT_PRE_ERROR_BIT_COUNT* and *DTV_STAT_PRE_TOTAL_BIT_C*)

- Those counters measure the number of bits and bit errors before the forward error correction (FEC) on the inner coding block (before Viterbi, LDPC or other inner code).
- Not all frontends provide this kind of statistics.
- Due to its nature, those statistics depend on inner coding lock (e. g. after `FE_HAS_VITERBI`, see `fe_status`).

Block counts (*DTV_STAT_ERROR_BLOCK_COUNT* and *DTV-STAT_TOTAL_BLOCK_COUNT*)

- Those counters measure the number of blocks and block errors after the forward error correction (FEC) on the inner coding block (before Viterbi, LDPC or other inner code).

² On a few devices, the gain keeps floating if there is no carrier. On such devices, strength report should check first if carrier is detected at the tuner (`FE_HAS_CARRIER`, see `fe_status`), and otherwise return the lowest possible value.

- Due to its nature, those statistics depend on full coding lock (e. g. after FE_HAS_SYNC or after FE_HAS_LOCK, see fe_status).

Note: All counters should be monotonically increased as they're collected from the hardware.

A typical example of the logic that handle status and statistics is:

```
static int foo_get_status_and_stats(struct dvb_frontend *fe)
{
    struct foo_state *state = fe->demodulator_priv;
    struct dtv_frontend_properties *c = &fe->dtv_property_cache;

    int rc;
    enum fe_status *status;

    /* Both status and strength are always available */
    rc = foo_read_status(fe, &status);
    if (rc < 0)
        return rc;

    rc = foo_read_strength(fe);
    if (rc < 0)
        return rc;

    /* Check if CNR is available */
    if (!(fe->status & FE_HAS_CARRIER))
        return 0;

    rc = foo_read_cnr(fe);
    if (rc < 0)
        return rc;

    /* Check if pre-BER stats are available */
    if (!(fe->status & FE_HAS_VITERBI))
        return 0;

    rc = foo_get_pre_ber(fe);
    if (rc < 0)
        return rc;

    /* Check if post-BER stats are available */
    if (!(fe->status & FE_HAS_SYNC))
        return 0;

    rc = foo_get_post_ber(fe);
    if (rc < 0)
        return rc;
}

static const struct dvb_frontend_ops ops = {
    /* ... */
    .read_status = foo_get_status_and_stats,
};
```

Statistics collection

On almost all frontend hardware, the bit and byte counts are stored by the hardware after a certain amount of time or after the total bit/block counter reaches a certain value (usually programmable), for example, on every 1000 ms or after receiving 1,000,000 bits.

So, if you read the registers too soon, you'll end by reading the same value as in the previous reading, causing the monotonic value to be incremented too often.

Drivers should take the responsibility to avoid too often reads. That can be done using two approaches:

if the driver have a bit that indicates when a collected data is ready

Driver should check such bit before making the statistics available.

An example of such behavior can be found at this code snippet (adapted from mb86a20s driver's logic):

```
static int foo_get_pre_ber(struct dvb_frontend *fe)
{
    struct foo_state *state = fe->demodulator_priv;
    struct dtv_frontend_properties *c = &fe->dtv_property_cache;
    int rc, bit_error;

    /* Check if the BER measures are already available */
    rc = foo_read_u8(state, 0x54);
    if (rc < 0)
        return rc;

    if (!rc)
        return 0;

    /* Read Bit Error Count */
    bit_error = foo_read_u32(state, 0x55);
    if (bit_error < 0)
        return bit_error;

    /* Read Total Bit Count */
    rc = foo_read_u32(state, 0x51);
    if (rc < 0)
        return rc;

    c->pre_bit_error.stat[0].scale = FE_SCALE_COUNTER;
    c->pre_bit_error.stat[0].uvalue += bit_error;
    c->pre_bit_count.stat[0].scale = FE_SCALE_COUNTER;
    c->pre_bit_count.stat[0].uvalue += rc;

    return 0;
}
```

If the driver doesn't provide a statistics available check bit

A few devices, however, may not provide a way to check if the stats are available (or the way to check it is unknown). They may not even provide a way to directly read the total number of bits or blocks.

On those devices, the driver need to ensure that it won't be reading from the register too often and/or estimate the total number of bits/blocks.

On such drivers, a typical routine to get statistics would be like (adapted from dib8000 driver's logic):

```
struct foo_state {
    /* ... */

    unsigned long per_jiffies_stats;
}

static int foo_get_pre_ber(struct dvb_frontend *fe)
{
    struct foo_state *state = fe->demodulator_priv;
    struct dtv_frontend_properties *c = &fe->dtv_property_cache;
    int rc, bit_error;
    u64 bits;

    /* Check if time for stats was elapsed */
    if (!time_after(jiffies, state->per_jiffies_stats))
        return 0;

    /* Next stat should be collected in 1000 ms */
    state->per_jiffies_stats = jiffies + msecs_to_jiffies(1000);

    /* Read Bit Error Count */
    bit_error = foo_read_u32(state, 0x55);
    if (bit_error < 0)
        return bit_error;

    /*
     * On this particular frontend, there's no register that
     * would provide the number of bits per 1000ms sample. So,
     * some function would calculate it based on DTV properties
     */
    bits = get_number_of_bits_per_1000ms(fe);

    c->pre_bit_error.stat[0].scale = FE_SCALE_COUNTER;
    c->pre_bit_error.stat[0].uvalue += bit_error;
    c->pre_bit_count.stat[0].scale = FE_SCALE_COUNTER;
    c->pre_bit_count.stat[0].uvalue += bits;

    return 0;
}
```

Please notice that, on both cases, we're getting the statistics using the `dvb_frontend_ops . read_status` callback. The rationale is that the frontend core will automatically call this function periodically (usually, 3 times per second, when the frontend is locked).

That warrants that we won't miss to collect a counter and increment the monotonic stats at the

right time.

2.3.2.3 Digital TV Frontend functions and types

struct **dvb_frontend_tune_settings**
parameters to adjust frontend tuning

Definition

```
struct dvb_frontend_tune_settings {
    int min_delay_ms;
    int step_size;
    int max_drift;
};
```

Members

min_delay_ms minimum delay for tuning, in ms

step_size step size between two consecutive frequencies

max_drift maximum drift

NOTE

step_size is in Hz, for terrestrial/cable or kHz for satellite

struct **dvb_tuner_info**
Frontend name and min/max ranges/bandwidths

Definition

```
struct dvb_tuner_info {
    char name[128];
    u32 frequency_min_hz;
    u32 frequency_max_hz;
    u32 frequency_step_hz;
    u32 bandwidth_min;
    u32 bandwidth_max;
    u32 bandwidth_step;
};
```

Members

name name of the Frontend

frequency_min_hz minimal frequency supported in Hz

frequency_max_hz maximum frequency supported in Hz

frequency_step_hz frequency step in Hz

bandwidth_min minimal frontend bandwidth supported

bandwidth_max maximum frontend bandwidth supported

bandwidth_step frontend bandwidth step

struct **analog_parameters**
Parameters to tune into an analog/radio channel

Definition

```
struct analog_parameters {
    unsigned int frequency;
    unsigned int mode;
    unsigned int audmode;
    u64 std;
};
```

Members

frequency Frequency used by analog TV tuner (either in 62.5 kHz step, for TV, or 62.5 Hz for radio)

mode Tuner mode, as defined on enum `v4l2_tuner_type`

audmode Audio mode as defined for the `rxsubchans` field at `videodev2.h`, e. g. `V4L2_TUNER_MODE_*`

std TV standard bitmap as defined at `videodev2.h`, e. g. `V4L2_STD_*`

Description

Hybrid tuners should be supported by both V4L2 and DVB APIs. This struct contains the data that are used by the V4L2 side. To avoid dependencies from V4L2 headers, all enums here are declared as integers.

enum **dvbfe_algo**
defines the algorithm used to tune into a channel

Constants

DVBFE_ALGO_HW Hardware Algorithm - Devices that support this algorithm do everything in hardware and no software support is needed to handle them. Requesting these devices to LOCK is the only thing required, device is supposed to do everything in the hardware.

DVBFE_ALGO_SW Software Algorithm - These are dumb devices, that require software to do everything

DVBFE_ALGO_CUSTOM Customizable Algorithm - Devices having this algorithm can be customized to have specific algorithms in the frontend driver, rather than simply doing a software zigzag. In this case the zigzag maybe hardware assisted or it maybe completely done in hardware. In all cases, usage of this algorithm, in conjunction with the search and track callbacks, utilizes the driver specific algorithm.

DVBFE_ALGO_RECOVERY Recovery Algorithm - These devices have AUTO recovery capabilities from LOCK failure

enum **dvbfe_search**
search callback possible return status

Constants

DVBFE_ALGO_SEARCH_SUCCESS

The frontend search algorithm completed and returned successfully

DVBFE_ALGO_SEARCH_ASLEEP

The frontend search algorithm is sleeping

DVBFE_ALGO_SEARCH_FAILED

The frontend search for a signal failed

DVBFE_ALGO_SEARCH_INVALID

The frontend search algorithm was probably supplied with invalid parameters and the search is an invalid one

DVBFE_ALGO_SEARCH_AGAIN

The frontend search algorithm was requested to search again

DVBFE_ALGO_SEARCH_ERROR

The frontend search algorithm failed due to some error

struct **dvb_tuner_ops**

Tuner information and callbacks

Definition

```
struct dvb_tuner_ops {
    struct dvb_tuner_info info;
    void (*release)(struct dvb_frontend *fe);
    int (*init)(struct dvb_frontend *fe);
    int (*sleep)(struct dvb_frontend *fe);
    int (*suspend)(struct dvb_frontend *fe);
    int (*resume)(struct dvb_frontend *fe);
    int (*set_params)(struct dvb_frontend *fe);
    int (*set_analog_params)(struct dvb_frontend *fe, struct analog_parameters *p);
    int (*set_config)(struct dvb_frontend *fe, void *priv_cfg);
    int (*get_frequency)(struct dvb_frontend *fe, u32 *frequency);
    int (*get_bandwidth)(struct dvb_frontend *fe, u32 *bandwidth);
    int (*get_if_frequency)(struct dvb_frontend *fe, u32 *frequency);
#define TUNER_STATUS_LOCKED 1;
#define TUNER_STATUS_STEREO 2;
    int (*get_status)(struct dvb_frontend *fe, u32 *status);
    int (*get_rf_strength)(struct dvb_frontend *fe, u16 *strength);
    int (*get_afc)(struct dvb_frontend *fe, s32 *afc);
    int (*calc_regs)(struct dvb_frontend *fe, u8 *buf, int buf_len);
    int (*set_frequency)(struct dvb_frontend *fe, u32 frequency);
    int (*set_bandwidth)(struct dvb_frontend *fe, u32 bandwidth);
};
```

Members

info embedded *struct dvb_tuner_info* with tuner properties

release callback function called when frontend is detached. drivers should free any allocated memory.

init callback function used to initialize the tuner device.

sleep callback function used to put the tuner to sleep.

suspend callback function used to inform that the Kernel will suspend.

resume callback function used to inform that the Kernel is resuming from suspend.

set_params callback function used to inform the tuner to tune into a digital TV channel. The properties to be used are stored at *struct dvb_frontend.dtv_property_cache*. The tuner demod can change the parameters to reflect the changes needed for the channel to be

tuned, and update statistics. This is the recommended way to set the tuner parameters and should be used on newer drivers.

set_analog_params callback function used to tune into an analog TV channel on hybrid tuners. It passes **analog_parameters** to the driver.

set_config callback function used to send some tuner-specific parameters.

get_frequency get the actual tuned frequency

get_bandwidth get the bandwidth used by the low pass filters

get_if_frequency get the Intermediate Frequency, in Hz. For baseband, should return 0.

get_status returns the frontend lock status

get_rf_strength returns the RF signal strength. Used mostly to support analog TV and radio. Digital TV should report, instead, via DVBv5 API (*struct dvb_frontend.dtv_property_cache*).

get_afc Used only by analog TV core. Reports the frequency drift due to AFC.

calc_regs callback function used to pass register data settings for simple tuners. Shouldn't be used on newer drivers.

set_frequency Set a new frequency. Shouldn't be used on newer drivers.

set_bandwidth Set a new frequency. Shouldn't be used on newer drivers.

NOTE

frequencies used on **get_frequency** and **set_frequency** are in Hz for terrestrial/cable or kHz for satellite.

struct **analog_demod_info**

Information struct for analog TV part of the demod

Definition

```
struct analog_demod_info {
    char *name;
};
```

Members

name Name of the analog TV demodulator

struct **analog_demod_ops**

Demodulation information and callbacks for analog TV and radio

Definition

```
struct analog_demod_ops {
    struct analog_demod_info info;
    void (*set_params)(struct dvb_frontend *fe, struct analog_parameters *params);
    int (*has_signal)(struct dvb_frontend *fe, ul6 *signal);
    int (*get_afc)(struct dvb_frontend *fe, s32 *afc);
    void (*tuner_status)(struct dvb_frontend *fe);
    void (*standby)(struct dvb_frontend *fe);
    void (*release)(struct dvb_frontend *fe);
    int (*i2c_gate_ctrl)(struct dvb_frontend *fe, int enable);
};
```

```
int (*set_config)(struct dvb_frontend *fe, void *priv_cfg);
};
```

Members

info pointer to *struct analog_demod_info*

set_params callback function used to inform the demod to set the demodulator parameters needed to decode an analog or radio channel. The properties are passed via *struct analog_params*.

has_signal returns 0xffff if has signal, or 0 if it doesn't.

get_afc Used only by analog TV core. Reports the frequency drift due to AFC.

tuner_status callback function that returns tuner status bits, e. g. TUNER_STATUS_LOCKED and TUNER_STATUS_STEREO.

standby set the tuner to standby mode.

release callback function called when frontend is detached. drivers should free any allocated memory.

i2c_gate_ctrl controls the I2C gate. Newer drivers should use I2C mux support instead.

set_config callback function used to send some tuner-specific parameters.

struct **dvb_frontend_internal_info**
Frontend properties and capabilities

Definition

```
struct dvb_frontend_internal_info {
    char name[128];
    u32 frequency_min_hz;
    u32 frequency_max_hz;
    u32 frequency_stepsize_hz;
    u32 frequency_tolerance_hz;
    u32 symbol_rate_min;
    u32 symbol_rate_max;
    u32 symbol_rate_tolerance;
    enum fe_caps caps;
};
```

Members

name Name of the frontend

frequency_min_hz Minimal frequency supported by the frontend.

frequency_max_hz Maximal frequency supported by the frontend.

frequency_stepsize_hz All frequencies are multiple of this value.

frequency_tolerance_hz Frequency tolerance.

symbol_rate_min Minimal symbol rate, in bauds (for Cable/Satellite systems).

symbol_rate_max Maximal symbol rate, in bauds (for Cable/Satellite systems).

symbol_rate_tolerance Maximal symbol rate tolerance, in ppm (for Cable/Satellite systems).

caps Capabilities supported by the frontend, as specified in enum `fe_caps`.

struct `dvb_frontend_ops`

Demodulation information and callbacks for ditiaht TV

Definition

```

struct dvb_frontend_ops {
    struct dvb_frontend_internal_info info;
    u8 delsys[MAX_DELSYS];
    void (*detach)(struct dvb_frontend *fe);
    void (*release)(struct dvb_frontend* fe);
    void (*release_sec)(struct dvb_frontend* fe);
    int (*init)(struct dvb_frontend* fe);
    int (*sleep)(struct dvb_frontend* fe);
    int (*suspend)(struct dvb_frontend *fe);
    int (*resume)(struct dvb_frontend *fe);
    int (*write)(struct dvb_frontend* fe, const u8 buf[], int len);
    int (*tune)(struct dvb_frontend* fe, bool re_tune, unsigned int mode_flags, unsigned
↪ int *delay, enum fe_status *status);
    enum dvbfe_algo (*get_frontend_algo)(struct dvb_frontend *fe);
    int (*set_frontend)(struct dvb_frontend *fe);
    int (*get_tune_settings)(struct dvb_frontend* fe, struct dvb_frontend_tune_settings*
↪ settings);
    int (*get_frontend)(struct dvb_frontend *fe, struct dtv_frontend_properties *props);
    int (*read_status)(struct dvb_frontend *fe, enum fe_status *status);
    int (*read_ber)(struct dvb_frontend* fe, u32* ber);
    int (*read_signal_strength)(struct dvb_frontend* fe, u16* strength);
    int (*read_snr)(struct dvb_frontend* fe, u16* snr);
    int (*read_ucblocks)(struct dvb_frontend* fe, u32* ucblocks);
    int (*diseqc_reset_overload)(struct dvb_frontend* fe);
    int (*diseqc_send_master_cmd)(struct dvb_frontend* fe, struct dvb_diseqc_master_cmd*
↪ cmd);
    int (*diseqc_rcv_slave_reply)(struct dvb_frontend* fe, struct dvb_diseqc_slave_
↪ reply* reply);
    int (*diseqc_send_burst)(struct dvb_frontend *fe, enum fe_sec_mini_cmd minicmd);
    int (*set_tone)(struct dvb_frontend *fe, enum fe_sec_tone_mode tone);
    int (*set_voltage)(struct dvb_frontend *fe, enum fe_sec_voltage voltage);
    int (*enable_high_lnb_voltage)(struct dvb_frontend* fe, long arg);
    int (*dishnetwork_send_legacy_command)(struct dvb_frontend* fe, unsigned long cmd);
    int (*i2c_gate_ctrl)(struct dvb_frontend* fe, int enable);
    int (*ts_bus_ctrl)(struct dvb_frontend* fe, int acquire);
    int (*set_lna)(struct dvb_frontend *);
    enum dvbfe_search (*search)(struct dvb_frontend *fe);
    struct dvb_tuner_ops tuner_ops;
    struct analog_demod_ops analog_ops;
};

```

Members

info embedded *struct dvb_tuner_info* with tuner properties

delsys Delivery systems supported by the frontend

detach callback function called when frontend is detached. drivers should clean up, but not yet free the *struct dvb_frontend* allocation.

release callback function called when frontend is ready to be freed. drivers should free any allocated memory.

release_sec callback function requesting that the Satellite Equipment Control (SEC) driver to release and free any memory allocated by the driver.

init callback function used to initialize the tuner device.

sleep callback function used to put the tuner to sleep.

suspend callback function used to inform that the Kernel will suspend.

resume callback function used to inform that the Kernel is resuming from suspend.

write callback function used by some demod legacy drivers to allow other drivers to write data into their registers. Should not be used on new drivers.

tune callback function used by demod drivers that use **DVBFE_ALGO_HW** to tune into a frequency.

get_frontend_algo returns the desired hardware algorithm.

set_frontend callback function used to inform the demod to set the parameters for demodulating a digital TV channel. The properties to be used are stored at *struct dvb_frontend.dtv_property_cache*. The demod can change the parameters to reflect the changes needed for the channel to be decoded, and update statistics.

get_tune_settings callback function

get_frontend callback function used to inform the parameters actually in use. The properties to be used are stored at *struct dvb_frontend.dtv_property_cache* and update statistics. Please notice that it should not return an error code if the statistics are not available because the demod is not locked.

read_status returns the locking status of the frontend.

read_ber legacy callback function to return the bit error rate. Newer drivers should provide such info via DVBv5 API, e. g. **set_frontend/get_frontend**, implementing this callback only if DVBv3 API compatibility is wanted.

read_signal_strength legacy callback function to return the signal strength. Newer drivers should provide such info via DVBv5 API, e. g. **set_frontend/get_frontend**, implementing this callback only if DVBv3 API compatibility is wanted.

read_snr legacy callback function to return the Signal/Noise rate. Newer drivers should provide such info via DVBv5 API, e. g. **set_frontend/get_frontend**, implementing this callback only if DVBv3 API compatibility is wanted.

read_ucblocks legacy callback function to return the Uncorrected Error Blocks. Newer drivers should provide such info via DVBv5 API, e. g. **set_frontend/get_frontend**, implementing this callback only if DVBv3 API compatibility is wanted.

diseqc_reset_overload callback function to implement the FE_DISEQC_RESET_OVERLOAD() ioctl (only Satellite)

diseqc_send_master_cmd callback function to implement the FE_DISEQC_SEND_MASTER_CMD() ioctl (only Satellite).

diseqc_recv_slave_reply callback function to implement the FE_DISEQC_RECV_SLAVE_REPLY() ioctl (only Satellite)

diseqc_send_burst callback function to implement the FE_DISEQC_SEND_BURST() ioctl (only Satellite).

set_tone callback function to implement the FE_SET_TONE() ioctl (only Satellite).

set_voltage callback function to implement the FE_SET_VOLTAGE() ioctl (only Satellite).

enable_high_lnb_voltage callback function to implement the FE_ENABLE_HIGH_LNB_VOLTAGE() ioctl (only Satellite).

dishnetwork_send_legacy_command callback function to implement the FE_DISHNETWORK_SEND_LEGACY_CMD() ioctl (only Satellite). Drivers should not use this, except when the DVB core emulation fails to provide proper support (e.g. if **set_voltage** takes more than 8ms to work), and when backward compatibility with this legacy API is required.

i2c_gate_ctrl controls the I2C gate. Newer drivers should use I2C mux support instead.

ts_bus_ctrl callback function used to take control of the TS bus.

set_lna callback function to power on/off/auto the LNA.

search callback function used on some custom algo search algos.

tuner_ops pointer to *struct dvb_tuner_ops*

analog_ops pointer to *struct analog_demod_ops*

struct **dtv_frontend_properties**

contains a list of properties that are specific to a digital TV standard.

Definition

```
struct dtv_frontend_properties {
    u32 frequency;
    enum fe_modulation      modulation;
    enum fe_sec_voltage     voltage;
    enum fe_sec_tone_mode   sectone;
    enum fe_spectral_inversion inversion;
    enum fe_code_rate       fec_inner;
    enum fe_transmit_mode   transmission_mode;
    u32 bandwidth_hz;
    enum fe_guard_interval  guard_interval;
    enum fe_hierarchy       hierarchy;
    u32 symbol_rate;
    enum fe_code_rate       code_rate_HP;
    enum fe_code_rate       code_rate_LP;
    enum fe_pilot           pilot;
    enum fe_rolloff         rolloff;
    enum fe_delivery_system delivery_system;
    enum fe_interleaving    interleaving;
    u8 isdbt_partial_reception;
    u8 isdbt_sb_mode;
    u8 isdbt_sb_subchannel;
    u32 isdbt_sb_segment_idx;
    u32 isdbt_sb_segment_count;
    u8 isdbt_layer_enabled;
    struct {
        u8 segment_count;
        enum fe_code_rate fec;
        enum fe_modulation modulation;
        u8 interleaving;
    } layer[3];
};
```

```

u32 stream_id;
u32 scrambling_sequence_index;
u8 atscmh_fic_ver;
u8 atscmh_parade_id;
u8 atscmh_nog;
u8 atscmh_tnog;
u8 atscmh_sgn;
u8 atscmh_prc;
u8 atscmh_rs_frame_mode;
u8 atscmh_rs_frame_ensemble;
u8 atscmh_rs_code_mode_pri;
u8 atscmh_rs_code_mode_sec;
u8 atscmh_sccc_block_mode;
u8 atscmh_sccc_code_mode_a;
u8 atscmh_sccc_code_mode_b;
u8 atscmh_sccc_code_mode_c;
u8 atscmh_sccc_code_mode_d;
u32 lna;
struct dtv_fe_stats      strength;
struct dtv_fe_stats      cnr;
struct dtv_fe_stats      pre_bit_error;
struct dtv_fe_stats      pre_bit_count;
struct dtv_fe_stats      post_bit_error;
struct dtv_fe_stats      post_bit_count;
struct dtv_fe_stats      block_error;
struct dtv_fe_stats      block_count;
};

```

Members

frequency frequency in Hz for terrestrial/cable or in kHz for Satellite

modulation Frontend modulation type

voltage SEC voltage (only Satellite)

sectone SEC tone mode (only Satellite)

inversion Spectral inversion

fec_inner Forward error correction inner Code Rate

transmission_mode Transmission Mode

bandwidth_hz Bandwidth, in Hz. A zero value means that userspace wants to autodetect.

guard_interval Guard Interval

hierarchy Hierarchy

symbol_rate Symbol Rate

code_rate_HP high priority stream code rate

code_rate_LP low priority stream code rate

pilot Enable/disable/autodetect pilot tones

rolloff Rolloff factor (alpha)

delivery_system FE delivery system (e. g. digital TV standard)

interleaving interleaving

isdbt_partial_reception ISDB-T partial reception (only ISDB standard)

isdbt_sb_mode ISDB-T Sound Broadcast (SB) mode (only ISDB standard)

isdbt_sb_subchannel ISDB-T SB subchannel (only ISDB standard)

isdbt_sb_segment_idx ISDB-T SB segment index (only ISDB standard)

isdbt_sb_segment_count ISDB-T SB segment count (only ISDB standard)

isdbt_layer_enabled ISDB Layer enabled (only ISDB standard)

layer ISDB per-layer data (only ISDB standard)

layer.segment_count Segment Count;

layer.fec per layer code rate;

layer.modulation per layer modulation;

layer.interleaving per layer interleaving.

stream_id If different than zero, enable substream filtering, if hardware supports (DVB-S2 and DVB-T2).

scrambling_sequence_index Carries the index of the DVB-S2 physical layer scrambling sequence.

atscmh_fic_ver Version number of the FIC (Fast Information Channel) signaling data (only ATSC-M/H)

atscmh_parade_id Parade identification number (only ATSC-M/H)

atscmh_nog Number of MH groups per MH subframe for a designated parade (only ATSC-M/H)

atscmh_tnog Total number of MH groups including all MH groups belonging to all MH parades in one MH subframe (only ATSC-M/H)

atscmh_sgn Start group number (only ATSC-M/H)

atscmh_prc Parade repetition cycle (only ATSC-M/H)

atscmh_rs_frame_mode Reed Solomon (RS) frame mode (only ATSC-M/H)

atscmh_rs_frame_ensemble RS frame ensemble (only ATSC-M/H)

atscmh_rs_code_mode_pri RS code mode pri (only ATSC-M/H)

atscmh_rs_code_mode_sec RS code mode sec (only ATSC-M/H)

atscmh_sccc_block_mode Series Concatenated Convolutional Code (SCCC) Block Mode (only ATSC-M/H)

atscmh_sccc_code_mode_a SCCC code mode A (only ATSC-M/H)

atscmh_sccc_code_mode_b SCCC code mode B (only ATSC-M/H)

atscmh_sccc_code_mode_c SCCC code mode C (only ATSC-M/H)

atscmh_sccc_code_mode_d SCCC code mode D (only ATSC-M/H)

lna Power ON/OFF/AUTO the Linear Now-noise Amplifier (LNA)

strength DVBv5 API statistics: Signal Strength

cnr DVBv5 API statistics: Signal to Noise ratio of the (main) carrier

pre_bit_error DVBv5 API statistics: pre-Viterbi bit error count

pre_bit_count DVBv5 API statistics: pre-Viterbi bit count

post_bit_error DVBv5 API statistics: post-Viterbi bit error count

post_bit_count DVBv5 API statistics: post-Viterbi bit count

block_error DVBv5 API statistics: block error count

block_count DVBv5 API statistics: block count

NOTE

derivated statistics like Uncorrected Error blocks (UCE) are calculated on userspace.

Description

Only a subset of the properties are needed for a given delivery system. For more info, consult the `media_api.html` with the documentation of the Userspace API.

struct **dvb_frontend**

Frontend structure to be used on drivers.

Definition

```
struct dvb_frontend {
    struct kref refcount;
    struct dvb_frontend_ops ops;
    struct dvb_adapter *dvb;
    void *demodulator_priv;
    void *tuner_priv;
    void *frontend_priv;
    void *sec_priv;
    void *analog_demod_priv;
    struct dtv_frontend_properties dtv_property_cache;
#define DVB_FRONTEND_COMPONENT_TUNER 0;
#define DVB_FRONTEND_COMPONENT_DEMOD 1;
    int (*callback)(void *adapter_priv, int component, int cmd, int arg);
    int id;
    unsigned int exit;
};
```

Members

refcount refcount to keep track of *struct dvb_frontend* references

ops embedded *struct dvb_frontend_ops*

dvb pointer to *struct dvb_adapter*

demodulator_priv demod private data

tuner_priv tuner private data

frontend_priv frontend private data

sec_priv SEC private data

analog_demod_priv Analog demod private data

dtv_property_cache embedded *struct dtv_frontend_properties*

callback callback function used on some drivers to call either the tuner or the demodulator.

id Frontend ID

exit Used to inform the DVB core that the frontend thread should exit (usually, means that the hardware got disconnected).

int **dvb_register_frontend**(struct *dvb_adapter* **dvb*, struct *dvb_frontend* **fe*)
Registers a DVB frontend at the adapter

Parameters

struct dvb_adapter *dvb pointer to *struct dvb_adapter*

struct dvb_frontend *fe pointer to *struct dvb_frontend*

Description

Allocate and initialize the private data needed by the frontend core to manage the frontend and calls *dvb_register_device()* to register a new frontend. It also cleans the property cache that stores the frontend parameters and selects the first available delivery system.

int **dvb_unregister_frontend**(struct *dvb_frontend* **fe*)
Unregisters a DVB frontend

Parameters

struct dvb_frontend *fe pointer to *struct dvb_frontend*

Description

Stops the frontend kthread, calls *dvb_unregister_device()* and frees the private frontend data allocated by *dvb_register_frontend()*.

NOTE

This function doesn't free the memory allocated by the demod, by the SEC driver and by the tuner. In order to free it, an explicit call to *dvb_frontend_detach()* is needed, after calling this function.

void **dvb_frontend_detach**(struct *dvb_frontend* **fe*)
Detaches and frees frontend specific data

Parameters

struct dvb_frontend *fe pointer to *struct dvb_frontend*

Description

This function should be called after *dvb_unregister_frontend()*. It calls the SEC, tuner and demod release functions: *dvb_frontend_ops.release_sec*, *dvb_frontend_ops.tuner_ops.release*, *dvb_frontend_ops.analog_ops.release* and *dvb_frontend_ops.release*.

If the driver is compiled with `CONFIG_MEDIA_ATTACH`, it also decreases the module reference count, needed to allow userspace to remove the previously used DVB frontend modules.

int **dvb_frontend_suspend**(struct *dvb_frontend* **fe*)
Suspends a Digital TV frontend

Parameters

struct dvb_frontend *fe pointer to *struct dvb_frontend*

Description

This function prepares a Digital TV frontend to suspend.

In order to prepare the tuner to suspend, if `dvb_frontend_ops.tuner_ops.suspend()` is available, it calls it. Otherwise, it will call `dvb_frontend_ops.tuner_ops.sleep()`, if available.

It will also call `dvb_frontend_ops.suspend()` to put the demod to suspend, if available. Otherwise it will call `dvb_frontend_ops.sleep()`.

The drivers should also call `dvb_frontend_suspend()` as part of their handler for the `device_driver.suspend()`.

```
int dvb_frontend_resume(struct dvb_frontend *fe)
    Resumes a Digital TV frontend
```

Parameters

struct dvb_frontend *fe pointer to *struct dvb_frontend*

Description

This function resumes the usual operation of the tuner after resume.

In order to resume the frontend, it calls the demod `dvb_frontend_ops.resume()` if available. Otherwise it calls demod `dvb_frontend_ops.init()`.

If `dvb_frontend_ops.tuner_ops.resume()` is available, It, it calls it. Otherwise,t will call `dvb_frontend_ops.tuner_ops.init()`, if available.

Once tuner and demods are resumed, it will enforce that the SEC voltage and tone are restored to their previous values and wake up the frontend's kthread in order to retune the frontend.

The drivers should also call `dvb_frontend_resume()` as part of their handler for the `device_driver.resume()`.

```
void dvb_frontend_reinitialise(struct dvb_frontend *fe)
    forces a reinitialisation at the frontend
```

Parameters

struct dvb_frontend *fe pointer to *struct dvb_frontend*

Description

Calls `dvb_frontend_ops.init()` and `dvb_frontend_ops.tuner_ops.init()`, and resets SEC tone and voltage (for Satellite systems).

NOTE

Currently, this function is used only by one driver (budget-av). It seems to be due to address some special issue with that specific frontend.

```
void dvb_frontend_sleep_until(ktime_t *waketime, u32 add_usec)
    Sleep for the amount of time given by add_usec parameter
```

Parameters

ktime_t *waketime pointer to struct ktime_t

u32 add_usec time to sleep, in microseconds

Description

This function is used to measure the time required for the `FE_DISHNETWORK_SEND_LEGACY_CMD()` ioctl to work. It needs to be as precise as possible, as it affects the detection of the dish tone command at the satellite subsystem.

Its used internally by the DVB frontend core, in order to emulate `FE_DISHNETWORK_SEND_LEGACY_CMD()` using the `dvb_frontend_ops.set_voltage()` callback.

NOTE

it should not be used at the drivers, as the emulation for the legacy callback is provided by the Kernel. The only situation where this should be at the drivers is when there are some bugs at the hardware that would prevent the core emulation to work. On such cases, the driver would be writing a `dvb_frontend_ops.dishnetwork_send_legacy_command()` and calling this function directly.

2.3.3 Digital TV Demux kABI

2.3.3.1 Digital TV Demux

The Kernel Digital TV Demux kABI defines a driver-internal interface for registering low-level, hardware specific driver to a hardware independent demux layer. It is only of interest for Digital TV device driver writers. The header file for this kABI is named `demux.h` and located in `include/media`.

The demux kABI 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 kABI, it receives a pointer to the kABI of that resource.

Each demux receives its TS input from a DVB front-end or from memory, as set via this demux kABI. In a system with more than one front-end, the kABI 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 kABI only controls front-ends regarding to their connections with demuxes; the kABI used to set the other front-end parameters, such as tuning, are devined via the Digital TV Frontend kABI.

The functions that implement the abstract interface demux should be defined static or module private and registered to the Demux core for external access. It is not necessary to implement every function in the struct `dmx_demux`. For example, a demux interface might support Section filtering, but not PES filtering. The kABI client is expected to check the value of any function pointer before calling the function: the value of NULL means that the function is not available.

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.

Note that functions called from a bottom half context must not sleep. Even a simple memory allocation without using `GFP_ATOMIC` 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 kABI function is called from network device code, the function must not sleep.

2.3.3.2 Demux Callback API

This kernel-space API comprises the callback functions that deliver filtered data to the demux client. Unlike the other DVB kABIs, these functions are provided by the client and called from the demux code.

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.

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.

This mechanism is implemented by `dmx_ts_cb()` and `dmx_section_cb()` callbacks.

2.3.3.3 Digital TV Demux device registration functions and data structures

enum **dmxdev_type**
type of demux filter type.

Constants

DMXDEV_TYPE_NONE no filter set.

DMXDEV_TYPE_SEC section filter.

DMXDEV_TYPE_PES Program Elementary Stream (PES) filter.

enum **dmxdev_state**
state machine for the dmxdev.

Constants

DMXDEV_STATE_FREE indicates that the filter is freed.

DMXDEV_STATE_ALLOCATED indicates that the filter was allocated to be used.

DMXDEV_STATE_SET indicates that the filter parameters are set.

DMXDEV_STATE_GO indicates that the filter is running.

DMXDEV_STATE_DONE indicates that a packet was already filtered and the filter is now disabled.
Set only if **DMX_ONESHOT**. See `dmx_sct_filter_params`.

DMXDEV_STATE_TIMEDOUT Indicates a timeout condition.

struct **dmxdev_feed**
digital TV dmxdev feed

Definition

```
struct dmxdev_feed {
    u16 pid;
    struct dmx_ts_feed *ts;
    struct list_head next;
};
```

Members

pid Program ID to be filtered

ts pointer to *struct dmx_ts_feed*

next struct list_head pointing to the next feed.

struct **dmxdev_filter**

digital TV dmxdev filter

Definition

```
struct dmxdev_filter {
    union {
        struct dmx_section_filter *sec;
    } filter;
    union {
        struct list_head ts;
        struct dmx_section_feed *sec;
    } feed;
    union {
        struct dmx_sct_filter_params sec;
        struct dmx_pes_filter_params pes;
    } params;
    enum dmxdev_type type;
    enum dmxdev_state state;
    struct dmxdev *dev;
    struct dvb_ringbuffer buffer;
    struct dvb_vb2_ctx vb2_ctx;
    struct mutex mutex;
    struct timer_list timer;
    int todo;
    u8 secheader[3];
};
```

Members

filter a union describing a dmxdev filter. Currently used only for section filters.

filter.sec a *struct dmx_section_filter* pointer. For section filter only.

feed a union describing a dmxdev feed. Depending on the filter type, it can be either **feed.ts** or **feed.sec**.

feed.ts a struct list_head list. For TS and PES feeds.

feed.sec a *struct dmx_section_feed* pointer. For section feed only.

params a union describing dmxdev filter parameters. Depending on the filter type, it can be either **params.sec** or **params.pes**.

params.sec a *struct dmx_sct_filter_params* embedded struct. For section filter only.

params.pes a *struct dmx_pes_filter_params* embedded struct. For PES filter only.

type type of the dmxdev filter, as defined by *enum dmxdev_type*.

state state of the dmxdev filter, as defined by *enum dmxdev_state*.

dev pointer to *struct dmxdev*.

buffer an embedded *struct dvb_ringbuffer* buffer.

vb2_ctx control struct for VB2 handler

mutex protects the access to *struct dmxdev_filter*.

timer struct *timer_list* embedded timer, used to check for feed timeouts. Only for section filter.

todo index for the **sechheader**. Only for section filter.

sechheader buffer cache to parse the section header. Only for section filter.

struct **dmxdev**

Describes a digital TV demux device.

Definition

```
struct dmxdev {
    struct dvb_device *dvbdev;
    struct dvb_device *dvr_dvbdev;
    struct dmxdev_filter *filter;
    struct dmx_demux *demux;
    int filternum;
    int capabilities;
    unsigned int may_do_mmap:1;
    unsigned int exit:1;
#define DMXDEV_CAP_DUPLEX 1;
    struct dmx_frontend *dvr_orig_fe;
    struct dvb_ringbuffer dvr_buffer;
#define DVR_BUFFER_SIZE (10*188*1024);
    struct dvb_vb2_ctx dvr_vb2_ctx;
    struct mutex mutex;
    spinlock_t lock;
};
```

Members

dvbdev pointer to *struct dvb_device* associated with the demux device node.

dvr_dvbdev pointer to *struct dvb_device* associated with the dvr device node.

filter pointer to *struct dmxdev_filter*.

demux pointer to *struct dmx_demux*.

filternum number of filters.

capabilities demux capabilities as defined by *enum dmx_demux_caps*.

may_do_mmap flag used to indicate if the device may do mmap.

exit flag to indicate that the demux is being released.

dvr_orig_fe pointer to *struct dmx_frontend*.

dvr_buffer embedded *struct dvb_ringbuffer* for DVB output.

dvr_vb2_ctx control struct for VB2 handler

mutex protects the usage of this structure.

lock protects access to *dmxdev->filter->data*.

int **dvb_dmxdev_init**(struct *dmxdev* **dmxdev*, struct *dvb_adapter* **adap*)
initializes a digital TV demux and registers both demux and DVR devices.

Parameters

struct dmxdev *dmxdev pointer to *struct dmxdev*.

struct dvb_adapter *adap pointer to *struct dvb_adapter*.

void **dvb_dmxdev_release**(struct *dmxdev* **dmxdev*)
releases a digital TV demux and unregisters it.

Parameters

struct dmxdev *dmxdev pointer to *struct dmxdev*.

2.3.3.4 High-level Digital TV demux interface

enum **dvb_dmx_filter_type**
type of demux feed.

Constants

DMX_TYPE_TS feed is in TS mode.

DMX_TYPE_SEC feed is in Section mode.

enum **dvb_dmx_state**
state machine for a demux filter.

Constants

DMX_STATE_FREE indicates that the filter is freed.

DMX_STATE_ALLOCATED indicates that the filter was allocated to be used.

DMX_STATE_READY indicates that the filter is ready to be used.

DMX_STATE_GO indicates that the filter is running.

struct **dvb_demux_filter**
Describes a DVB demux section filter.

Definition

```
struct dvb_demux_filter {
    struct dmx_section_filter filter;
    u8 maskandmode[DMX_MAX_FILTER_SIZE];
    u8 maskandnotmode[DMX_MAX_FILTER_SIZE];
    bool doneq;
    struct dvb_demux_filter *next;
    struct dvb_demux_feed *feed;
    int index;
    enum dvb_dmx_state state;
    enum dvb_dmx_filter_type type;
};
```

Members

filter Section filter as defined by *struct dmx_section_filter*.

maskandmode logical and bit mask.

maskandnotmode logical and not bit mask.

doneq flag that indicates when a filter is ready.

next pointer to the next section filter.

feed *struct dvb_demux_feed* pointer.

index index of the used demux filter.

state state of the filter as described by *enum dvb_dmx_state*.

type type of the filter as described by *enum dvb_dmx_filter_type*.

struct **dvb_demux_feed**
describes a DVB field

Definition

```
struct dvb_demux_feed {
    union {
        struct dmx_ts_feed ts;
        struct dmx_section_feed sec;
    } feed;
    union {
        dmx_ts_cb ts;
        dmx_section_cb sec;
    } cb;
    struct dvb_demux *demux;
    void *priv;
    enum dvb_dmx_filter_type type;
    enum dvb_dmx_state state;
    u16 pid;
    ktime_t timeout;
    struct dvb_demux_filter *filter;
    u32 buffer_flags;
    enum ts_filter_type ts_type;
    enum dmx_ts_pes pes_type;
    int cc;
    bool pusi_seen;
    u16 peslen;
    struct list_head list_head;
    unsigned int index;
};
```

Members

feed a union describing a digital TV feed. Depending on the feed type, it can be either **feed.ts** or **feed.sec**.

feed.ts a *struct dmx_ts_feed* pointer. For TS feed only.

feed.sec a *struct dmx_section_feed* pointer. For section feed only.

cb a union describing digital TV callbacks. Depending on the feed type, it can be either **cb.ts** or **cb.sec**.

cb.ts a *dmx_ts_cb()* callback function pointer. For TS feed only.

cb.sec a *dmx_section_cb()* callback function pointer. For section feed only.

demux pointer to *struct dvb_demux*.

priv private data that can optionally be used by a DVB driver.

type type of the filter, as defined by *enum dvb_dmx_filter_type*.

state state of the filter as defined by *enum dvb_dmx_state*.

pid PID to be filtered.

timeout feed timeout.

filter pointer to *struct dvb_demux_filter*.

buffer_flags Buffer flags used to report discontinuity users via DVB memory mapped API, as defined by *enum dmx_buffer_flags*.

ts_type type of TS, as defined by *enum ts_filter_type*.

pes_type type of PES, as defined by *enum dmx_ts_pes*.

cc MPEG-TS packet continuity counter

pusi_seen if true, indicates that a discontinuity was detected. it is used to prevent feeding of garbage from previous section.

peslen length of the PES (Packet Elementary Stream).

list_head head for the list of digital TV demux feeds.

index a unique index for each feed. Can be used as hardware pid filter index.

struct **dvb_demux**

represents a digital TV demux

Definition

```
struct dvb_demux {
    struct dmx_demux dmx;
    void *priv;
    int filternum;
    int feednum;
    int (*start_feed)(struct dvb_demux_feed *feed);
    int (*stop_feed)(struct dvb_demux_feed *feed);
    int (*write_to_decoder)(struct dvb_demux_feed *feed, const u8 *buf, size_t len);
    u32 (*check_crc32)(struct dvb_demux_feed *feed, const u8 *buf, size_t len);
    void (*memcpy)(struct dvb_demux_feed *feed, u8 *dst, const u8 *src, size_t len);
    int users;
#define MAX_DVB_DEMUX_USERS 10;
    struct dvb_demux_filter *filter;
    struct dvb_demux_feed *feed;
    struct list_head frontend_list;
    struct dvb_demux_feed *pesfilter[DMX_PES_OTHER];
    u16 pids[DMX_PES_OTHER];
#define DMX_MAX_PID 0x2000;
    struct list_head feed_list;
    u8 tsbuf[204];
    int tsbufp;
    struct mutex mutex;
    spinlock_t lock;
    uint8_t *cnt_storage;
    ktime_t speed_last_time;
```

```
uint32_t speed_pkts_cnt;
};
```

Members

dmx embedded *struct dmx_demux* with demux capabilities and callbacks.

priv private data that can optionally be used by a DVB driver.

filternum maximum amount of DVB filters.

feednum maximum amount of DVB feeds.

start_feed callback routine to be called in order to start a DVB feed.

stop_feed callback routine to be called in order to stop a DVB feed.

write_to_decoder callback routine to be called if the feed is TS and it is routed to an A/V decoder, when a new TS packet is received. Used only on av7110-av.c.

check_crc32 callback routine to check CRC. If not initialized, *dvb_demux* will use an internal one.

memcpy callback routine to memcpy received data. If not initialized, *dvb_demux* will default to *memcpy()*.

users counter for the number of demux opened file descriptors. Currently, it is limited to 10 users.

filter pointer to *struct dvb_demux_filter*.

feed pointer to *struct dvb_demux_feed*.

frontend_list *struct list_head* with frontends used by the demux.

pesfilter array of *struct dvb_demux_feed* with the PES types that will be filtered.

pids list of filtered program IDs.

feed_list *struct list_head* with feeds.

tsbuf temporary buffer used internally to store TS packets.

tsbufp temporary buffer index used internally.

mutex pointer to *struct mutex* used to protect feed set logic.

lock pointer to *spinlock_t*, used to protect buffer handling.

cnt_storage buffer used for TS/TEI continuity check.

speed_last_time *mtime_t* used for TS speed check.

speed_pkts_cnt packets count used for TS speed check.

int **dvb_dmx_init**(*struct dvb_demux *demux*)
initialize a digital TV demux struct.

Parameters

*struct dvb_demux *demux* *struct dvb_demux* to be initialized.

Description

Before being able to register a digital TV demux struct, drivers should call this routine. On its typical usage, some fields should be initialized at the driver before calling it.

A typical usecase is:

```
dvb->demux.dmx.capabilities =
    DMX_TS_FILTERING | DMX_SECTION_FILTERING |
    DMX_MEMORY_BASED_FILTERING;
dvb->demux.priv      = dvb;
dvb->demux.filternum = 256;
dvb->demux.feednum   = 256;
dvb->demux.start_feed = driver_start_feed;
dvb->demux.stop_feed  = driver_stop_feed;
ret = dvb_dmx_init(&dvb->demux);
if (ret < 0)
    return ret;
```

void **dvb_dmx_release**(struct *dvb_demux* **demux*)
releases a digital TV demux internal buffers.

Parameters

struct dvb_demux *demux *struct dvb_demux* to be released.

Description

The DVB core internally allocates data at **demux**. This routine releases those data. Please notice that the struct itself is not released, as it can be embedded on other structs.

void **dvb_dmx_swfilter_packets**(struct *dvb_demux* **demux*, const u8 **buf*, size_t *count*)
use dvb software filter for a buffer with multiple MPEG-TS packets with 188 bytes each.

Parameters

struct dvb_demux *demux pointer to *struct dvb_demux*

const u8 *buf buffer with data to be filtered

size_t count number of MPEG-TS packets with size of 188.

Description

The routine will discard a DVB packet that don't start with 0x47.

Use this routine if the DVB demux fills MPEG-TS buffers that are already aligned.

NOTE

The **buf** size should have size equal to `count * 188`.

void **dvb_dmx_swfilter**(struct *dvb_demux* **demux*, const u8 **buf*, size_t *count*)
use dvb software filter for a buffer with multiple MPEG-TS packets with 188 bytes each.

Parameters

struct dvb_demux *demux pointer to *struct dvb_demux*

const u8 *buf buffer with data to be filtered

size_t count number of MPEG-TS packets with size of 188.

Description

If a DVB packet doesn't start with 0x47, it will seek for the first byte that starts with 0x47.

Use this routine if the DVB demux fill buffers that may not start with a packet start mark (0x47).

NOTE

The **buf** size should have size equal to `count * 188`.

void **dvb_dmx_swfilter_204**(struct *dvb_demux* **demux*, const u8 **buf*, size_t *count*)
 use dvb software filter for a buffer with multiple MPEG-TS packets with 204 bytes each.

Parameters

struct dvb_demux *demux pointer to *struct dvb_demux*
const u8 *buf buffer with data to be filtered
size_t count number of MPEG-TS packets with size of 204.

Description

If a DVB packet doesn't start with 0x47, it will seek for the first byte that starts with 0x47.

Use this routine if the DVB demux fill buffers that may not start with a packet start mark (0x47).

NOTE

The **buf** size should have size equal to `count * 204`.

void **dvb_dmx_swfilter_raw**(struct *dvb_demux* **demux*, const u8 **buf*, size_t *count*)
 make the raw data available to userspace without filtering

Parameters

struct dvb_demux *demux pointer to *struct dvb_demux*
const u8 *buf buffer with data
size_t count number of packets to be passed. The actual size of each packet depends on the *dvb_demux->feed->cb.ts* logic.

Description

Use it if the driver needs to deliver the raw payload to userspace without passing through the kernel demux. That is meant to support some delivery systems that aren't based on MPEG-TS.

This function relies on *dvb_demux->feed->cb.ts* to actually handle the buffer.

2.3.3.5 Driver-internal low-level hardware specific driver demux interface

enum **ts_filter_type**
 filter type bitmap for *dmx_ts_feed.set()*

Constants

TS_PACKET Send TS packets (188 bytes) to callback (default).

TS_PAYLOAD_ONLY In case **TS_PACKET** is set, only send the TS payload (<=184 bytes per packet) to callback

TS_DECODER Send stream to built-in decoder (if present).

TS_DEMUX In case **TS_PACKET** is set, send the TS to the demux device, not to the dvr device

struct **dmx_ts_feed**
 Structure that contains a TS feed filter

Definition

```
struct dm_x_ts_feed {
    int is_filtering;
    struct dm_x_demux *parent;
    void *priv;
    int (*set)(struct dm_x_ts_feed *feed, u16 pid, int type, enum dm_x_ts_pes pes_type, ktime_
    ↪ t timeout);
    int (*start_filtering)(struct dm_x_ts_feed *feed);
    int (*stop_filtering)(struct dm_x_ts_feed *feed);
};
```

Members

is_filtering Set to non-zero when filtering in progress

parent pointer to *struct dm_x_demux*

priv pointer to private data of the API client

set sets the TS filter

start_filtering starts TS filtering

stop_filtering stops TS filtering

Description

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.

struct **dm_x_section_filter**

Structure that describes a section filter

Definition

```
struct dm_x_section_filter {
    u8 filter_value[DMX_MAX_FILTER_SIZE];
    u8 filter_mask[DMX_MAX_FILTER_SIZE];
    u8 filter_mode[DMX_MAX_FILTER_SIZE];
    struct dm_x_section_feed *parent;
    void *priv;
};
```

Members

filter_value Contains up to 16 bytes (128 bits) of the TS section header that will be matched by the section filter

filter_mask Contains a 16 bytes (128 bits) filter mask with the bits specified by **filter_value** that will be used on the filter match logic.

filter_mode Contains a 16 bytes (128 bits) filter mode.

parent Back-pointer to *struct dm_x_section_feed*.

priv Pointer to private data of the API client.

Description

The **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.

struct **dmx_section_feed**

Structure that contains a section feed filter

Definition

```
struct dmx_section_feed {
    int is_filtering;
    struct dmx_demux *parent;
    void *priv;
    int check_crc;
    int (*set)(struct dmx_section_feed *feed, u16 pid, int check_crc);
    int (*allocate_filter)(struct dmx_section_feed *feed, struct dmx_section_filter_
↳**filter);
    int (*release_filter)(struct dmx_section_feed *feed, struct dmx_section_filter_
↳*filter);
    int (*start_filtering)(struct dmx_section_feed *feed);
    int (*stop_filtering)(struct dmx_section_feed *feed);
};
```

Members

is_filtering Set to non-zero when filtering in progress

parent pointer to *struct dmx_demux*

priv pointer to private data of the API client

check_crc If non-zero, check the CRC values of filtered sections.

set sets the section filter

allocate_filter 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.

release_filter 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.

start_filtering starts section filtering

stop_filtering stops section filtering

Description

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.

dmx_ts_cb

Typedef: DVB demux TS filter callback function prototype

Syntax

```
int dmx_ts_cb (const u8 *buffer1, size_t buffer1_length, const u8
*buffer2, size_t buffer2_length, struct dmx_ts_feed *source, u32
*buffer_flags)
```

Parameters

const u8 *buffer1 Pointer to the start of the filtered TS packets.

size_t buffer1_length Length of the TS data in buffer1.

const u8 *buffer2 Pointer to the tail of the filtered TS packets, or NULL.

size_t buffer2_length Length of the TS data in buffer2.

struct dm_x_ts_feed *source Indicates which TS feed is the source of the callback.

u32 *buffer_flags Address where buffer flags are stored. Those are used to report discontinuity users via DVB memory mapped API, as defined by [enum dm_x_buffer_flags](#).

Description

This function callback prototype, provided by the client of the demux API, is called from the demux code. The function is only called when filtering on a TS feed has been enabled using the [start_filtering\(\)](#) function at the [dm_x_demux](#). 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. 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.

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. 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.

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 and return `-EOVERFLOW`.

The type of data returned to the callback can be selected by the [dm_x_ts_feed.**set**](#) function. 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.

- 0, on success;
- `-EOVERFLOW`, on buffer overflow.

Return

dm_x_section_cb

Typedef: DVB demux TS filter callback function prototype

Syntax


```
int dmux_section_cb (const u8 *buffer1, size_t buffer1_len, const u8
 *buffer2, size_t buffer2_len, struct dmux_section_filter *source, u32
 *buffer_flags)
```

Parameters

const u8 *buffer1 Pointer to the start of the filtered section, e.g. within the circular buffer of the demux driver.

size_t buffer1_len Length of the filtered section data in **buffer1**, including headers and CRC.

const u8 *buffer2 Pointer to the tail of the filtered section data, or NULL. Useful to handle the wrapping of a circular buffer.

size_t buffer2_len Length of the filtered section data in **buffer2**, including headers and CRC.

struct dmux_section_filter *source Indicates which section feed is the source of the callback.

u32 *buffer_flags Address where buffer flags are stored. Those are used to report discontinuity users via DVB memory mapped API, as defined by *enum dmux_buffer_flags*.

Description

This function callback prototype, 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 *dmux_ts_feed.**start_filtering***. 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 *dmux_ts_feed.**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.

enum dmux_frontend_source

Used to identify the type of frontend

Constants

DMX_MEMORY_FE The source of the demux is memory. It means that the MPEG-TS to be filtered comes from userspace, via write() syscall.

DMX_FRONTEND_0 The source of the demux is a frontend connected to the demux.

struct dmux_frontend

Structure that lists the frontends associated with a demux

Definition

```
struct dmux_frontend {
    struct list_head connectivity_list;
```

```
enum dmx_frontend_source source;
};
```

Members

connectivity_list List of front-ends that can be connected to a particular demux;

source Type of the frontend.

Description

FIXME: this structure should likely be replaced soon by some media-controller based logic.

enum **dmx_demux_caps**
MPEG-2 TS Demux capabilities bitmap

Constants

DMX_TS_FILTERING set if TS filtering is supported;

DMX_SECTION_FILTERING set if section filtering is supported;

DMX_MEMORY_BASED_FILTERING set if write() available.

Description

Those flags are OR'ed in the *dmx_demux.capabilities* field

DMX_FE_ENTRY(*list*)

Casts elements in the list of registered front-ends from the generic type struct *list_head* to the type **struct dmx_frontend*

Parameters

list list of *struct dmx_frontend*

struct **dmx_demux**

Structure that contains the demux capabilities and callbacks.

Definition

```
struct dmx_demux {
    enum dmx_demux_caps capabilities;
    struct dmx_frontend *frontend;
    void *priv;
    int (*open)(struct dmx_demux *demux);
    int (*close)(struct dmx_demux *demux);
    int (*write)(struct dmx_demux *demux, const char __user *buf, size_t count);
    int (*allocate_ts_feed)(struct dmx_demux *demux, struct dmx_ts_feed **feed, dmx_ts_cb_
↪callback);
    int (*release_ts_feed)(struct dmx_demux *demux, struct dmx_ts_feed *feed);
    int (*allocate_section_feed)(struct dmx_demux *demux, struct dmx_section_feed **feed,
↪dmx_section_cb callback);
    int (*release_section_feed)(struct dmx_demux *demux, struct dmx_section_feed *feed);
    int (*add_frontend)(struct dmx_demux *demux, struct dmx_frontend *frontend);
    int (*remove_frontend)(struct dmx_demux *demux, struct dmx_frontend *frontend);
    struct list_head *(*get_frontends)(struct dmx_demux *demux);
    int (*connect_frontend)(struct dmx_demux *demux, struct dmx_frontend *frontend);
    int (*disconnect_frontend)(struct dmx_demux *demux);
```

```
int (*get_pes_pids)(struct dmx_demux *demux, u16 *pids);
};
```

Members

capabilities Bitfield of capability flags.

frontend Front-end connected to the demux

priv Pointer to private data of the API client

open 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. The **demux** function parameter contains a pointer to the demux API and instance data. It returns: 0 on success; -EUSERS, if maximum usage count was reached; -EINVAL, on bad parameter.

close 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. The **demux** function parameter contains a pointer to the demux API and instance data. It returns: 0 on success; -ENODEV, if demux was not in use (e. g. no users); -EINVAL, on bad parameter.

write 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. The **demux** function parameter contains a pointer to the demux API and instance data. The **buf** function parameter contains a pointer to the TS data in kernel-space memory. The **count** function parameter contains the length of the TS data. It returns: 0 on success; -ERESTARTSYS, if mutex lock was interrupted; -EINTR, if a signal handling is pending; -ENODEV, if demux was removed; -EINVAL, on bad parameter.

allocate_ts_feed 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. The **demux** function parameter contains a pointer to the demux API and instance data. The **feed** function parameter contains a pointer to the TS feed API and instance data. The **callback** function parameter contains a pointer to the callback function for passing received TS packet. It returns: 0 on success; -ERESTARTSYS, if mutex lock was interrupted; -EBUSY, if no more TS feeds is available; -EINVAL, on bad parameter.

release_ts_feed Releases the resources allocated with **allocate_ts_feed**. Any filtering in progress on the TS feed should be stopped before calling this function. The **demux** function parameter contains a pointer to the demux API and instance data. The **feed** function parameter contains a pointer to the TS feed API and instance data. It returns: 0 on success; -EINVAL on bad parameter.

allocate_section_feed 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. The **demux** function parameter contains a pointer to the demux API and instance data. The **feed** function parameter contains a pointer to the TS feed API and instance data. The **callback** function parameter contains a pointer to the callback function for passing received TS packet. It returns: 0 on success; -EBUSY, if no more TS feeds is available; -EINVAL, on bad parameter.

release_section_feed 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. The **demux** function parameter contains a pointer to the demux API and instance data. The **feed** function parameter contains a pointer to the TS feed API and instance data. It returns: 0 on success; -EINVAL, on bad parameter.

add_frontend 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**. The **demux** function parameter contains a pointer to the demux API and instance data. The **frontend** function parameter contains a pointer to the front-end instance data. It returns: 0 on success; -EINVAL, on bad parameter.

remove_frontend 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. The **demux** function parameter contains a pointer to the demux API and instance data. The **frontend** function parameter contains a pointer to the front-end instance data. It returns: 0 on success; -ENODEV, if the front-end was not found, -EINVAL, on bad parameter.

get_frontends 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**. 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 `struct dmx_frontend`. *The caller must not free the memory of any of the elements obtained via this function call. The ****demux*** function parameter contains a pointer to the demux API and instance data. It returns a `struct list_head` pointer to the list of front-end interfaces, or NULL in the case of an empty list.*

connect_frontend 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**. 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**. The **demux** function parameter contains a pointer to the demux API and instance data. The **frontend** function parameter contains a pointer to the front-end instance data. It returns: 0 on success; -EINVAL, on bad parameter.

disconnect_frontend Disconnects the demux and a front-end previously connected by a **con-**

nect_frontend call. The **demux** function parameter contains a pointer to the demux API and instance data. It returns: 0 on success; -EINVAL on bad parameter.

get_pes_pids Get the PIDs for DMX_PES_AUDIO0, DMX_PES_VIDEO0, DMX_PES_TELETEXT0, DMX_PES_SUBTITLE0 and DMX_PES_PCR0. The **demux** function parameter contains a pointer to the demux API and instance data. The **pids** function parameter contains an array with five u16 elements where the PIDs will be stored. It returns: 0 on success; -EINVAL on bad parameter.

2.3.4 Digital TV Conditional Access kABI

struct **dvb_ca_en50221**

Structure describing a CA interface

Definition

```
struct dvb_ca_en50221 {
    struct module *owner;
    int (*read_attribute_mem)(struct dvb_ca_en50221 *ca, int slot, int address);
    int (*write_attribute_mem)(struct dvb_ca_en50221 *ca, int slot, int address, u8
↪value);
    int (*read_cam_control)(struct dvb_ca_en50221 *ca, int slot, u8 address);
    int (*write_cam_control)(struct dvb_ca_en50221 *ca, int slot, u8 address, u8 value);
    int (*read_data)(struct dvb_ca_en50221 *ca, int slot, u8 *ebuf, int ecount);
    int (*write_data)(struct dvb_ca_en50221 *ca, int slot, u8 *ebuf, int ecount);
    int (*slot_reset)(struct dvb_ca_en50221 *ca, int slot);
    int (*slot_shutdown)(struct dvb_ca_en50221 *ca, int slot);
    int (*slot_ts_enable)(struct dvb_ca_en50221 *ca, int slot);
    int (*poll_slot_status)(struct dvb_ca_en50221 *ca, int slot, int open);
    void *data;
    void *private;
};
```

Members

owner the module owning this structure

read_attribute_mem function for reading attribute memory on the CAM

write_attribute_mem function for writing attribute memory on the CAM

read_cam_control function for reading the control interface on the CAM

write_cam_control function for reading the control interface on the CAM

read_data function for reading data (block mode)

write_data function for writing data (block mode)

slot_reset function to reset the CAM slot

slot_shutdown function to shutdown a CAM slot

slot_ts_enable function to enable the Transport Stream on a CAM slot

poll_slot_status function to poll slot status. Only necessary if
DVB_CA_FLAG_EN50221_IRQ_CAMCHANGE is not set.

data private data, used by caller.

private Opaque data used by the `dvb_ca` core. Do not modify!

NOTE

the `read_*`, `write_*` and `poll_slot_status` functions will be called for different slots concurrently and need to use locks where and if appropriate. There will be no concurrent access to one slot.

```
void dvb_ca_en50221_camchange_irq(struct dvb_ca_en50221 *pubca,      int slot,  
                                int change_type)
```

A CAMCHANGE IRQ has occurred.

Parameters

struct dvb_ca_en50221 *pubca CA instance.

int slot Slot concerned.

int change_type One of the `DVB_CA_CAMCHANGE_*` values

```
void dvb_ca_en50221_camready_irq(struct dvb_ca_en50221 *pubca, int slot)  
    A CAMREADY IRQ has occurred.
```

Parameters

struct dvb_ca_en50221 *pubca CA instance.

int slot Slot concerned.

```
void dvb_ca_en50221_frda_irq(struct dvb_ca_en50221 *ca, int slot)  
    An FR or a DA IRQ has occurred.
```

Parameters

struct dvb_ca_en50221 *ca CA instance.

int slot Slot concerned.

```
int dvb_ca_en50221_init(struct dvb_adapter *dvb_adapter, struct dvb_ca_en50221 *ca,  
                       int flags, int slot_count)  
    Initialise a new DVB CA device.
```

Parameters

struct dvb_adapter *dvb_adapter DVB adapter to attach the new CA device to.

struct dvb_ca_en50221 *ca The `dvb_ca` instance.

int flags Flags describing the CA device (`DVB_CA_EN50221_FLAG_*`).

int slot_count Number of slots supported.

Description

return 0 on success, nonzero on failure

```
void dvb_ca_en50221_release(struct dvb_ca_en50221 *ca)  
    Release a DVB CA device.
```

Parameters

struct dvb_ca_en50221 *ca The associated `dvb_ca` instance.

2.3.5 Digital TV Network kABI

struct **dvb_net**
describes a DVB network interface

Definition

```
struct dvb_net {
    struct dvb_device *dvbdev;
    struct net_device *device[DVB_NET_DEVICES_MAX];
    int state[DVB_NET_DEVICES_MAX];
    unsigned int exit:1;
    struct dmxdemux *demux;
    struct mutex ioctl_mutex;
};
```

Members

dvbdev pointer to *struct dvb_device*.

device array of pointers to struct *net_device*.

state array of integers to each net device. A value different than zero means that the interface is in usage.

exit flag to indicate when the device is being removed.

demux pointer to *struct dmxdemux*.

ioctl_mutex protect access to this struct.

Description

Currently, the core supports up to DVB_NET_DEVICES_MAX (10) network devices.

int **dvb_net_init**(struct *dvb_adapter* *adap, struct *dvb_net* *dvbnet, struct *dmxdemux* *dmxdemux)
initializes a digital TV network device and registers it.

Parameters

struct *dvb_adapter* *adap pointer to *struct dvb_adapter*.

struct *dvb_net* *dvbnet pointer to *struct dvb_net*.

struct *dmxdemux* *dmxdemux pointer to *struct dmxdemux*.

void **dvb_net_release**(struct *dvb_net* *dvbnet)
releases a digital TV network device and unregisters it.

Parameters

struct *dvb_net* *dvbnet pointer to *struct dvb_net*.

2.4 Remote Controller devices

2.4.1 Remote Controller core

The remote controller core implements infrastructure to receive and send remote controller keyboard keystrokes and mouse events.

Every time a key is pressed on a remote controller, a scan code is produced. Also, on most hardware, keeping a key pressed for more than a few dozens of milliseconds produce a repeat key event. That's somewhat similar to what a normal keyboard or mouse is handled internally on Linux¹. So, the remote controller core is implemented on the top of the linux input/evdev interface.

However, most of the remote controllers use infrared (IR) to transmit signals. As there are several protocols used to modulate infrared signals, one important part of the core is dedicated to adjust the driver and the core system to support the infrared protocol used by the emitter.

The infrared transmission is done by blinking a infrared emitter using a carrier. The carrier can be switched on or off by the IR transmitter hardware. When the carrier is switched on, it is called *PULSE*. When the carrier is switched off, it is called *SPACE*.

In other words, a typical IR transmission can be viewed as a sequence of *PULSE* and *SPACE* events, each with a given duration.

The carrier parameters (frequency, duty cycle) and the intervals for *PULSE* and *SPACE* events depend on the protocol. For example, the NEC protocol uses a carrier of 38kHz, and transmissions start with a 9ms *PULSE* and a 4.5ms *SPACE*. It then transmits 16 bits of scan code, being 8 bits for address (usually it is a fixed number for a given remote controller), followed by 8 bits of code. A bit "1" is modulated with 560µs *PULSE* followed by 1690µs *SPACE* and a bit "0" is modulated with 560µs *PULSE* followed by 560µs *SPACE*.

At receiver, a simple low-pass filter can be used to convert the received signal in a sequence of *PULSE/SPACE* events, filtering out the carrier frequency. Due to that, the receiver doesn't care about the carrier's actual frequency parameters: all it has to do is to measure the amount of time it receives *PULSE/SPACE* events. So, a simple IR receiver hardware will just provide a sequence of timings for those events to the Kernel. The drivers for hardware with such kind of receivers are identified by `RC_DRIVER_IR_RAW`, as defined by *rc_driver_type*². Other hardware come with a microcontroller that decode the *PULSE/SPACE* sequence and return scan codes to the Kernel. Such kind of receivers are identified by `RC_DRIVER_SCANCODE`.

When the RC core receives events produced by `RC_DRIVER_IR_RAW` IR receivers, it needs to decode the IR protocol, in order to obtain the corresponding scan code. The protocols supported by the RC core are defined at enum `rc_proto`.

When the RC code receives a scan code (either directly, by a driver of the type `RC_DRIVER_SCANCODE`, or via its IR decoders), it needs to convert into a Linux input event code. This is done via a mapping table.

The Kernel has support for mapping tables available on most media devices. It also supports loading a table in runtime, via some sysfs nodes. See the *RC userspace API* for more details.

¹ The main difference is that, on keyboard events, the keyboard controller produces one event for a key press and another one for key release. On infrared-based remote controllers, there's no key release event. Instead, an extra code is produced to indicate key repeats.

² The RC core also supports devices that have just IR emitters, without any receivers. Right now, all such devices work only in raw TX mode. Such kind of hardware is identified as `RC_DRIVER_IR_RAW_TX`.

2.4.1.1 Remote controller data structures and functions

enum **rc_driver_type**
type of the RC driver.

Constants

RC_DRIVER_SCANCODE Driver or hardware generates a scancode.

RC_DRIVER_IR_RAW Driver or hardware generates pulse/space sequences. It needs a Infra-Red pulse/space decoder

RC_DRIVER_IR_RAW_TX Device transmitter only, driver requires pulse/space data sequence.

struct **rc_scancode_filter**
Filter scan codes.

Definition

```
struct rc_scancode_filter {
    u32 data;
    u32 mask;
};
```

Members

data Scancode data to match.

mask Mask of bits of scancode to compare.

enum **rc_filter_type**
Filter type constants.

Constants

RC_FILTER_NORMAL Filter for normal operation.

RC_FILTER_WAKEUP Filter for waking from suspend.

RC_FILTER_MAX Number of filter types.

struct **lirc_fh**
represents an open lirc file

Definition

```
struct lirc_fh {
    struct list_head list;
    struct rc_dev *rc;
    int carrier_low;
    bool send_timeout_reports;
    unsigned int *rawir;
    struct lirc_scancode *scancodes;
    wait_queue_head_t wait_poll;
    u8 send_mode;
    u8 rec_mode;
};
```

Members

list list of open file handles

rc rcdev for this lirc chardev

carrier_low when setting the carrier range, first the low end must be set with an ioctl and then the high end with another ioctl

send_timeout_reports report timeouts in lirc raw IR.

rawir queue for incoming raw IR

scancodes queue for incoming decoded scancodes

wait_poll poll struct for lirc device

send_mode lirc mode for sending, either LIRC_MODE_SCANCODE or LIRC_MODE_PULSE

rec_mode lirc mode for receiving, either LIRC_MODE_SCANCODE or LIRC_MODE_MODE2

struct **rc_dev**

represents a remote control device

Definition

```
struct rc_dev {
    struct device                dev;
    bool managed_alloc;
    const struct attribute_group *sysfs_groups[5];
    const char                   *device_name;
    const char                   *input_phys;
    struct input_id              input_id;
    const char                   *driver_name;
    const char                   *map_name;
    struct rc_map                rc_map;
    struct mutex                 lock;
    unsigned int                 minor;
    struct ir_raw_event_ctrl     *raw;
    struct input_dev             *input_dev;
    enum rc_driver_type          driver_type;
    bool idle;
    bool encode_wakeup;
    u64 allowed_protocols;
    u64 enabled_protocols;
    u64 allowed_wakeup_protocols;
    enum rc_proto                wakeup_protocol;
    struct rc_scancode_filter    scancode_filter;
    struct rc_scancode_filter    scancode_wakeup_filter;
    u32 scancode_mask;
    u32 users;
    void *priv;
    spinlock_t keylock;
    bool keypressed;
    unsigned long                keyup_jiffies;
    struct timer_list            timer_keyup;
    struct timer_list            timer_repeat;
    u32 last_keycode;
    enum rc_proto                last_protocol;
    u64 last_scancode;
    u8 last_toggle;
    u32 timeout;
    u32 min_timeout;
    u32 max_timeout;
};
```

```

    u32 rx_resolution;
    u32 tx_resolution;
#ifdef CONFIG_LIRC;
    struct device          lirc_dev;
    struct cdev           lirc_cdev;
    ktime_t gap_start;
    u64 gap_duration;
    bool gap;
    spinlock_t lirc_fh_lock;
    struct list_head      lirc_fh;
#endif;
    bool registered;
    int (*change_protocol)(struct rc_dev *dev, u64 *rc_proto);
    int (*open)(struct rc_dev *dev);
    void (*close)(struct rc_dev *dev);
    int (*s_tx_mask)(struct rc_dev *dev, u32 mask);
    int (*s_tx_carrier)(struct rc_dev *dev, u32 carrier);
    int (*s_tx_duty_cycle)(struct rc_dev *dev, u32 duty_cycle);
    int (*s_rx_carrier_range)(struct rc_dev *dev, u32 min, u32 max);
    int (*tx_ir)(struct rc_dev *dev, unsigned *txbuf, unsigned n);
    void (*s_idle)(struct rc_dev *dev, bool enable);
    int (*s_wideband_receiver)(struct rc_dev *dev, int enable);
    int (*s_carrier_report)(struct rc_dev *dev, int enable);
    int (*s_filter)(struct rc_dev *dev, struct rc_scancode_filter *filter);
    int (*s_wakeup_filter)(struct rc_dev *dev, struct rc_scancode_filter *filter);
    int (*s_timeout)(struct rc_dev *dev, unsigned int timeout);
};

```

Members

dev driver model's view of this device

managed_alloc devm_rc_allocate_device was used to create rc_dev

sysfs_groups sysfs attribute groups

device_name name of the rc child device

input_phys physical path to the input child device

input_id id of the input child device (struct input_id)

driver_name name of the hardware driver which registered this device

map_name name of the default keymap

rc_map current scan/key table

lock used to ensure we've filled in all protocol details before anyone can call show_protocols or store_protocols

minor unique minor remote control device number

raw additional data for raw pulse/space devices

input_dev the input child device used to communicate events to userspace

driver_type specifies if protocol decoding is done in hardware or software

idle used to keep track of RX state

encode_wakeup wakeup filtering uses IR encode API, therefore the allowed wakeup protocols is the set of all raw encoders

allowed_protocols bitmask with the supported RC_PROTO_BIT_* protocols

enabled_protocols bitmask with the enabled RC_PROTO_BIT_* protocols

allowed_wakeup_protocols bitmask with the supported RC_PROTO_BIT_* wakeup protocols

wakeup_protocol the enabled RC_PROTO_* wakeup protocol or RC_PROTO_UNKNOWN if disabled.

scancode_filter scancode filter

scancode_wakeup_filter scancode wakeup filters

scancode_mask some hardware decoders are not capable of providing the full scancode to the application. As this is a hardware limit, we can't do anything with it. Yet, as the same keycode table can be used with other devices, a mask is provided to allow its usage. Drivers should generally leave this field in blank

users number of current users of the device

priv driver-specific data

keylock protects the remaining members of the struct

keypressed whether a key is currently pressed

keyup_jiffies time (in jiffies) when the current keypress should be released

timer_keyup timer for releasing a keypress

timer_repeat timer for autorepeat events. This is needed for CEC, which has non-standard repeats.

last_keycode keycode of last keypress

last_protocol protocol of last keypress

last_scancode scancode of last keypress

last_toggle toggle value of last command

timeout optional time after which device stops sending data

min_timeout minimum timeout supported by device

max_timeout maximum timeout supported by device

rx_resolution resolution (in us) of input sampler

tx_resolution resolution (in us) of output sampler

lirc_dev lirc device

lirc_cdev lirc char cdev

gap_start time when gap starts

gap_duration duration of initial gap

gap true if we're in a gap

lirc_fh_lock protects lirc_fh list

lirc_fh list of open files

registered set to true by *rc_register_device()*, false by *rc_unregister_device*

change_protocol allow changing the protocol used on hardware decoders

open callback to allow drivers to enable polling/irq when IR input device is opened.

close callback to allow drivers to disable polling/irq when IR input device is opened.

s_tx_mask set transmitter mask (for devices with multiple tx outputs)

s_tx_carrier set transmit carrier frequency

s_tx_duty_cycle set transmit duty cycle (0% - 100%)

s_rx_carrier_range inform driver about carrier it is expected to handle

tx_ir transmit IR

s_idle enable/disable hardware idle mode, upon which, device doesn't interrupt host until it sees IR pulses

s_wideband_receiver enable wide band receiver used for learning

s_carrier_report enable carrier reports

s_filter set the scancode filter

s_wakeup_filter set the wakeup scancode filter. If the mask is zero then wakeup should be disabled. *wakeup_protocol* will be set to a valid protocol if mask is nonzero.

s_timeout set hardware timeout in us

struct *rc_dev* * **rc_allocate_device**(enum *rc_driver_type*)
Allocates a RC device

Parameters

enum **rc_driver_type** specifies the type of the RC output to be allocated returns a pointer to *struct rc_dev*.

struct *rc_dev* * **devm_rc_allocate_device**(struct device **dev*, enum *rc_driver_type*)
Managed RC device allocation

Parameters

struct device ***dev** pointer to struct device

enum **rc_driver_type** specifies the type of the RC output to be allocated returns a pointer to *struct rc_dev*.

void **rc_free_device**(struct *rc_dev* **dev*)
Frees a RC device

Parameters

struct *rc_dev* ***dev** pointer to *struct rc_dev*.

int **rc_register_device**(struct *rc_dev* **dev*)
Registers a RC device

Parameters

struct *rc_dev* ***dev** pointer to *struct rc_dev*.

int **devm_rc_register_device**(struct device *parent, struct *rc_dev* *dev)
Managed registering of a RC device

Parameters

struct device *parent pointer to struct device.

struct rc_dev *dev pointer to *struct rc_dev*.

void **rc_unregister_device**(struct *rc_dev* *dev)
Unregisters a RC device

Parameters

struct rc_dev *dev pointer to *struct rc_dev*.

struct rc_map_table
represents a scancode/keycode pair

Definition

```
struct rc_map_table {
    u64 scancode;
    u32 keycode;
};
```

Members

scancode scan code (u64)

keycode Linux input keycode

struct rc_map
represents a keycode map table

Definition

```
struct rc_map {
    struct rc_map_table *scan;
    unsigned int size;
    unsigned int len;
    unsigned int alloc;
    enum rc_proto rc_proto;
    const char *name;
    spinlock_t lock;
};
```

Members

scan pointer to struct *rc_map_table*

size Max number of entries

len Number of entries that are in use

alloc size of *scan, in bytes

rc_proto type of the remote controller protocol, as defined at enum *rc_proto*

name name of the key map table

lock lock to protect access to this structure

struct **rc_map_list**
list of the registered *rc_map* maps

Definition

```
struct rc_map_list {
    struct list_head    list;
    struct rc_map map;
};
```

Members

list pointer to struct `list_head`

map pointer to struct *rc_map*

int **rc_map_register**(struct *rc_map_list* *map)
Registers a Remote Controller scancode map

Parameters

struct *rc_map_list* *map pointer to *struct rc_map_list*

void **rc_map_unregister**(struct *rc_map_list* *map)
Unregisters a Remote Controller scancode map

Parameters

struct *rc_map_list* *map pointer to *struct rc_map_list*

struct *rc_map* * **rc_map_get**(const char *name)
gets an RC map from its name

Parameters

const char *name name of the RC scancode map

2.5 Media Controller devices

2.5.1 Media Controller

The media controller userspace API is documented in *the Media Controller uAPI book*. This document focus on the kernel-side implementation of the media framework.

2.5.1.1 Abstract media device model

Discovering a device internal topology, and configuring it at runtime, is one of the goals of the media framework. To achieve this, hardware devices are modelled as an oriented graph of building blocks called entities connected through pads.

An entity is a basic media hardware building block. It can correspond to a large variety of logical blocks such as physical hardware devices (CMOS sensor for instance), logical hardware devices (a building block in a System-on-Chip image processing pipeline), DMA channels or physical connectors.

A pad is a connection endpoint through which an entity can interact with other entities. Data (not restricted to video) produced by an entity flows from the entity's output to one or more entity inputs. Pads should not be confused with physical pins at chip boundaries.

A link is a point-to-point oriented connection between two pads, either on the same entity or on different entities. Data flows from a source pad to a sink pad.

2.5.1.2 Media device

A media device is represented by a *struct media_device* instance, defined in `include/media/media-device.h`. Allocation of the structure is handled by the media device driver, usually by embedding the *media_device* instance in a larger driver-specific structure.

Drivers register media device instances by calling `__media_device_register()` via the macro `media_device_register()` and unregistered by calling `media_device_unregister()`.

2.5.1.3 Entities

Entities are represented by a *struct media_entity* instance, defined in `include/media/media-entity.h`. The structure is usually embedded into a higher-level structure, such as *v4l2_subdev* or *video_device* instances, although drivers can allocate entities directly.

Drivers initialize entity pads by calling `media_entity_pads_init()`.

Drivers register entities with a media device by calling `media_device_register_entity()` and unregistered by calling `media_device_unregister_entity()`.

2.5.1.4 Interfaces

Interfaces are represented by a *struct media_interface* instance, defined in `include/media/media-entity.h`. Currently, only one type of interface is defined: a device node. Such interfaces are represented by a *struct media_intf_devnode*.

Drivers initialize and create device node interfaces by calling `media_devnode_create()` and remove them by calling: `media_devnode_remove()`.

2.5.1.5 Pads

Pads are represented by a *struct media_pad* instance, defined in `include/media/media-entity.h`. Each entity stores its pads in a pads array managed by the entity driver. Drivers usually embed the array in a driver-specific structure.

Pads are identified by their entity and their 0-based index in the pads array.

Both information are stored in the *struct media_pad*, making the *struct media_pad* pointer the canonical way to store and pass link references.

Pads have flags that describe the pad capabilities and state.

`MEDIA_PAD_FL_SINK` indicates that the pad supports sinking data. `MEDIA_PAD_FL_SOURCE` indicates that the pad supports sourcing data.

Note: One and only one of MEDIA_PAD_FL_SINK or MEDIA_PAD_FL_SOURCE must be set for each pad.

2.5.1.6 Links

Links are represented by a `struct media_link` instance, defined in `include/media/media-entity.h`. There are two types of links:

1. pad to pad links:

Associate two entities via their PADS. Each entity has a list that points to all links originating at or targeting any of its pads. A given link is thus stored twice, once in the source entity and once in the target entity.

Drivers create pad to pad links by calling: `media_create_pad_link()` and remove with `media_entity_remove_links()`.

2. interface to entity links:

Associate one interface to a Link.

Drivers create interface to entity links by calling: `media_create_intf_link()` and remove with `media_remove_intf_links()`.

Note: Links can only be created after having both ends already created.

Links have flags that describe the link capabilities and state. The valid values are described at `media_create_pad_link()` and `media_create_intf_link()`.

2.5.1.7 Graph traversal

The media framework provides APIs to iterate over entities in a graph.

To iterate over all entities belonging to a media device, drivers can use the `media_device_for_each_entity` macro, defined in `include/media/media-device.h`.

```
struct media_entity *entity;

media_device_for_each_entity(entity, mdev) {
// entity will point to each entity in turn
...
}
```

Drivers might also need to iterate over all entities in a graph that can be reached only through enabled links starting at a given entity. The media framework provides a depth-first graph traversal API for that purpose.

Note: Graphs with cycles (whether directed or undirected) are **NOT** supported by the graph traversal API. To prevent infinite loops, the graph traversal code limits the maximum depth to MEDIA_ENTITY_ENUM_MAX_DEPTH, currently defined as 16.

Drivers initiate a graph traversal by calling `media_graph_walk_start()`

The graph structure, provided by the caller, is initialized to start graph traversal at the given entity.

Drivers can then retrieve the next entity by calling `media_graph_walk_next()`

When the graph traversal is complete the function will return NULL.

Graph traversal can be interrupted at any moment. No cleanup function call is required and the graph structure can be freed normally.

Helper functions can be used to find a link between two given pads, or a pad connected to another pad through an enabled link `media_entity_find_link()` and `media_entity_remote_pad()`.

2.5.1.8 Use count and power handling

Due to the wide differences between drivers regarding power management needs, the media controller does not implement power management. However, the `struct media_entity` includes a `use_count` field that media drivers can use to track the number of users of every entity for power management needs.

The `media_entity.use_count` field is owned by media drivers and must not be touched by entity drivers. Access to the field must be protected by the `media_device.graph_mutex` lock.

2.5.1.9 Links setup

Link properties can be modified at runtime by calling `media_entity_setup_link()`.

2.5.1.10 Pipelines and media streams

When starting streaming, drivers must notify all entities in the pipeline to prevent link states from being modified during streaming by calling `media_pipeline_start()`.

The function will mark all entities connected to the given entity through enabled links, either directly or indirectly, as streaming.

The `struct media_pipeline` instance pointed to by the `pipe` argument will be stored in every entity in the pipeline. Drivers should embed the `struct media_pipeline` in higher-level pipeline structures and can then access the pipeline through the `struct media_entity` `pipe` field.

Calls to `media_pipeline_start()` can be nested. The pipeline pointer must be identical for all nested calls to the function.

`media_pipeline_start()` may return an error. In that case, it will clean up any of the changes it did by itself.

When stopping the stream, drivers must notify the entities with `media_pipeline_stop()`.

If multiple calls to `media_pipeline_start()` have been made the same number of `media_pipeline_stop()` calls are required to stop streaming. The `media_entity.pipe` field is reset to NULL on the last nested stop call.

Link configuration will fail with `-EBUSY` by default if either end of the link is a streaming entity. Links that can be modified while streaming must be marked with the `MEDIA_LNK_FL_DYNAMIC` flag.

If other operations need to be disallowed on streaming entities (such as changing entities configuration parameters) drivers can explicitly check the `media_entity.stream_count` field to find out if an entity is streaming. This operation must be done with the `media_device` `graph_mutex` held.

2.5.1.11 Link validation

Link validation is performed by `media_pipeline_start()` for any entity which has sink pads in the pipeline. The `media_entity.link_validate()` callback is used for that purpose. In `link_validate()` callback, entity driver should check that the properties of the source pad of the connected entity and its own sink pad match. It is up to the type of the entity (and in the end, the properties of the hardware) what matching actually means.

Subsystems should facilitate link validation by providing subsystem specific helper functions to provide easy access for commonly needed information, and in the end provide a way to use driver-specific callbacks.

2.5.1.12 Media Controller Device Allocator API

When the media device belongs to more than one driver, the shared media device is allocated with the shared struct device as the key for look ups.

The shared media device should stay in registered state until the last driver unregisters it. In addition, the media device should be released when all the references are released. Each driver gets a reference to the media device during probe, when it allocates the media device. If media device is already allocated, the allocate API bumps up the refcount and returns the existing media device. The driver puts the reference back in its disconnect routine when it calls `media_device_delete()`.

The media device is unregistered and cleaned up from the `kref` put handler to ensure that the media device stays in registered state until the last driver unregisters the media device.

Driver Usage

Drivers should use the appropriate media-core routines to manage the shared media device life-time handling the two states: 1. allocate -> register -> delete 2. get reference to already registered device -> delete

call `media_device_delete()` routine to make sure the shared media device delete is handled correctly.

driver probe: Call `media_device_usb_allocate()` to allocate or get a reference Call `media_device_register()`, if media devnode isn't registered

driver disconnect: Call `media_device_delete()` to free the `media_device`. Freeing is handled by the `kref` put handler.

2.5.1.13 API Definitions

struct **media_entity_notify**

Media Entity Notify

Definition

```
struct media_entity_notify {
    struct list_head list;
    void *notify_data;
    void (*notify)(struct media_entity *entity, void *notify_data);
};
```

Members

list List head

notify_data Input data to invoke the callback

notify Callback function pointer

Description

Drivers may register a callback to take action when new entities get registered with the media device. This handler is intended for creating links between existing entities and should not create entities and register them.

struct **media_device_ops**

Media device operations

Definition

```
struct media_device_ops {
    int (*link_notify)(struct media_link *link, u32 flags, unsigned int notification);
    struct media_request *(*req_alloc)(struct media_device *mdev);
    void (*req_free)(struct media_request *req);
    int (*req_validate)(struct media_request *req);
    void (*req_queue)(struct media_request *req);
};
```

Members

link_notify Link state change notification callback. This callback is called with the `graph_mutex` held.

req_alloc Allocate a request. Set this if you need to allocate a struct larger than *struct media_request*. **req_alloc** and **req_free** must either both be set or both be NULL.

req_free Free a request. Set this if **req_alloc** was set as well, leave to NULL otherwise.

req_validate Validate a request, but do not queue yet. The `req_queue_mutex` lock is held when this op is called.

req_queue Queue a validated request, cannot fail. If something goes wrong when queueing this request then it should be marked as such internally in the driver and any related buffers must eventually return to vb2 with state `VB2_BUF_STATE_ERROR`. The `req_queue_mutex` lock is held when this op is called. It is important that vb2 buffer objects are queued last after all other object types are queued: queueing a buffer kickstarts the request processing, so all other objects related to the request (and thus the buffer) must be available to

the driver. And once a buffer is queued, then the driver can complete or delete objects from the request before `req_queue` exits.

struct **media_device**
Media device

Definition

```
struct media_device {
    struct device *dev;
    struct media_devnode *devnode;
    char model[32];
    char driver_name[32];
    char serial[40];
    char bus_info[32];
    u32 hw_revision;
    u64 topology_version;
    u32 id;
    struct ida entity_internal_idx;
    int entity_internal_idx_max;
    struct list_head entities;
    struct list_head interfaces;
    struct list_head pads;
    struct list_head links;
    struct list_head entity_notify;
    struct mutex graph_mutex;
    struct media_graph pm_count_walk;
    void *source_priv;
    int (*enable_source)(struct media_entity *entity, struct media_pipeline *pipe);
    void (*disable_source)(struct media_entity *entity);
    const struct media_device_ops *ops;
    struct mutex req_queue_mutex;
    atomic_t request_id;
};
```

Members

dev Parent device

devnode Media device node

model Device model name

driver_name Optional device driver name. If not set, calls to `MEDIA_IOC_DEVICE_INFO` will return `dev->driver->name`. This is needed for USB drivers for example, as otherwise they'll all appear as if the driver name was "usb".

serial Device serial number (optional)

bus_info Unique and stable device location identifier

hw_revision Hardware device revision

topology_version Monotonic counter for storing the version of the graph topology. Should be incremented each time the topology changes.

id Unique ID used on the last registered graph object

entity_internal_idx Unique internal entity ID used by the graph traversal algorithms

entity_internal_idx_max Allocated internal entity indices

entities List of registered entities

interfaces List of registered interfaces

pads List of registered pads

links List of registered links

entity_notify List of registered entity_notify callbacks

graph_mutex Protects access to *struct media_device* data

pm_count_walk Graph walk for power state walk. Access serialised using graph_mutex.

source_priv Driver Private data for enable/disable source handlers

enable_source Enable Source Handler function pointer

disable_source Disable Source Handler function pointer

ops Operation handler callbacks

req_queue_mutex Serialise the MEDIA_REQUEST_IOC_QUEUE ioctl w.r.t. other operations that stop or start streaming.

request_id Used to generate unique request IDs

Description

This structure represents an abstract high-level media device. It allows easy access to entities and provides basic media device-level support. The structure can be allocated directly or embedded in a larger structure.

The parent **dev** is a physical device. It must be set before registering the media device.

model is a descriptive model name exported through sysfs. It doesn't have to be unique.

enable_source is a handler to find source entity for the sink entity and activate the link between them if source entity is free. Drivers should call this handler before accessing the source.

disable_source is a handler to find source entity for the sink entity and deactivate the link between them. Drivers should call this handler to release the source.

Use-case: find tuner entity connected to the decoder entity and check if it is available, and activate the link between them from **enable_source** and deactivate from **disable_source**.

Note: Bridge driver is expected to implement and set the handler when *media_device* is registered or when bridge driver finds the *media_device* during probe. Bridge driver sets *source_priv* with information necessary to run **enable_source** and **disable_source** handlers. Callers should hold *graph_mutex* to access and call **enable_source** and **disable_source** handlers.

```
int media_entity_enum_init(struct media_entity_enum *ent_enum,    struct media_device *mdev)
```

Initialise an entity enumeration

Parameters

struct media_entity_enum *ent_enum Entity enumeration to be initialised

struct media_device *mdev The related media device

Return

zero on success or a negative error code.

void **media_device_init**(struct *media_device* *mdev)
 Initializes a media device element

Parameters

struct media_device *mdev pointer to struct *media_device*

Description

This function initializes the media device prior to its registration. The media device initialization and registration is split in two functions to avoid race conditions and make the media device available to user-space before the media graph has been completed.

So drivers need to first initialize the media device, register any entity within the media device, create pad to pad links and then finally register the media device by calling *media_device_register()* as a final step.

void **media_device_cleanup**(struct *media_device* *mdev)
 Cleans up a media device element

Parameters

struct media_device *mdev pointer to struct *media_device*

Description

This function that will destroy the graph_mutex that is initialized in *media_device_init()*.

int **__media_device_register**(struct *media_device* *mdev, struct module *owner)
 Registers a media device element

Parameters

struct media_device *mdev pointer to struct *media_device*

struct module *owner should be filled with THIS_MODULE

Description

Users, should, instead, call the *media_device_register()* macro.

The caller is responsible for initializing the *media_device* structure before registration. The following fields of *media_device* must be set:

- *media_entity.dev* must point to the parent device (usually a *pci_dev*, *usb_interface* or *platform_device* instance).
- *media_entity.model* must be filled with the device model name as a NUL-terminated UTF-8 string. The device/model revision must not be stored in this field.

The following fields are optional:

- *media_entity.serial* is a unique serial number stored as a NUL-terminated ASCII string. The field is big enough to store a GUID in text form. If the hardware doesn't provide a unique serial number this field must be left empty.
- *media_entity.bus_info* represents the location of the device in the system as a NUL-terminated ASCII string. For PCI/PCIe devices *media_entity.bus_info* must be set to "PCI:" (or "PCIe:") followed by the value of *pci_name()*. For USB devices, the

`usb_make_path()` function must be used. This field is used by applications to distinguish between otherwise identical devices that don't provide a serial number.

- `media_entity.hw_revision` is the hardware device revision in a driver-specific format. When possible the revision should be formatted with the `KERNEL_VERSION()` macro.

Note:

- 1) Upon successful registration a character device named `media[0-9]+` is created. The device major and minor numbers are dynamic. The model name is exported as a sysfs attribute.
 - 2) Unregistering a media device that hasn't been registered is **NOT** safe.
-

Return

returns zero on success or a negative error code.

`media_device_register(mdev)`

Registers a media device element

Parameters

`mdev` pointer to struct `media_device`

Description

This macro calls `__media_device_register()` passing `THIS_MODULE` as the `__media_device_register()` second argument (**owner**).

void `media_device_unregister(struct media_device *mdev)`

Unregisters a media device element

Parameters

`struct media_device *mdev` pointer to struct `media_device`

Description

It is safe to call this function on an unregistered (but initialised) media device.

int `media_device_register_entity(struct media_device *mdev, struct media_entity *entity)`

registers a media entity inside a previously registered media device.

Parameters

`struct media_device *mdev` pointer to struct `media_device`

`struct media_entity *entity` pointer to struct `media_entity` to be registered

Description

Entities are identified by a unique positive integer ID. The media controller framework will such ID automatically. IDs are not guaranteed to be contiguous, and the ID number can change on newer Kernel versions. So, neither the driver nor userspace should hardcode ID numbers to refer to the entities, but, instead, use the framework to find the ID, when needed.

The `media_entity` name, type and flags fields should be initialized before calling `media_device_register_entity()`. Entities embedded in higher-level standard structures can have some of those fields set by the higher-level framework.

If the device has pads, `media_entity_pads_init()` should be called before this function. Otherwise, the `media_entity.pad` and `media_entity.num_pads` should be zeroed before calling this function.

Entities have flags that describe the entity capabilities and state:

MEDIA_ENT_FL_DEFAULT indicates the default entity for a given type. This can be used to report the default audio and video devices or the default camera sensor.

Note: Drivers should set the entity function before calling this function. Please notice that the values `MEDIA_ENT_F_V4L2_SUBDEV_UNKNOWN` and `MEDIA_ENT_F_UNKNOWN` should not be used by the drivers.

void **media_device_unregister_entity**(struct *media_entity* *entity)
unregisters a media entity.

Parameters

struct media_entity *entity pointer to struct *media_entity* to be unregistered

Description

All links associated with the entity and all PADS are automatically unregistered from the `media_device` when this function is called.

Unregistering an entity will not change the IDs of the other entities and the previously used ID will never be reused for a newly registered entities.

When a media device is unregistered, all its entities are unregistered automatically. No manual entities unregistration is then required.

Note: The `media_entity` instance itself must be freed explicitly by the driver if required.

int **media_device_register_entity_notify**(struct *media_device* *mdev, struct *media_entity_notify* *nptr)
Registers a media entity_notify callback

Parameters

struct media_device *mdev The media device

struct media_entity_notify *nptr The `media_entity_notify`

Description

Note: When a new entity is registered, all the registered `media_entity_notify` callbacks are invoked.

void **media_device_unregister_entity_notify**(struct *media_device* *mdev, struct *media_entity_notify* *nptr)
Unregister a media entity_notify callback

Parameters

struct media_device *mdev The media device

struct media_entity_notify *nptr The `media_entity_notify`

void media_device_pci_init(struct *media_device* *mdev, struct pci_dev *pci_dev, const char *name)
create and initialize a struct *media_device* from a PCI device.

Parameters

struct media_device *mdev pointer to struct *media_device*

struct pci_dev *pci_dev pointer to struct pci_dev

const char *name media device name. If NULL, the routine will use the default name for the pci device, given by `pci_name()` macro.

void __media_device_usb_init(struct *media_device* *mdev, struct usb_device *udev, const char *board_name, const char *driver_name)
create and initialize a struct *media_device* from a PCI device.

Parameters

struct media_device *mdev pointer to struct *media_device*

struct usb_device *udev pointer to struct usb_device

const char *board_name media device name. If NULL, the routine will use the usb product name, if available.

const char *driver_name name of the driver. if NULL, the routine will use the name given by `udev->dev->driver->name`, with is usually the wrong thing to do.

Description

Note: It is better to call `media_device_usb_init()` instead, as such macro fills `driver_name` with `KBUILD_MODNAME`.

media_device_usb_init(mdev, udev, name)
create and initialize a struct *media_device* from a PCI device.

Parameters

mdev pointer to struct *media_device*

udev pointer to struct usb_device

name media device name. If NULL, the routine will use the usb product name, if available.

Description

This macro calls `media_device_usb_init()` passing the `media_device_usb_init()` **driver_name** parameter filled with `KBUILD_MODNAME`.

struct media_file_operations
Media device file operations

Definition

```
struct media_file_operations {
    struct module *owner;
    ssize_t (*read) (struct file *, char __user *, size_t, loff_t *);
    ssize_t (*write) (struct file *, const char __user *, size_t, loff_t *);
};
```

```

__poll_t (*poll) (struct file *, struct poll_table_struct *);
long (*ioctl) (struct file *, unsigned int, unsigned long);
long (*compat_ioctl) (struct file *, unsigned int, unsigned long);
int (*open) (struct file *);
int (*release) (struct file *);
};

```

Members

owner should be filled with `THIS_MODULE`

read pointer to the function that implements `read()` syscall

write pointer to the function that implements `write()` syscall

poll pointer to the function that implements `poll()` syscall

ioctl pointer to the function that implements `ioctl()` syscall

compat_ioctl pointer to the function that will handle 32 bits userspace calls to the `ioctl()` syscall on a Kernel compiled with 64 bits.

open pointer to the function that implements `open()` syscall

release pointer to the function that will release the resources allocated by the **open** function.

struct **media_devnode**
Media device node

Definition

```

struct media_devnode {
    struct media_device *media_dev;
    const struct media_file_operations *fops;
    struct device dev;
    struct cdev cdev;
    struct device *parent;
    int minor;
    unsigned long flags;
    void (*release)(struct media_devnode *devnode);
};

```

Members

media_dev pointer to struct *media_device*

fops pointer to struct *media_file_operations* with media device ops

dev pointer to struct `device` containing the media controller device

cdev struct `cdev` pointer character device

parent parent device

minor device node minor number

flags flags, combination of the `MEDIA_FLAG_*` constants

release release callback called at the end of `media_devnode_release()` routine at `media-device.c`.

Description

This structure represents a media-related device node.

The **parent** is a physical device. It must be set by core or device drivers before registering the node.

```
int media_devnode_register(struct media_device *mdev, struct media_devnode *devnode, struct module *owner)
```

register a media device node

Parameters

struct media_device *mdev *struct media_device* we want to register a device node

struct media_devnode *devnode media device node structure we want to register

struct module *owner should be filled with THIS_MODULE

Description

The registration code assigns minor numbers and registers the new device node with the kernel. An error is returned if no free minor number can be found, or if the registration of the device node fails.

Zero is returned on success.

Note that if the `media_devnode_register` call fails, the `release()` callback of the `media_devnode` structure is *not* called, so the caller is responsible for freeing any data.

```
void media_devnode_unregister_prepare(struct media_devnode *devnode)
```

clear the media device node register bit

Parameters

struct media_devnode *devnode the device node to prepare for unregister

Description

This clears the passed device register bit. Future open calls will be met with errors. Should be called before `media_devnode_unregister()` to avoid races with unregister and device file open calls.

This function can safely be called if the device node has never been registered or has already been unregistered.

```
void media_devnode_unregister(struct media_devnode *devnode)
```

unregister a media device node

Parameters

struct media_devnode *devnode the device node to unregister

Description

This unregisters the passed device. Future open calls will be met with errors.

Should be called after `media_devnode_unregister_prepare()`

```
struct media_devnode * media_devnode_data(struct file *filp)
```

returns a pointer to the `media_devnode`

Parameters

struct file *filp pointer to struct file

int **media_devnode_is_registered**(struct *media_devnode* *devnode)
returns true if *media_devnode* is registered; false otherwise.

Parameters

struct media_devnode *devnode pointer to struct *media_devnode*.

Note

If mdev is NULL, it also returns false.

enum **media_gobj_type**
type of a graph object

Constants

MEDIA_GRAPH_ENTITY Identify a media entity

MEDIA_GRAPH_PAD Identify a media pad

MEDIA_GRAPH_LINK Identify a media link

MEDIA_GRAPH_INTF_DEVNODE Identify a media Kernel API interface via a device node

struct **media_gobj**
Define a graph object.

Definition

```
struct media_gobj {
    struct media_device    *mdev;
    u32 id;
    struct list_head      list;
};
```

Members

mdev Pointer to the struct *media_device* that owns the object

id Non-zero object ID identifier. The ID should be unique inside a *media_device*, as it is composed by `MEDIA_BITS_PER_TYPE` to store the type plus `MEDIA_BITS_PER_ID` to store the ID

list List entry stored in one of the per-type mdev object lists

Description

All objects on the media graph should have this struct embedded

struct **media_entity_enum**
An enumeration of media entities.

Definition

```
struct media_entity_enum {
    unsigned long *bmap;
    int idx_max;
};
```

Members

bmap Bit map in which each bit represents one entity at *struct media_entity->internal_idx*.

idx_max Number of bits in bmap

struct **media_graph**

Media graph traversal state

Definition

```
struct media_graph {
    struct {
        struct media_entity *entity;
        struct list_head *link;
    } stack[MEDIA_ENTITY_ENUM_MAX_DEPTH];
    struct media_entity_enum ent_enum;
    int top;
};
```

Members

stack Graph traversal stack; the stack contains information on the path the media entities to be walked and the links through which they were reached.

stack.entity pointer to *struct media_entity* at the graph.

stack.link pointer to struct *list_head*.

ent_enum Visited entities

top The top of the stack

struct **media_pipeline**

Media pipeline related information

Definition

```
struct media_pipeline {
    int streaming_count;
    struct media_graph graph;
};
```

Members

streaming_count Streaming start count - streaming stop count

graph Media graph walk during pipeline start / stop

struct **media_link**

A link object part of a media graph.

Definition

```
struct media_link {
    struct media_gobj graph_obj;
    struct list_head list;
    union {
        struct media_gobj *gobj0;
        struct media_pad *source;
        struct media_interface *intf;
    };
    union {
```

```

    struct media_gobj *gobj1;
    struct media_pad *sink;
    struct media_entity *entity;
};
struct media_link *reverse;
unsigned long flags;
bool is_backlink;
};

```

Members

graph_obj Embedded structure containing the media object common data

list Linked list associated with an entity or an interface that owns the link.

{unnamed_union} anonymous

gobj0 Part of a union. Used to get the pointer for the first graph_object of the link.

source Part of a union. Used only if the first object (gobj0) is a pad. In that case, it represents the source pad.

intf Part of a union. Used only if the first object (gobj0) is an interface.

{unnamed_union} anonymous

gobj1 Part of a union. Used to get the pointer for the second graph_object of the link.

sink Part of a union. Used only if the second object (gobj1) is a pad. In that case, it represents the sink pad.

entity Part of a union. Used only if the second object (gobj1) is an entity.

reverse Pointer to the link for the reverse direction of a pad to pad link.

flags Link flags, as defined in uapi/media.h (MEDIA_LNK_FL_*)

is_backlink Indicate if the link is a backlink.

enum **media_pad_signal_type**

type of the signal inside a media pad

Constants

PAD_SIGNAL_DEFAULT

Default signal. Use this when all inputs or all outputs are uniquely identified by the pad number.

PAD_SIGNAL_ANALOG

The pad contains an analog signal. It can be Radio Frequency, Intermediate Frequency, a baseband signal or sub-carriers. Tuner inputs, IF-PLL demodulators, composite and s-video signals should use it.

PAD_SIGNAL_DV

Contains a digital video signal, with can be a bitstream of samples taken from an analog TV video source. On such case, it usually contains the VBI data on it.

PAD_SIGNAL_AUDIO

Contains an Intermediate Frequency analog signal from an audio sub-carrier or an audio bitstream. IF signals are provided by tuners and consumed by audio AM/FM decoders. Bitstream audio is provided by an audio decoder.

struct **media_pad**

A media pad graph object.

Definition

```
struct media_pad {
    struct media_gobj graph_obj;
    struct media_entity *entity;
    u16 index;
    enum media_pad_signal_type sig_type;
    unsigned long flags;
};
```

Members

graph_obj Embedded structure containing the media object common data

entity Entity this pad belongs to

index Pad index in the entity pads array, numbered from 0 to n

sig_type Type of the signal inside a media pad

flags Pad flags, as defined in *include/uapi/linux/media.h* (seek for MEDIA_PAD_FL_*)

struct **media_entity_operations**

Media entity operations

Definition

```
struct media_entity_operations {
    int (*get_fwnode_pad)(struct media_entity *entity, struct fwnode_endpoint *endpoint);
    int (*link_setup)(struct media_entity *entity, const struct media_pad *local, const
↳ struct media_pad *remote, u32 flags);
    int (*link_validate)(struct media_link *link);
};
```

Members

get_fwnode_pad Return the pad number based on a fwnode endpoint or a negative value on error. This operation can be used to map a fwnode to a media pad number. Optional.

link_setup Notify the entity of link changes. The operation can return an error, in which case link setup will be cancelled. Optional.

link_validate Return whether a link is valid from the entity point of view. The *media_pipeline_start()* function validates all links by calling this operation. Optional.

Description

Note: Those these callbacks are called with struct *media_device.graph_mutex* mutex held.

enum **media_entity_type**

Media entity type

Constants

`MEDIA_ENTITY_TYPE_BASE`

The entity isn't embedded in another subsystem structure.

`MEDIA_ENTITY_TYPE_VIDEO_DEVICE`

The entity is embedded in a *struct video_device* instance.

`MEDIA_ENTITY_TYPE_V4L2_SUBDEV`

The entity is embedded in a *struct v4l2_subdev* instance.

Description

Media entity objects are often not instantiated directly, but the media entity structure is inherited by (through embedding) other subsystem-specific structures. The media entity type identifies the type of the subclass structure that implements a media entity instance.

This allows runtime type identification of media entities and safe casting to the correct object type. For instance, a media entity structure instance embedded in a *v4l2_subdev* structure instance will have the type `MEDIA_ENTITY_TYPE_V4L2_SUBDEV` and can safely be cast to a *v4l2_subdev* structure using the `container_of()` macro.

`struct media_entity`

A media entity graph object.

Definition

```

struct media_entity {
    struct media_gobj graph_obj;
    const char *name;
    enum media_entity_type obj_type;
    u32 function;
    unsigned long flags;
    u16 num_pads;
    u16 num_links;
    u16 num_backlinks;
    int internal_idx;
    struct media_pad *pads;
    struct list_head links;
    const struct media_entity_operations *ops;
    int stream_count;
    int use_count;
    struct media_pipeline *pipe;
    union {
        struct {
            u32 major;
            u32 minor;
        } dev;
    } info;
};

```

Members

graph_obj Embedded structure containing the media object common data.

name Entity name.

obj_type Type of the object that implements the `media_entity`.

function Entity main function, as defined in *include/uapi/linux/media.h* (seek for MEDIA_ENT_F_*)

flags Entity flags, as defined in *include/uapi/linux/media.h* (seek for MEDIA_ENT_FL_*)

num_pads Number of sink and source pads.

num_links Total number of links, forward and back, enabled and disabled.

num_backlinks Number of backlinks

internal_idx An unique internal entity specific number. The numbers are re-used if entities are unregistered or registered again.

pads Pads array with the size defined by **num_pads**.

links List of data links.

ops Entity operations.

stream_count Stream count for the entity.

use_count Use count for the entity.

pipe Pipeline this entity belongs to.

info Union with devnode information. Kept just for backward compatibility.

info.dev Contains device major and minor info.

info.dev.major device node major, if the device is a devnode.

info.dev.minor device node minor, if the device is a devnode.

Description

Note: **stream_count** and **use_count** reference counts must never be negative, but are signed integers on purpose: a simple `WARN_ON(<0)` check can be used to detect reference count bugs that would make them negative.

struct **media_interface**

A media interface graph object.

Definition

```
struct media_interface {
    struct media_gobj          graph_obj;
    struct list_head          links;
    u32 type;
    u32 flags;
};
```

Members

graph_obj embedded graph object

links List of links pointing to graph entities

type Type of the interface as defined in *include/uapi/linux/media.h* (seek for MEDIA_INTF_T_*)

flags Interface flags as defined in *include/uapi/linux/media.h* (seek for MEDIA_INTF_FL_*)

Description

Note: Currently, no flags for *media_interface* is defined.

struct *media_intf_devnode*

A media interface via a device node.

Definition

```

struct media_intf_devnode {
    struct media_interface      intf;
    u32 major;
    u32 minor;
};

```

Members

intf embedded interface object

major Major number of a device node

minor Minor number of a device node

u32 **media_entity_id**(struct *media_entity* *entity)
return the media entity graph object id

Parameters

struct *media_entity* *entity pointer to *media_entity*

enum *media_gobj_type* **media_type**(struct *media_gobj* *gobj)
return the media object type

Parameters

struct *media_gobj* *gobj Pointer to the struct *media_gobj* graph object

u32 **media_id**(struct *media_gobj* *gobj)
return the media object ID

Parameters

struct *media_gobj* *gobj Pointer to the struct *media_gobj* graph object

u32 **media_gobj_gen_id**(enum *media_gobj_type* type, u64 *local_id*)
encapsulates type and ID on at the object ID

Parameters

enum *media_gobj_type* **type** object type as define at enum *media_gobj_type*.

u64 **local_id** next ID, from struct *media_device.id*.

bool **is_media_entity_v4l2_video_device**(struct *media_entity* *entity)
Check if the entity is a video_device

Parameters

struct *media_entity* *entity pointer to entity

Return

true if the entity is an instance of a `video_device` object and can safely be cast to a `struct video_device` using the `container_of()` macro, or false otherwise.

bool `is_media_entity_v4l2_subdev`(struct `media_entity` *entity)
Check if the entity is a v4l2_subdev

Parameters

struct `media_entity` *entity pointer to entity

Return

true if the entity is an instance of a `v4l2_subdev` object and can safely be cast to a struct `v4l2_subdev` using the `container_of()` macro, or false otherwise.

int `__media_entity_enum_init`(struct `media_entity_enum` *ent_enum, int idx_max)
Initialise an entity enumeration

Parameters

struct `media_entity_enum` *ent_enum Entity enumeration to be initialised

int `idx_max` Maximum number of entities in the enumeration

Return

Returns zero on success or a negative error code.

void `media_entity_enum_cleanup`(struct `media_entity_enum` *ent_enum)
Release resources of an entity enumeration

Parameters

struct `media_entity_enum` *ent_enum Entity enumeration to be released

void `media_entity_enum_zero`(struct `media_entity_enum` *ent_enum)
Clear the entire enum

Parameters

struct `media_entity_enum` *ent_enum Entity enumeration to be cleared

void `media_entity_enum_set`(struct `media_entity_enum` *ent_enum, struct `media_entity` *entity)
Mark a single entity in the enum

Parameters

struct `media_entity_enum` *ent_enum Entity enumeration

struct `media_entity` *entity Entity to be marked

void `media_entity_enum_clear`(struct `media_entity_enum` *ent_enum, struct `media_entity` *entity)
Unmark a single entity in the enum

Parameters

struct `media_entity_enum` *ent_enum Entity enumeration

struct `media_entity` *entity Entity to be unmarked

bool **media_entity_enum_test**(struct *media_entity_enum* *ent_enum, struct *media_entity* *entity)
 Test whether the entity is marked

Parameters

struct media_entity_enum *ent_enum Entity enumeration

struct media_entity *entity Entity to be tested

Description

Returns true if the entity was marked.

bool **media_entity_enum_test_and_set**(struct *media_entity_enum* *ent_enum, struct *media_entity* *entity)
 Test whether the entity is marked, and mark it

Parameters

struct media_entity_enum *ent_enum Entity enumeration

struct media_entity *entity Entity to be tested

Description

Returns true if the entity was marked, and mark it before doing so.

bool **media_entity_enum_empty**(struct *media_entity_enum* *ent_enum)
 Test whether the entire enum is empty

Parameters

struct media_entity_enum *ent_enum Entity enumeration

Return

true if the entity was empty.

bool **media_entity_enum_intersects**(struct *media_entity_enum* *ent_enum1, struct *media_entity_enum* *ent_enum2)
 Test whether two enums intersect

Parameters

struct media_entity_enum *ent_enum1 First entity enumeration

struct media_entity_enum *ent_enum2 Second entity enumeration

Return

true if entity enumerations **ent_enum1** and **ent_enum2** intersect, otherwise false.

gobj_to_entity(*gobj*)
 returns the struct *media_entity* pointer from the **gobj** contained on it.

Parameters

gobj Pointer to the struct *media_gobj* graph object

gobj_to_pad(*gobj*)
 returns the struct *media_pad* pointer from the **gobj** contained on it.

Parameters

gobj Pointer to the struct *media_gobj* graph object

gobj_to_link(*gobj*)

returns the struct *media_link* pointer from the **gobj** contained on it.

Parameters

gobj Pointer to the struct *media_gobj* graph object

gobj_to_intf(*gobj*)

returns the struct *media_interface* pointer from the **gobj** contained on it.

Parameters

gobj Pointer to the struct *media_gobj* graph object

intf_to_devnode(*intf*)

returns the *struct media_intf_devnode* pointer from the **intf** contained on it.

Parameters

intf Pointer to struct *media_intf_devnode*

void **media_gobj_create**(struct *media_device* **mdev*, enum *media_gobj_type* *type*, struct *media_gobj* **gobj*)

Initialize a graph object

Parameters

struct media_device **mdev* Pointer to the *media_device* that contains the object

enum media_gobj_type *type* Type of the object

struct media_gobj **gobj* Pointer to the struct *media_gobj* graph object

Description

This routine initializes the embedded struct *media_gobj* inside a media graph object. It is called automatically if *media_*_create* function calls are used. However, if the object (entity, link, pad, interface) is embedded on some other object, this function should be called before registering the object at the media controller.

void **media_gobj_destroy**(struct *media_gobj* **gobj*)

Stop using a graph object on a media device

Parameters

struct media_gobj **gobj* Pointer to the struct *media_gobj* graph object

Description

This should be called by all routines like *media_device_unregister()* that remove/destroy media graph objects.

int **media_entity_pads_init**(struct *media_entity* **entity*, u16 *num_pads*, struct *media_pad* **pads*)

Initialize the entity pads

Parameters

struct media_entity **entity* entity where the pads belong

u16 num_pads total number of sink and source pads

struct media_pad **pads* Array of **num_pads** pads.

Description

The pads array is managed by the entity driver and passed to `media_entity_pads_init()` where its pointer will be stored in the `media_entity` structure.

If no pads are needed, drivers could either directly fill `media_entity->num_pads` with 0 and `media_entity->pads` with NULL or call this function that will do the same.

As the number of pads is known in advance, the pads array is not allocated dynamically but is managed by the entity driver. Most drivers will embed the pads array in a driver-specific structure, avoiding dynamic allocation.

Drivers must set the direction of every pad in the pads array before calling `media_entity_pads_init()`. The function will initialize the other pads fields.

void **media_entity_cleanup**(struct `media_entity` *entity)
free resources associated with an entity

Parameters

struct media_entity *entity entity where the pads belong

Description

This function must be called during the cleanup phase after unregistering the entity (currently, it does nothing).

int **media_get_pad_index**(struct `media_entity` *entity, bool *is_sink*, enum *media_pad_signal_type sig_type*)
retrieves a pad index from an entity

Parameters

struct media_entity *entity entity where the pads belong

bool is_sink true if the pad is a sink, false if it is a source

enum media_pad_signal_type sig_type type of signal of the pad to be search

Description

This helper function finds the first pad index inside an entity that satisfies both **is_sink** and **sig_type** conditions.

On success, return the pad number. If the pad was not found or the media entity is a NULL pointer, return -EINVAL.

Return

int **media_create_pad_link**(struct `media_entity` *source, u16 *source_pad*, struct `media_entity` *sink, u16 *sink_pad*, u32 *flags*)
creates a link between two entities.

Parameters

struct media_entity *source pointer to `media_entity` of the source pad.

u16 source_pad number of the source pad in the pads array

struct media_entity *sink pointer to `media_entity` of the sink pad.

u16 sink_pad number of the sink pad in the pads array.

u32 flags Link flags, as defined in `include/uapi/linux/media.h` (seek for MEDIA_LNK_FL_*)

Description

Valid values for flags:

MEDIA_LNK_FL_ENABLED Indicates that the link is enabled and can be used to transfer media data. When two or more links target a sink pad, only one of them can be enabled at a time.

MEDIA_LNK_FL_IMMUTABLE Indicates that the link enabled state can't be modified at runtime. If **MEDIA_LNK_FL_IMMUTABLE** is set, then **MEDIA_LNK_FL_ENABLED** must also be set, since an immutable link is always enabled.

Note: Before calling this function, `media_entity_pads_init()` and `media_device_register_entity()` should be called previously for both ends.

```
int media_create_pad_links(const struct media_device *mdev, const
                           u32 source_function, struct media_entity *source, const
                           u16 source_pad, const u32 sink_function, struct me-
                           dia_entity *sink, const u16 sink_pad, u32 flags, const
                           bool allow_both_undefined)
```

creates a link between two entities.

Parameters

const struct media_device *mdev Pointer to the `media_device` that contains the object

const u32 source_function Function of the source entities. Used only if **source** is NULL.

struct media_entity *source pointer to `media_entity` of the source pad. If NULL, it will use all entities that matches the **sink_function**.

const u16 source_pad number of the source pad in the pads array

const u32 sink_function Function of the sink entities. Used only if **sink** is NULL.

struct media_entity *sink pointer to `media_entity` of the sink pad. If NULL, it will use all entities that matches the **sink_function**.

const u16 sink_pad number of the sink pad in the pads array.

u32 flags Link flags, as defined in `include/uapi/linux/media.h`.

const bool allow_both_undefined if true, then both **source** and **sink** can be NULL. In such case, it will create a crossbar between all entities that matches **source_function** to all entities that matches **sink_function**. If false, it will return 0 and won't create any link if both **source** and **sink** are NULL.

Description

Valid values for flags:

A MEDIA_LNK_FL_ENABLED flag indicates that the link is enabled and can be used to transfer media data. If multiple links are created and this flag is passed as an argument, only the first created link will have this flag.

A MEDIA_LNK_FL_IMMUTABLE flag indicates that the link enabled state can't be modified at runtime. If **MEDIA_LNK_FL_IMMUTABLE** is set, then **MEDIA_LNK_FL_ENABLED** must also be set since an immutable link is always enabled.

It is common for some devices to have multiple source and/or sink entities of the same type that should be linked. While `media_create_pad_link()` creates link by link, this function is meant to allow 1:n, n:1 and even cross-bar (n:n) links.

Note: Before calling this function, `media_entity_pads_init()` and `media_device_register_entity()` should be called previously for the entities to be linked.

void **media_entity_remove_links**(struct *media_entity* *entity)
 remove all links associated with an entity

Parameters

struct media_entity *entity pointer to *media_entity*

Description

Note: This is called automatically when an entity is unregistered via `media_device_register_entity()`.

int **__media_entity_setup_link**(struct *media_link* *link, u32 flags)
 Configure a media link without locking

Parameters

struct media_link *link The link being configured

u32 flags Link configuration flags

Description

The bulk of link setup is handled by the two entities connected through the link. This function notifies both entities of the link configuration change.

If the link is immutable or if the current and new configuration are identical, return immediately.

The user is expected to hold `link->source->parent->mutex`. If not, `media_entity_setup_link()` should be used instead.

int **media_entity_setup_link**(struct *media_link* *link, u32 flags)
 changes the link flags properties in runtime

Parameters

struct media_link *link pointer to *media_link*

u32 flags the requested new link flags

Description

The only configurable property is the `MEDIA_LNK_FL_ENABLED` link flag to enable/disable a link. Links marked with the `MEDIA_LNK_FL_IMMUTABLE` link flag can not be enabled or disabled.

When a link is enabled or disabled, the media framework calls the `link_setup` operation for the two entities at the source and sink of the link, in that order. If the second `link_setup` call fails, another `link_setup` call is made on the first entity to restore the original link flags.

Media device drivers can be notified of link setup operations by setting the `media_device.link_notify` pointer to a callback function. If provided, the notification callback will be called before enabling and after disabling links.

Entity drivers must implement the `link_setup` operation if any of their links is non-immutable. The operation must either configure the hardware or store the configuration information to be applied later.

Link configuration must not have any side effect on other links. If an enabled link at a sink pad prevents another link at the same pad from being enabled, the `link_setup` operation must return `-EBUSY` and can't implicitly disable the first enabled link.

Note: The valid values of the flags for the link is the same as described on `media_create_pad_link()`, for pad to pad links or the same as described on `media_create_intf_link()`, for interface to entity links.

```
struct media_link *media_entity_find_link(struct media_pad *source, struct media_pad *sink)
```

Find a link between two pads

Parameters

struct media_pad *source Source pad

struct media_pad *sink Sink pad

Return

returns a pointer to the link between the two entities. If no such link exists, return `NULL`.

```
struct media_pad *media_entity_remote_pad(const struct media_pad *pad)
```

Find the pad at the remote end of a link

Parameters

const struct media_pad *pad Pad at the local end of the link

Description

Search for a remote pad connected to the given pad by iterating over all links originating or terminating at that pad until an enabled link is found.

Return

returns a pointer to the pad at the remote end of the first found enabled link, or `NULL` if no enabled link has been found.

```
int media_entity_get_fwnode_pad(struct media_entity *entity, struct fwnode_handle *fwnode, unsigned long direction_flags)
```

Get pad number from fwnode

Parameters

struct media_entity *entity The entity

struct fwnode_handle *fwnode Pointer to the `fwnode_handle` which should be used to find the pad

unsigned long direction_flags Expected direction of the pad, as defined in `include/uapi/linux/media.h` (seek for `MEDIA_PAD_FL_*`)

Description

This function can be used to resolve the media pad number from a fwnode. This is useful for devices which use more complex mappings of media pads.

If the entity does not implement the `get_fwnode_pad()` operation then this function searches the entity for the first pad that matches the **direction_flags**.

Return

returns the pad number on success or a negative error code.

```
int media_graph_walk_init(struct media_graph *graph, struct media_device *mdev)
    Allocate resources used by graph walk.
```

Parameters

struct media_graph *graph Media graph structure that will be used to walk the graph

struct media_device *mdev Pointer to the *media_device* that contains the object

Description

The caller is required to hold the `media_device` `graph_mutex` during the graph walk until the graph state is released.

Returns zero on success or a negative error code otherwise.

```
void media_graph_walk_cleanup(struct media_graph *graph)
    Release resources used by graph walk.
```

Parameters

struct media_graph *graph Media graph structure that will be used to walk the graph

```
void media_graph_walk_start(struct media_graph *graph, struct media_entity *entity)
    Start walking the media graph at a given entity
```

Parameters

struct media_graph *graph Media graph structure that will be used to walk the graph

struct media_entity *entity Starting entity

Description

Before using this function, `media_graph_walk_init()` must be used to allocate resources used for walking the graph. This function initializes the graph traversal structure to walk the entities graph starting at the given entity. The traversal structure must not be modified by the caller during graph traversal. After the graph walk, the resources must be released using `media_graph_walk_cleanup()`.

```
struct media_entity * media_graph_walk_next(struct media_graph *graph)
    Get the next entity in the graph
```

Parameters

struct media_graph *graph Media graph structure

Description

Perform a depth-first traversal of the given media entities graph.

The graph structure must have been previously initialized with a call to *media_graph_walk_start()*.

Return

returns the next entity in the graph or NULL if the whole graph have been traversed.

int **media_pipeline_start**(struct *media_entity* *entity, struct *media_pipeline* *pipe)
Mark a pipeline as streaming

Parameters

struct media_entity *entity Starting entity

struct media_pipeline *pipe Media pipeline to be assigned to all entities in the pipeline.

Description

Mark all entities connected to a given entity through enabled links, either directly or indirectly, as streaming. The given pipeline object is assigned to every entity in the pipeline and stored in the *media_entity* pipe field.

Calls to this function can be nested, in which case the same number of *media_pipeline_stop()* calls will be required to stop streaming. The pipeline pointer must be identical for all nested calls to *media_pipeline_start()*.

int **__media_pipeline_start**(struct *media_entity* *entity, struct *media_pipeline* *pipe)
Mark a pipeline as streaming

Parameters

struct media_entity *entity Starting entity

struct media_pipeline *pipe Media pipeline to be assigned to all entities in the pipeline.

Description

..note:: This is the non-locking version of *media_pipeline_start()*

void **media_pipeline_stop**(struct *media_entity* *entity)
Mark a pipeline as not streaming

Parameters

struct media_entity *entity Starting entity

Description

Mark all entities connected to a given entity through enabled links, either directly or indirectly, as not streaming. The *media_entity* pipe field is reset to NULL.

If multiple calls to *media_pipeline_start()* have been made, the same number of calls to this function are required to mark the pipeline as not streaming.

void **__media_pipeline_stop**(struct *media_entity* *entity)
Mark a pipeline as not streaming

Parameters

struct media_entity *entity Starting entity

Description

Note: This is the non-locking version of *media_pipeline_stop()*

```
struct media_intf_devnode * media_devnode_create(struct media_device *mdev,
                                                u32 type, u32 flags, u32 major,
                                                u32 minor)
```

creates and initializes a device node interface

Parameters

struct *media_device* *mdev pointer to struct *media_device*

u32 type type of the interface, as given by *include/uapi/linux/media.h* (seek for MEDIA_INTF_T_*) macros.

u32 flags Interface flags, as defined in *include/uapi/linux/media.h* (seek for MEDIA_INTF_FL_*)

u32 major Device node major number.

u32 minor Device node minor number.

Return

if succeeded, returns a pointer to the newly allocated *media_intf_devnode* pointer.

Description

Note: Currently, no flags for *media_interface* is defined.

```
void media_devnode_remove(struct media_intf_devnode *devnode)
    removes a device node interface
```

Parameters

struct *media_intf_devnode* *devnode pointer to *media_intf_devnode* to be freed.

Description

When a device node interface is removed, all links to it are automatically removed.

```
media_create_intf_link(struct media_entity *entity, struct media_interface *intf,
                        u32 flags)
```

creates a link between an entity and an interface

Parameters

struct *media_entity* *entity pointer to *media_entity*

struct *media_interface* *intf pointer to *media_interface*

u32 flags Link flags, as defined in *include/uapi/linux/media.h* (seek for MEDIA_LNK_FL_*)

Description

Valid values for flags:

MEDIA_LNK_FL_ENABLED Indicates that the interface is connected to the entity hardware. That's the default value for interfaces. An interface may be disabled if the hardware is busy due to the usage of some other interface that it is currently controlling the hardware.

A typical example is an hybrid TV device that handle only one type of stream on a given time. So, when the digital TV is streaming, the V4L2 interfaces won't be enabled, as such device is not able to also stream analog TV or radio.

Note: Before calling this function, *media_devnode_create()* should be called for the interface and *media_device_register_entity()* should be called for the interface that will be part of the link.

void **__media_remove_intf_link**(struct *media_link* *link)
remove a single interface link

Parameters

struct media_link *link pointer to *media_link*.

Description

Note: This is an unlocked version of *media_remove_intf_link()*

void **media_remove_intf_link**(struct *media_link* *link)
remove a single interface link

Parameters

struct media_link *link pointer to *media_link*.

Description

Note: Prefer to use this one, instead of *__media_remove_intf_link()*

void **__media_remove_intf_links**(struct *media_interface* *intf)
remove all links associated with an interface

Parameters

struct media_interface *intf pointer to *media_interface*

Description

Note: This is an unlocked version of *media_remove_intf_links()*.

void **media_remove_intf_links**(struct *media_interface* *intf)
remove all links associated with an interface

Parameters

struct media_interface *intf pointer to *media_interface*

Description

Note:

- 1) This is called automatically when an entity is unregistered via `media_device_unregister_entity()` and by `media_devnode_remove()`.
- 2) Prefer to use this one, instead of `__media_remove_intf_links()`.

media_entity_call(*entity, operation, args...*)

Calls a *struct media_entity_operations* operation on an entity

Parameters

entity entity where the **operation** will be called

operation type of the operation. Should be the name of a member of struct *media_entity_operations*.

args... variable arguments

Description

This helper function will check if **operation** is not NULL. On such case, it will issue a call to **operation(entity, args)**.

enum **media_request_state**
media request state

Constants

MEDIA_REQUEST_STATE_IDLE Idle

MEDIA_REQUEST_STATE_VALIDATING Validating the request, no state changes allowed

MEDIA_REQUEST_STATE_QUEUED Queued

MEDIA_REQUEST_STATE_COMPLETE Completed, the request is done

MEDIA_REQUEST_STATE_CLEANSING Cleansing, the request is being re-initiated

MEDIA_REQUEST_STATE_UPDATING The request is being updated, i.e. request objects are being added, modified or removed

NR_OF_MEDIA_REQUEST_STATE The number of media request states, used internally for sanity check purposes

struct **media_request**
Media device request

Definition

```
struct media_request {
    struct media_device *mdev;
    struct kref kref;
    char debug_str[TASK_COMM_LEN + 11];
    enum media_request_state state;
    unsigned int updating_count;
    unsigned int access_count;
    struct list_head objects;
    unsigned int num_incomplete_objects;
    wait_queue_head_t poll_wait;
    spinlock_t lock;
};
```

Members

mdev Media device this request belongs to

kref Reference count

debug_str Prefix for debug messages (process name:fd)

state The state of the request

updating_count count the number of request updates that are in progress

access_count count the number of request accesses that are in progress

objects List of **struct** `media_request_object` request objects

num_incomplete_objects The number of incomplete objects in the request

poll_wait Wait queue for poll

lock Serializes access to this struct

int **media_request_lock_for_access**(struct *media_request* *req)
Lock the request to access its objects

Parameters

struct media_request *req The media request

Description

Use before accessing a completed request. A reference to the request must be held during the access. This usually takes place automatically through a file handle. Use **media_request_unlock_for_access** when done.

void **media_request_unlock_for_access**(struct *media_request* *req)
Unlock a request previously locked for access

Parameters

struct media_request *req The media request

Description

Unlock a request that has previously been locked using **media_request_lock_for_access**.

int **media_request_lock_for_update**(struct *media_request* *req)
Lock the request for updating its objects

Parameters

struct media_request *req The media request

Description

Use before updating a request, i.e. adding, modifying or removing a request object in it. A reference to the request must be held during the update. This usually takes place automatically through a file handle. Use **media_request_unlock_for_update** when done.

void **media_request_unlock_for_update**(struct *media_request* *req)
Unlock a request previously locked for update

Parameters

struct media_request *req The media request

Description

Unlock a request that has previously been locked using **media_request_lock_for_update**.

```
void media_request_get(struct media_request *req)
    Get the media request
```

Parameters

struct media_request *req The media request

Description

Get the media request.

```
void media_request_put(struct media_request *req)
    Put the media request
```

Parameters

struct media_request *req The media request

Description

Put the media request. The media request will be released when the refcount reaches 0.

```
struct media_request * media_request_get_by_fd(struct media_device *mdev,
                                               int request_fd)
    Get a media request by fd
```

Parameters

struct media_device *mdev Media device this request belongs to

int request_fd The file descriptor of the request

Description

Get the request represented by **request_fd** that is owned by the media device.

Return a -EBADR error pointer if requests are not supported by this driver. Return -EINVAL if the request was not found. Return the pointer to the request if found: the caller will have to call **media_request_put** when it finished using the request.

```
int media_request_alloc(struct media_device *mdev, int *alloc_fd)
    Allocate the media request
```

Parameters

struct media_device *mdev Media device this request belongs to

int *alloc_fd Store the request's file descriptor in this int

Description

Allocated the media request and put the fd in **alloc_fd**.

```
struct media_request_object_ops
    Media request object operations
```

Definition

```
struct media_request_object_ops {
    int (*prepare)(struct media_request_object *object);
    void (*unprepare)(struct media_request_object *object);
    void (*queue)(struct media_request_object *object);
    void (*unbind)(struct media_request_object *object);
    void (*release)(struct media_request_object *object);
};
```

Members

prepare Validate and prepare the request object, optional.

unprepare Unprepare the request object, optional.

queue Queue the request object, optional.

unbind Unbind the request object, optional.

release Release the request object, required.

struct **media_request_object**

An opaque object that belongs to a media request

Definition

```
struct media_request_object {
    const struct media_request_object_ops *ops;
    void *priv;
    struct media_request *req;
    struct list_head list;
    struct kref kref;
    bool completed;
};
```

Members

ops object's operations

priv object's priv pointer

req the request this object belongs to (can be NULL)

list List entry of the object for **struct** `media_request`

kref Reference count of the object, acquire before releasing `req->lock`

completed If true, then this object was completed.

Description

An object related to the request. This struct is always embedded in another struct that contains the actual data for this request object.

void **media_request_object_get**(struct *media_request_object* *obj)
Get a media request object

Parameters

struct media_request_object *obj The object

Description

Get a media request object.

void **media_request_object_put**(struct *media_request_object* *obj)
Put a media request object

Parameters

struct media_request_object *obj The object

Description

Put a media request object. Once all references are gone, the object's memory is released.

struct *media_request_object* * **media_request_object_find**(struct *media_request* *req,
const struct *media_request_object_ops* *ops,
void *priv)

Find an object in a request

Parameters

struct media_request *req The media request

const struct media_request_object_ops *ops Find an object with this ops value

void *priv Find an object with this priv value

Description

Both **ops** and **priv** must be non-NULL.

Returns the object pointer or NULL if not found. The caller must call *media_request_object_put()* once it finished using the object.

Since this function needs to walk the list of objects it takes the **req->lock** spin lock to make this safe.

void **media_request_object_init**(struct *media_request_object* *obj)
Initialise a media request object

Parameters

struct media_request_object *obj The object

Description

Initialise a media request object. The object will be released using the release callback of the ops once it has no references (this function initialises references to one).

int **media_request_object_bind**(struct *media_request* *req, const struct *media_request_object_ops* *ops, void *priv, bool is_buffer,
struct *media_request_object* *obj)

Bind a media request object to a request

Parameters

struct media_request *req The media request

const struct media_request_object_ops *ops The object ops for this object

void *priv A driver-specific priv pointer associated with this object

bool is_buffer Set to true if the object a buffer object.

struct media_request_object *obj The object

Description

Bind this object to the request and set the ops and priv values of the object so it can be found later with `media_request_object_find()`.

Every bound object must be unbound or completed by the kernel at some point in time, otherwise the request will never complete. When the request is released all completed objects will be unbound by the request core code.

Buffer objects will be added to the end of the request's object list, non-buffer objects will be added to the front of the list. This ensures that all buffer objects are at the end of the list and that all non-buffer objects that they depend on are processed first.

```
void media_request_object_unbind(struct media_request_object *obj)
    Unbind a media request object
```

Parameters

struct media_request_object *obj The object

Description

Unbind the media request object from the request.

```
void media_request_object_complete(struct media_request_object *obj)
    Mark the media request object as complete
```

Parameters

struct media_request_object *obj The object

Description

Mark the media request object as complete. Only bound objects can be completed.

```
struct media_device * media_device_usb_allocate(struct usb_device *udev, const
                                                char *module_name, struct mod-
                                                ular *owner)
    Allocate and return struct media device
```

Parameters

struct usb_device *udev struct usb_device pointer

const char *module_name should be filled with KBUILD_MODNAME

struct module *owner struct module pointer THIS_MODULE for the driver. THIS_MODULE is null for a built-in driver. It is safe even when THIS_MODULE is null.

Description

This interface should be called to allocate a Media Device when multiple drivers share usb_device and the media device. This interface allocates `media_device` structure and calls `media_device_usb_init()` to initialize it.

```
void media_device_delete(struct media_device *mdev, const char *module_name, struct
                          module *owner)
    Release media device. Calls kref_put().
```

Parameters

struct media_device *mdev struct `media_device` pointer

const char *module_name should be filled with KBUILD_MODNAME

struct module *owner struct module pointer THIS_MODULE for the driver. THIS_MODULE is null for a built-in driver. It is safe even when THIS_MODULE is null.

Description

This interface should be called to put Media Device Instance kref.

2.6 CEC Kernel Support

The CEC framework provides a unified kernel interface for use with HDMI CEC hardware. It is designed to handle a multiple types of hardware (receivers, transmitters, USB dongles). The framework also gives the option to decide what to do in the kernel driver and what should be handled by userspace applications. In addition it integrates the remote control passthrough feature into the kernel's remote control framework.

2.6.1 The CEC Protocol

The CEC protocol enables consumer electronic devices to communicate with each other through the HDMI connection. The protocol uses logical addresses in the communication. The logical address is strictly connected with the functionality provided by the device. The TV acting as the communication hub is always assigned address 0. The physical address is determined by the physical connection between devices.

The CEC framework described here is up to date with the CEC 2.0 specification. It is documented in the HDMI 1.4 specification with the new 2.0 bits documented in the HDMI 2.0 specification. But for most of the features the freely available HDMI 1.3a specification is sufficient:

<https://www.hdmi.org/spec/index>

2.6.2 CEC Adapter Interface

The struct `cec_adapter` represents the CEC adapter hardware. It is created by calling `cec_allocate_adapter()` and deleted by calling `cec_delete_adapter()`:

```
struct cec_adapter *cec_allocate_adapter(const struct cec_adap_ops *ops,
void cec_delete_adapter(struct cec_adapter *adap);
```

To create an adapter you need to pass the following information:

ops: adapter operations which are called by the CEC framework and that you have to implement.

priv: will be stored in `adap->priv` and can be used by the adapter ops. Use `cec_get_drvdata(adap)` to get the priv pointer.

name: the name of the CEC adapter. Note: this name will be copied.

caps: capabilities of the CEC adapter. These capabilities determine the capabilities of the hardware and which parts are to be handled by userspace and which parts are handled by kernelspace. The capabilities are returned by `CEC_ADAP_G_CAPS`.

available_las: the number of simultaneous logical addresses that this adapter can handle. Must be $1 \leq \text{available_las} \leq \text{CEC_MAX_LOG_ADDRS}$.

To obtain the `priv` pointer use this helper function:

```
void *cec_get_drvdata(const struct cec_adapter *adap);
```

To register the `/dev/cecX` device node and the remote control device (if `CEC_CAP_RC` is set) you call:

```
int cec_register_adapter(struct cec_adapter *adap, struct device *parent);
```

where `parent` is the parent device.

To unregister the devices call:

```
void cec_unregister_adapter(struct cec_adapter *adap);
```

Note: if `cec_register_adapter()` fails, then call `cec_delete_adapter()` to clean up. But if `cec_register_adapter()` succeeded, then only call `cec_unregister_adapter()` to clean up, never `cec_delete_adapter()`. The unregister function will delete the adapter automatically once the last user of that `/dev/cecX` device has closed its file handle.

2.6.3 Implementing the Low-Level CEC Adapter

The following low-level adapter operations have to be implemented in your driver:

`cec_adap_ops`

```
struct cec_adap_ops
{
    /* Low-level callbacks */
    int (*adap_enable)(struct cec_adapter *adap, bool enable);
    int (*adap_monitor_all_enable)(struct cec_adapter *adap, bool enable);
    int (*adap_monitor_pin_enable)(struct cec_adapter *adap, bool enable);
    int (*adap_log_addr)(struct cec_adapter *adap, u8 logical_addr);
    int (*adap_transmit)(struct cec_adapter *adap, u8 attempts,
                        u32 signal_free_time, struct cec_msg *msg);
    void (*adap_status)(struct cec_adapter *adap, struct seq_file *file);
    void (*adap_free)(struct cec_adapter *adap);

    /* Error injection callbacks */
    ...

    /* High-level callbacks */
    ...
};
```

The seven low-level ops deal with various aspects of controlling the CEC adapter hardware:

To enable/disable the hardware:

```
int (*adap_enable)(struct cec_adapter *adap, bool enable);
```

This callback enables or disables the CEC hardware. Enabling the CEC hardware means powering it up in a state where no logical addresses are claimed. The physical address will always be valid if `CEC_CAP_NEEDS_HPD` is set. If that capability is not set, then the physical address can change while the CEC hardware is enabled. CEC drivers should not set `CEC_CAP_NEEDS_HPD` unless the hardware design requires that as this will make it impossible to wake up displays that pull the HPD low when in standby mode. The initial state of the CEC adapter after calling `cec_allocate_adapter()` is disabled.

Note that `adap_enable` must return 0 if `enable` is false.

To enable/disable the 'monitor all' mode:

```
int (*adap_monitor_all_enable)(struct cec_adapter *adap, bool enable);
```

If enabled, then the adapter should be put in a mode to also monitor messages that are not for us. Not all hardware supports this and this function is only called if the `CEC_CAP_MONITOR_ALL` capability is set. This callback is optional (some hardware may always be in 'monitor all' mode).

Note that `adap_monitor_all_enable` must return 0 if `enable` is false.

To enable/disable the 'monitor pin' mode:

```
int (*adap_monitor_pin_enable)(struct cec_adapter *adap, bool enable);
```

If enabled, then the adapter should be put in a mode to also monitor CEC pin changes. Not all hardware supports this and this function is only called if the `CEC_CAP_MONITOR_PIN` capability is set. This callback is optional (some hardware may always be in 'monitor pin' mode).

Note that `adap_monitor_pin_enable` must return 0 if `enable` is false.

To program a new logical address:

```
int (*adap_log_addr)(struct cec_adapter *adap, u8 logical_addr);
```

If `logical_addr == CEC_LOG_ADDR_INVALID` then all programmed logical addresses are to be erased. Otherwise the given logical address should be programmed. If the maximum number of available logical addresses is exceeded, then it should return `-ENXIO`. Once a logical address is programmed the CEC hardware can receive directed messages to that address.

Note that `adap_log_addr` must return 0 if `logical_addr` is `CEC_LOG_ADDR_INVALID`.

To transmit a new message:

```
int (*adap_transmit)(struct cec_adapter *adap, u8 attempts,
                    u32 signal_free_time, struct cec_msg *msg);
```

This transmits a new message. The `attempts` argument is the suggested number of attempts for the transmit.

The `signal_free_time` is the number of data bit periods that the adapter should wait when the line is free before attempting to send a message. This value depends on whether this transmit is a retry, a message from a new initiator or a new message for the same initiator. Most hardware will handle this automatically, but in some cases this information is needed.

The `CEC_FREE_TIME_TO_USEC` macro can be used to convert `signal_free_time` to microseconds (one data bit period is 2.4 ms).

To log the current CEC hardware status:

```
void (*adap_status)(struct cec_adapter *adap, struct seq_file *file);
```

This optional callback can be used to show the status of the CEC hardware. The status is available through debugfs: `cat /sys/kernel/debug/cec/cecX/status`

To free any resources when the adapter is deleted:

```
void (*adap_free)(struct cec_adapter *adap);
```

This optional callback can be used to free any resources that might have been allocated by the driver. It's called from `cec_delete_adapter`.

Your adapter driver will also have to react to events (typically interrupt driven) by calling into the framework in the following situations:

When a transmit finished (successfully or otherwise):

```
void cec_transmit_done(struct cec_adapter *adap, u8 status,
                      u8 arb_lost_cnt, u8 nack_cnt, u8 low_drive_cnt,
                      u8 error_cnt);
```

or:

```
void cec_transmit_attempt_done(struct cec_adapter *adap, u8 status);
```

The status can be one of:

CEC_TX_STATUS_OK: the transmit was successful.

CEC_TX_STATUS_ARB_LOST: arbitration was lost: another CEC initiator took control of the CEC line and you lost the arbitration.

CEC_TX_STATUS_NACK: the message was nacked (for a directed message) or acked (for a broadcast message). A retransmission is needed.

CEC_TX_STATUS_LOW_DRIVE: low drive was detected on the CEC bus. This indicates that a follower detected an error on the bus and requested a retransmission.

CEC_TX_STATUS_ERROR: some unspecified error occurred: this can be one of `ARB_LOST` or `LOW_DRIVE` if the hardware cannot differentiate or something else entirely. Some hardware only supports `OK` and `FAIL` as the result of a transmit, i.e. there is no way to differentiate between the different possible errors. In that case map `FAIL` to `CEC_TX_STATUS_NACK` and not to `CEC_TX_STATUS_ERROR`.

CEC_TX_STATUS_MAX_RETRIES: could not transmit the message after trying multiple times. Should only be set by the driver if it has hardware support for retrying messages. If set, then the framework assumes that it doesn't have to make another attempt to transmit the message since the hardware did that already.

The hardware must be able to differentiate between `OK`, `NACK` and 'something else'.

The `*_cnt` arguments are the number of error conditions that were seen. This may be 0 if no information is available. Drivers that do not support hardware retry can just set the counter corresponding to the transmit error to 1, if the hardware does support retry then either set these counters to 0 if the hardware provides no feedback of which errors occurred and how many times, or fill in the correct values as reported by the hardware.

Be aware that calling these functions can immediately start a new transmit if there is one pending in the queue. So make sure that the hardware is in a state where new transmits can be started *before* calling these functions.

The `cec_transmit_attempt_done()` function is a helper for cases where the hardware never retries, so the transmit is always for just a single attempt. It will call `cec_transmit_done()` in turn, filling in 1 for the count argument corresponding to the status. Or all 0 if the status was `OK`.

When a CEC message was received:

```
void cec_received_msg(struct cec_adapter *adap, struct cec_msg *msg);
```

Speaks for itself.

2.6.4 Implementing the interrupt handler

Typically the CEC hardware provides interrupts that signal when a transmit finished and whether it was successful or not, and it provides an interrupt when a CEC message was received.

The CEC driver should always process the transmit interrupts first before handling the receive interrupt. The framework expects to see the `cec_transmit_done` call before the `cec_received_msg` call, otherwise it can get confused if the received message was in reply to the transmitted message.

2.6.5 Optional: Implementing Error Injection Support

If the CEC adapter supports Error Injection functionality, then that can be exposed through the Error Injection callbacks:

```
struct cec_adap_ops {
    /* Low-level callbacks */
    ...

    /* Error injection callbacks */
    int (*error_inj_show)(struct cec_adapter *adap, struct seq_file *sf);
    bool (*error_inj_parse_line)(struct cec_adapter *adap, char *line);

    /* High-level CEC message callback */
    ...
};
```

If both callbacks are set, then an `error-inj` file will appear in `debugfs`. The basic syntax is as follows:

Leading spaces/tabs are ignored. If the next character is a `#` or the end of the line was reached, then the whole line is ignored. Otherwise a command is expected.

This basic parsing is done in the CEC Framework. It is up to the driver to decide what commands to implement. The only requirement is that the command `clear` without any arguments must be implemented and that it will remove all current error injection commands.

This ensures that you can always do `echo clear >error-inj` to clear any error injections without having to know the details of the driver-specific commands.

Note that the output of `error-inj` shall be valid as input to `error-inj`. So this must work:

```
$ cat error-inj >einj.txt
$ cat einj.txt >error-inj
```

The first callback is called when this file is read and it should show the current error injection state:

```
int (*error_inj_show)(struct cec_adapter *adap, struct seq_file *sf);
```

It is recommended that it starts with a comment block with basic usage information. It returns 0 for success and an error otherwise.

The second callback will parse commands written to the error-inj file:

```
bool (*error_inj_parse_line)(struct cec_adapter *adap, char *line);
```

The line argument points to the start of the command. Any leading spaces or tabs have already been skipped. It is a single line only (so there are no embedded newlines) and it is 0-terminated. The callback is free to modify the contents of the buffer. It is only called for lines containing a command, so this callback is never called for empty lines or comment lines.

Return true if the command was valid or false if there were syntax errors.

2.6.6 Implementing the High-Level CEC Adapter

The low-level operations drive the hardware, the high-level operations are CEC protocol driven. The following high-level callbacks are available:

```
struct cec_adap_ops {
    /* Low-level callbacks */
    ...

    /* Error injection callbacks */
    ...

    /* High-level CEC message callback */
    int (*received)(struct cec_adapter *adap, struct cec_msg *msg);
};
```

The received() callback allows the driver to optionally handle a newly received CEC message:

```
int (*received)(struct cec_adapter *adap, struct cec_msg *msg);
```

If the driver wants to process a CEC message, then it can implement this callback. If it doesn't want to handle this message, then it should return -ENOMSG, otherwise the CEC framework assumes it processed this message and it will not do anything with it.

2.6.7 CEC framework functions

CEC Adapter drivers can call the following CEC framework functions:

```
int cec_transmit_msg(struct cec_adapter *adap, struct cec_msg *msg,
```

Transmit a CEC message. If block is true, then wait until the message has been transmitted, otherwise just queue it and return.

```
void cec_s_phys_addr(struct cec_adapter *adap, u16 phys_addr, bool block);
```

Change the physical address. This function will set adap->phys_addr and send an event if it has changed. If cec_s_log_addrs() has been called and the physical address has become valid, then the CEC framework will start claiming the logical addresses. If block is true, then this function won't return until this process has finished.

When the physical address is set to a valid value the CEC adapter will be enabled (see the `adap_enable` op). When it is set to `CEC_PHYS_ADDR_INVALID`, then the CEC adapter will be disabled. If you change a valid physical address to another valid physical address, then this function will first set the address to `CEC_PHYS_ADDR_INVALID` before enabling the new physical address.

```
void cec_s_phys_addr_from_edid(struct cec_adapter *adap,
```

A helper function that extracts the physical address from the edid struct and calls `cec_s_phys_addr()` with that address, or `CEC_PHYS_ADDR_INVALID` if the EDID did not contain a physical address or edid was a NULL pointer.

```
int cec_s_log_addrs(struct cec_adapter *adap,                                struct cec_log_addr
```

Claim the CEC logical addresses. Should never be called if `CEC_CAP_LOG_ADDRS` is set. If block is true, then wait until the logical addresses have been claimed, otherwise just queue it and return. To unconfigure all logical addresses call this function with `log_addrs` set to NULL or with `log_addrs->num_log_addrs` set to 0. The block argument is ignored when unconfiguring. This function will just return if the physical address is invalid. Once the physical address becomes valid, then the framework will attempt to claim these logical addresses.

2.6.8 CEC Pin framework

Most CEC hardware operates on full CEC messages where the software provides the message and the hardware handles the low-level CEC protocol. But some hardware only drives the CEC pin and software has to handle the low-level CEC protocol. The CEC pin framework was created to handle such devices.

Note that due to the close-to-realtime requirements it can never be guaranteed to work 100%. This framework uses highres timers internally, but if a timer goes off too late by more than 300 microseconds wrong results can occur. In reality it appears to be fairly reliable.

One advantage of this low-level implementation is that it can be used as a cheap CEC analyser, especially if interrupts can be used to detect CEC pin transitions from low to high or vice versa.

```
struct cec_pin_ops
    low-level CEC pin operations
```

Definition

```
struct cec_pin_ops {
    int (*read)(struct cec_adapter *adap);
    void (*low)(struct cec_adapter *adap);
    void (*high)(struct cec_adapter *adap);
    bool (*enable_irq)(struct cec_adapter *adap);
    void (*disable_irq)(struct cec_adapter *adap);
    void (*free)(struct cec_adapter *adap);
    void (*status)(struct cec_adapter *adap, struct seq_file *file);
    int (*read_hpd)(struct cec_adapter *adap);
    int (*read_5v)(struct cec_adapter *adap);
    int (*received)(struct cec_adapter *adap, struct cec_msg *msg);
};
```

Members

read read the CEC pin. Returns > 0 if high, 0 if low, or an error if negative.

low drive the CEC pin low.

high stop driving the CEC pin. The pull-up will drive the pin high, unless someone else is driving the pin low.

enable_irq optional, enable the interrupt to detect pin voltage changes.

disable_irq optional, disable the interrupt.

free optional. Free any allocated resources. Called when the adapter is deleted.

status optional, log status information.

read_hpd optional. Read the HPD pin. Returns > 0 if high, 0 if low or an error if negative.

read_5v optional. Read the 5V pin. Returns > 0 if high, 0 if low or an error if negative.

received optional. High-level CEC message callback. Allows the driver to process CEC messages.

Description

These operations (except for the **received** op) are used by the cec pin framework to manipulate the CEC pin.

void **cec_pin_changed**(struct cec_adapter *adap, bool value)
update pin state from interrupt

Parameters

struct cec_adapter *adap pointer to the cec adapter

bool value when true the pin is high, otherwise it is low

Description

If changes of the CEC voltage are detected via an interrupt, then `cec_pin_changed` is called from the interrupt with the new value.

struct cec_adapter * **cec_pin_allocate_adapter**(const struct *cec_pin_ops* *pin_ops,
void *priv, const char *name,
u32 caps)
allocate a pin-based cec adapter

Parameters

const struct cec_pin_ops *pin_ops low-level pin operations

void *priv will be stored in `adap->priv` and can be used by the adapter ops. Use `cec_get_drvdata(adap)` to get the priv pointer.

const char *name the name of the CEC adapter. Note: this name will be copied.

u32 caps capabilities of the CEC adapter. This will be ORed with `CEC_CAP_MONITOR_ALL` and `CEC_CAP_MONITOR_PIN`.

Description

Allocate a cec adapter using the cec pin framework.

Return

a pointer to the cec adapter or an error pointer

2.6.9 CEC Notifier framework

Most drm HDMI implementations have an integrated CEC implementation and no notifier support is needed. But some have independent CEC implementations that have their own driver. This could be an IP block for an SoC or a completely separate chip that deals with the CEC pin. For those cases a drm driver can install a notifier and use the notifier to inform the CEC driver about changes in the physical address.

```
struct cec_notifier * cec_notifier_conn_register(struct device *hdmi_dev, const
                                                char *port_name, const struct
                                                cec_connector_info *conn_info)
    find or create a new cec_notifier for the given HDMI device and connector tuple.
```

Parameters

struct device *hdmi_dev HDMI device that sends the events.

const char *port_name the connector name from which the event occurs. May be NULL if there is always only one HDMI connector created by the HDMI device.

const struct cec_connector_info *conn_info the connector info from which the event occurs (may be NULL)

Description

If a notifier for device **dev** and connector **port_name** already exists, then increase the refcount and return that notifier.

If it doesn't exist, then allocate a new notifier struct and return a pointer to that new struct.

Return NULL if the memory could not be allocated.

```
void cec_notifier_conn_unregister(struct cec_notifier *n)
    decrease refcount and delete when the refcount reaches 0.
```

Parameters

struct cec_notifier *n notifier. If NULL, then this function does nothing.

```
struct cec_notifier * cec_notifier_cec_adap_register(struct device *hdmi_dev,
                                                    const char *port_name, struct
                                                    cec_adapter *adap)
    find or create a new cec_notifier for the given device.
```

Parameters

struct device *hdmi_dev HDMI device that sends the events.

const char *port_name the connector name from which the event occurs. May be NULL if there is always only one HDMI connector created by the HDMI device.

struct cec_adapter *adap the cec adapter that registered this notifier.

Description

If a notifier for device **dev** and connector **port_name** already exists, then increase the refcount and return that notifier.

If it doesn't exist, then allocate a new notifier struct and return a pointer to that new struct.

Return NULL if the memory could not be allocated.

void **cec_notifier_cec_adap_unregister**(struct cec_notifier *n, struct cec_adapter *adap)
decrease refcount and delete when the refcount reaches 0.

Parameters

struct cec_notifier *n notifier. If NULL, then this function does nothing.

struct cec_adapter *adap the cec adapter that registered this notifier.

void **cec_notifier_set_phys_addr**(struct cec_notifier *n, u16 pa)
set a new physical address.

Parameters

struct cec_notifier *n the CEC notifier

u16 pa the CEC physical address

Description

Set a new CEC physical address. Does nothing if **n** == NULL.

void **cec_notifier_set_phys_addr_from_edid**(struct cec_notifier *n, const struct edid *edid)
set parse the PA from the EDID.

Parameters

struct cec_notifier *n the CEC notifier

const struct edid *edid the struct edid pointer

Description

Parses the EDID to obtain the new CEC physical address and set it. Does nothing if **n** == NULL.

struct device * **cec_notifier_parse_hdmi_phandle**(struct device *dev)
find the hdmi device from "hdmi-phandle"

Parameters

struct device *dev the device with the "hdmi-phandle" device tree property

Description

Returns the device pointer referenced by the "hdmi-phandle" property. Note that the refcount of the returned device is not incremented. This device pointer is only used as a key value in the notifier list, but it is never accessed by the CEC driver.

void **cec_notifier_phys_addr_invalidate**(struct cec_notifier *n)
set the physical address to INVALID

Parameters

struct cec_notifier *n the CEC notifier

Description

This is a simple helper function to invalidate the physical address. Does nothing if **n** == NULL.

2.7 Pixel data transmitter and receiver drivers

V4L2 supports various devices that transmit and receive pixel data. Examples of these devices include a camera sensor, a TV tuner and a parallel or a CSI-2 receiver in an SoC.

2.7.1 Bus types

The following busses are the most common. This section discusses these two only.

2.7.1.1 MIPI CSI-2

CSI-2 is a data bus intended for transferring images from cameras to the host SoC. It is defined by the [MIPI alliance](#).

2.7.1.2 Parallel

BT.601 and BT.656 are the most common parallel busses.

2.7.2 Transmitter drivers

Transmitter drivers generally need to provide the receiver drivers with the configuration of the transmitter. What is required depends on the type of the bus. These are common for both busses.

2.7.2.1 Media bus pixel code

See *Media Bus Pixel Codes*.

2.7.2.2 Link frequency

The `V4L2_CID_LINK_FREQ` control is used to tell the receiver the frequency of the bus (i.e. it is not the same as the symbol rate).

2.7.2.3 `.s_stream()` callback

The struct `struct v4l2_subdev_video_ops->s_stream()` callback is used by the receiver driver to control the transmitter driver's streaming state.

2.7.3 CSI-2 transmitter drivers

2.7.3.1 Pixel rate

The pixel rate on the bus is calculated as follows:

```
pixel_rate = link_freq * 2 * nr_of_lanes * 16 / k / bits_per_sample
```

where

Table 1: variables in pixel rate calculation

variable or constant	description
<code>link_freq</code>	The value of the <code>V4L2_CID_LINK_FREQ</code> integer64 menu item.
<code>nr_of_lanes</code>	Number of data lanes used on the CSI-2 link. This can be obtained from the OF endpoint configuration.
2	Data is transferred on both rising and falling edge of the signal.
<code>bits_per_sample</code>	Number of bits per sample.
<code>k</code>	16 for D-PHY and 7 for C-PHY

Note: The pixel rate calculated this way is **not** the same thing as the pixel rate on the camera sensor's pixel array which is indicated by the `V4L2_CID_PIXEL_RATE` control.

2.7.3.2 LP-11 and LP-111 modes

As part of transitioning to high speed mode, a CSI-2 transmitter typically briefly sets the bus to LP-11 or LP-111 state, depending on the PHY. This period may be as short as 100 μ s, during which the receiver observes this state and proceeds its own part of high speed mode transition.

Most receivers are capable of autonomously handling this once the software has configured them to do so, but there are receivers which require software involvement in observing LP-11 or LP-111 state. 100 μ s is a brief period to hit in software, especially when there is no interrupt telling something is happening.

One way to address this is to configure the transmitter side explicitly to LP-11 or LP-111 mode, which requires support from the transmitter hardware. This is not universally available. Many devices return to this state once streaming is stopped while the state after power-on is LP-00 or LP-000.

The `.pre_streamon()` callback may be used to prepare a transmitter for transitioning to streaming state, but not yet start streaming. Similarly, the `.post_streamoff()` callback is used to undo what was done by the `.pre_streamon()` callback. The caller of `.pre_streamon()` is thus required to call `.post_streamoff()` for each successful call of `.pre_streamon()`.

In the context of CSI-2, the `.pre_streamon()` callback is used to transition the transmitter to the LP-11 or LP-111 mode. This also requires powering on the device, so this should be only done when it is needed.

Receiver drivers that do not need explicit LP-11 or LP-111 mode setup are waived from calling the two callbacks.

2.7.3.3 Stopping the transmitter

A transmitter stops sending the stream of images as a result of calling the `.s_stream()` callback. Some transmitters may stop the stream at a frame boundary whereas others stop immediately, effectively leaving the current frame unfinished. The receiver driver should not make assumptions either way, but function properly in both cases.

2.8 Writing camera sensor drivers

2.8.1 CSI-2 and parallel (BT.601 and BT.656) busses

Please see *Pixel data transmitter and receiver drivers*.

2.8.2 Handling clocks

Camera sensors have an internal clock tree including a PLL and a number of divisors. The clock tree is generally configured by the driver based on a few input parameters that are specific to the hardware:: the external clock frequency and the link frequency. The two parameters generally are obtained from system firmware. **No other frequencies should be used in any circumstances.**

The reason why the clock frequencies are so important is that the clock signals come out of the SoC, and in many cases a specific frequency is designed to be used in the system. Using another frequency may cause harmful effects elsewhere. Therefore only the pre-determined frequencies are configurable by the user.

2.8.2.1 ACPI

Read the `clock-frequency_DSD` property to denote the frequency. The driver can rely on this frequency being used.

2.8.2.2 Devicetree

The currently preferred way to achieve this is using `assigned-clocks`, `assigned-clock-parents` and `assigned-clock-rates` properties. See `Documentation/devicetree/bindings/clock/clock-bindings.txt` for more information. The driver then gets the frequency using `clk_get_rate()`.

This approach has the drawback that there's no guarantee that the frequency hasn't been modified directly or indirectly by another driver, or supported by the board's clock tree to begin with. Changes to the Common Clock Framework API are required to ensure reliability.

2.8.3 Frame size

There are two distinct ways to configure the frame size produced by camera sensors.

2.8.3.1 Freely configurable camera sensor drivers

Freely configurable camera sensor drivers expose the device's internal processing pipeline as one or more sub-devices with different cropping and scaling configurations. The output size of the device is the result of a series of cropping and scaling operations from the device's pixel array's size.

An example of such a driver is the CCS driver (see `drivers/media/i2c/ccs`).

2.8.3.2 Register list based drivers

Register list based drivers generally, instead of able to configure the device they control based on user requests, are limited to a number of preset configurations that combine a number of different parameters that on hardware level are independent. How a driver picks such configuration is based on the format set on a source pad at the end of the device's internal pipeline.

Most sensor drivers are implemented this way, see e.g. `drivers/media/i2c/imx319.c` for an example.

2.8.4 Frame interval configuration

There are two different methods for obtaining possibilities for different frame intervals as well as configuring the frame interval. Which one to implement depends on the type of the device.

2.8.4.1 Raw camera sensors

Instead of a high level parameter such as frame interval, the frame interval is a result of the configuration of a number of camera sensor implementation specific parameters. Luckily, these parameters tend to be the same for more or less all modern raw camera sensors.

The frame interval is calculated using the following equation:

$$\text{frame interval} = \frac{(\text{analogue crop width} + \text{horizontal blanking}) * (\text{analogue crop height} + \text{vertical blanking})}{\text{pixel rate}}$$

The formula is bus independent and is applicable for raw timing parameters on large variety of devices beyond camera sensors. Devices that have no analogue crop, use the full source image size, i.e. pixel array size.

Horizontal and vertical blanking are specified by `V4L2_CID_HBLANK` and `V4L2_CID_VBLANK`, respectively. The unit of the `V4L2_CID_HBLANK` control is pixels and the unit of the `V4L2_CID_VBLANK` is lines. The pixel rate in the sensor's **pixel array** is specified by `V4L2_CID_PIXEL_RATE` in the same sub-device. The unit of that control is pixels per second.

Register list based drivers need to implement read-only sub-device nodes for the purpose. Devices that are not register list based need these to configure the device's internal processing pipeline.

The first entity in the linear pipeline is the pixel array. The pixel array may be followed by other entities that are there to allow configuring binning, skipping, scaling or digital crop *Selections: cropping, scaling and composition*.

2.8.4.2 USB cameras etc. devices

USB video class hardware, as well as many cameras offering a similar higher level interface natively, generally use the concept of frame interval (or frame rate) on device level in firmware or hardware. This means lower level controls implemented by raw cameras may not be used on uAPI (or even kAPI) to control the frame interval on these devices.

2.8.5 Power management

Always use runtime PM to manage the power states of your device. Camera sensor drivers are in no way special in this respect: they are responsible for controlling the power state of the device they otherwise control as well. In general, the device must be powered on at least when its registers are being accessed and when it is streaming.

Existing camera sensor drivers may rely on the old `struct v4l2_subdev_core_ops->s_power()` callback for bridge or ISP drivers to manage their power state. This is however **deprecated**. If you feel you need to begin calling an `s_power` from an ISP or a bridge driver, instead please add runtime PM support to the sensor driver you are using. Likewise, new drivers should not use `s_power`.

Please see examples in e.g. `drivers/media/i2c/ov8856.c` and `drivers/media/i2c/ccs/ccs-core.c`. The two drivers work in both ACPI and DT based systems.

2.8.5.1 Control framework

`v4l2_ctrl_handler_setup()` function may not be used in the device's runtime PM `runtime_resume` callback, as it has no way to figure out the power state of the device. This is because the power state of the device is only changed after the power state transition has taken place. The `s_ctrl` callback can be used to obtain device's power state after the power state transition:

```
int pm_runtime_get_if_in_use(struct device *dev);
```

The function returns a non-zero value if it succeeded getting the power count or runtime PM was disabled, in either of which cases the driver may proceed to access the device.

2.9 Media driver-specific documentation

2.9.1 Video4Linux (V4L) drivers

2.9.1.1 The bttv driver

bttv and sound mini howto

There are a lot of different bt848/849/878/879 based boards available. Making video work often is not a big deal, because this is handled completely by the bt8xx chip, which is common on all boards. But sound is handled in slightly different ways on each board.

To handle the grabber boards correctly, there is a array tvcards[] in bttv-cards.c, which holds the information required for each board. Sound will work only, if the correct entry is used (for video it often makes no difference). The bttv driver prints a line to the kernel log, telling which card type is used. Like this one:

```
bttv0: model: BT848(Hauppauge old) [autodetected]
```

You should verify this is correct. If it isn't, you have to pass the correct board type as insmod argument, `insmod bttv card=2` for example. The file Documentation/admin-guide/media/bttv-cardlist.rst has a list of valid arguments for card.

If your card isn't listed there, you might check the source code for new entries which are not listed yet. If there isn't one for your card, you can check if one of the existing entries does work for you (just trial and error...).

Some boards have an extra processor for sound to do stereo decoding and other nice features. The msp34xx chips are used by Hauppauge for example. If your board has one, you might have to load a helper module like `msp3400` to make sound work. If there isn't one for the chip used on your board: Bad luck. Start writing a new one. Well, you might want to check the video4linux mailing list archive first...

Of course you need a correctly installed soundcard unless you have the speakers connected directly to the grabber board. Hint: check the mixer settings too. ALSA for example has everything muted by default.

How sound works in detail

Still doesn't work? Looks like some driver hacking is required. Below is a do-it-yourself description for you.

The bt8xx chips have 32 general purpose pins, and registers to control these pins. One register is the output enable register (`BT848_GPIO_OUT_EN`), it says which pins are actively driven by the bt848 chip. Another one is the data register (`BT848_GPIO_DATA`), where you can get/set the status if these pins. They can be used for input and output.

Most grabber board vendors use these pins to control an external chip which does the sound routing. But every board is a little different. These pins are also used by some companies to drive remote control receiver chips. Some boards use the i2c bus instead of the gpio pins to connect the mux chip.

As mentioned above, there is a array which holds the required information for each known board. You basically have to create a new line for your board. The important fields are these two:

```
struct tvcard
{
    [ ... ]
    u32 gpiomask;
```

```

u32 audiomux[6]; /* Tuner, Radio, external, internal, mute, stereo */
};

```

`gpiomask` specifies which pins are used to control the audio mux chip. The corresponding bits in the output enable register (`BT848_GPIO_OUT_EN`) will be set as these pins must be driven by the `bt848` chip.

The `audiomux[]` array holds the data values for the different inputs (i.e. which pins must be high/low for tuner/mute/...). This will be written to the data register (`BT848_GPIO_DATA`) to switch the audio mux.

What you have to do is figure out the correct values for `gpiomask` and the `audiomux` array. If you have Windows and the drivers for your card installed, you might want to check out if you can read these registers' values used by the Windows driver. A tool to do this is available from <http://btwincap.sourceforge.net/download.html>.

You might also dig around in the `*.ini` files of the Windows applications. You can have a look at the board to see which of the `gpio` pins are connected at all and then start trial-and-error ...

Starting with release 0.7.41 `bttv` has a number of `insmod` options to make the `gpio` debugging easier:

<code>bttv_gpio=0/1</code>	enable/disable <code>gpio</code> debug messages
<code>gpiomask=n</code>	set the <code>gpiomask</code> value
<code>audiomux=i,j,...</code>	set the values of the <code>audiomux</code> array
<code>audioall=a</code>	set the values of the <code>audiomux</code> array (one value for all array elements, useful to check out which effect the particular value has).

The messages printed with `bttv_gpio=1` look like this:

```

bttv0: gpio: en=00000027, out=00000024 in=00ffffd8 [audio: off]

en =  output_en_able register (BT848_GPIO_OUT_EN)
out =  _out_put bits of the data register (BT848_GPIO_DATA),
      i.e. BT848_GPIO_DATA & BT848_GPIO_OUT_EN
in  =  _in_put bits of the data register,
      i.e. BT848_GPIO_DATA & ~BT848_GPIO_OUT_EN

```

2.9.1.2 The cpia2 driver

Authors: Peter Pregler <Peter_Pregler@email.com>, Scott J. Bertin <scot-tbertin@yahoo.com>, and Jarl Totland <Jarl.Totland@bdc.no> for the original cpia driver, which this one was modelled from.

Notes to developers

- This is a driver version stripped of the 2.4 back compatibility and old MJPEG ioctl API. See cpia2.sf.net for 2.4 support.

Programmer's overview of cpia2 driver

Cpia2 is the second generation video coprocessor from VLSI Vision Ltd (now a division of ST Microelectronics). There are two versions. The first is the STV0672, which is capable of up to 30 frames per second (fps) in frame sizes up to CIF, and 15 fps for VGA frames. The STV0676 is an improved version, which can handle up to 30 fps VGA. Both coprocessors can be attached to two CMOS sensors - the vvl6410 CIF sensor and the vvl6500 VGA sensor. These will be referred to as the 410 and the 500 sensors, or the CIF and VGA sensors.

The two chipsets operate almost identically. The core is an 8051 processor, running two different versions of firmware. The 672 runs the VP4 video processor code, the 676 runs VP5. There are a few differences in register mappings for the two chips. In these cases, the symbols defined in the header files are marked with VP4 or VP5 as part of the symbol name.

The cameras appear externally as three sets of registers. Setting register values is the only way to control the camera. Some settings are interdependant, such as the sequence required to power up the camera. I will try to make note of all of these cases.

The register sets are called blocks. Block 0 is the system block. This section is always powered on when the camera is plugged in. It contains registers that control housekeeping functions such as powering up the video processor. The video processor is the VP block. These registers control how the video from the sensor is processed. Examples are timing registers, user mode (vga, qvga), scaling, cropping, framerates, and so on. The last block is the video compressor (VC). The video stream sent from the camera is compressed as Motion JPEG (JPEGA). The VC controls all of the compression parameters. Looking at the file cpia2_registers.h, you can get a full view of these registers and the possible values for most of them.

One or more registers can be set or read by sending a usb control message to the camera. There are three modes for this. Block mode requests a number of contiguous registers. Random mode reads or writes random registers with a tuple structure containing address/value pairs. The repeat mode is only used by VP4 to load a firmware patch. It contains a starting address and a sequence of bytes to be written into a gpio port.

2.9.1.3 The cx2341x driver

Memory at cx2341x chips

This section describes the cx2341x memory map and documents some of the register space.

Note: the memory long words are little-endian ('intel format').

Warning: This information was figured out from searching through the memory and registers, this information may not be correct and is certainly not complete, and was not derived from anything more than searching through the memory space with commands like:

```
ivtvctl -0 min=0x02000000,max=0x020000ff
```

So take this as is, I'm always searching for more stuff, it's a large register space :-).

Memory Map

The cx2341x exposes its entire 64M memory space to the PCI host via the PCI BAR0 (Base Address Register 0). The addresses here are offsets relative to the address held in BAR0.

```
0x00000000-0x00ffffff Encoder memory space
0x00000000-0x0003ffff Encode.rom
???-???      MPEG buffer(s)
???-???      Raw video capture buffer(s)
???-???      Raw audio capture buffer(s)
???-???      Display buffers (6 or 9)

0x01000000-0x01ffffff Decoder memory space
0x01000000-0x0103ffff Decode.rom
???-???      MPEG buffers(s)
0x0114b000-0x0115afff Audio.rom (deprecated?)

0x02000000-0x0200ffff Register Space
```

Registers

The registers occupy the 64k space starting at the 0x02000000 offset from BAR0. All of these registers are 32 bits wide.

DMA Registers 0x000-0xff:

```
0x00 - Control:
      0=reset/cancel, 1=read, 2=write, 4=stop
0x04 - DMA status:
      1=read busy, 2=write busy, 4=read error, 8=write error, 16=link list error
0x08 - pci DMA pointer for read link list
0x0c - pci DMA pointer for write link list
0x10 - read/write DMA enable:
```

```
    1=read enable, 2=write enable
0x14 - always 0xffffffff, if set any lower instability occurs, 0x00 crashes
0x18 - ??
0x1c - always 0x20 or 32, smaller values slow down DMA transactions
0x20 - always value of 0x780a010a
0x24-0x3c - usually just random values???)
0x40 - Interrupt status
0x44 - Write a bit here and shows up in Interrupt status 0x40
0x48 - Interrupt Mask
0x4C - always value of 0xffffdfff,
      if changed to 0xffffffff DMA write interrupts break.
0x50 - always 0xffffffff
0x54 - always 0xffffffff (0x4c, 0x50, 0x54 seem like interrupt masks, are
      3 processors on chip, Java ones, VPU, SPU, APU, maybe these are the
      interrupt masks???)
0x60-0x7C - random values
0x80 - first write linked list reg, for Encoder Memory addr
0x84 - first write linked list reg, for pci memory addr
0x88 - first write linked list reg, for length of buffer in memory addr
      (|0x80000000 or this for last link)
0x8c-0xdc - rest of write linked list reg, 8 sets of 3 total, DMA goes here
      from linked list addr in reg 0x0c, firmware must push through or
      something.
0xe0 - first (and only) read linked list reg, for pci memory addr
0xe4 - first (and only) read linked list reg, for Decoder memory addr
0xe8 - first (and only) read linked list reg, for length of buffer
0xec-0xff - Nothing seems to be in these registers, 0xec-f4 are 0x00000000.
```

Memory locations for Encoder Buffers 0x700-0x7ff:

These registers show offsets of memory locations pertaining to each buffer area used for encoding, have to shift them by <<1 first.

- 0x07F8: Encoder SDRAM refresh
- 0x07FC: Encoder SDRAM pre-charge

Memory locations for Decoder Buffers 0x800-0x8ff:

These registers show offsets of memory locations pertaining to each buffer area used for decoding, have to shift them by <<1 first.

- 0x08F8: Decoder SDRAM refresh
- 0x08FC: Decoder SDRAM pre-charge

Other memory locations:

- 0x2800: Video Display Module control
- 0x2D00: AO (audio output?) control
- 0x2D24: Bytes Flushed
- 0x7000: LSB I2C write clock bit (inverted)
- 0x7004: LSB I2C write data bit (inverted)
- 0x7008: LSB I2C read clock bit
- 0x700c: LSB I2C read data bit

- 0x9008: GPIO get input state
- 0x900c: GPIO set output state
- 0x9020: GPIO direction (Bit7 (GPIO 0..7) - 0:input, 1:output)
- 0x9050: SPU control
- 0x9054: Reset HW blocks
- 0x9058: VPU control
- 0xA018: Bit6: interrupt pending?
- 0xA064: APU command

Interrupt Status Register

The definition of the bits in the interrupt status register 0x0040, and the interrupt mask 0x0048. If a bit is cleared in the mask, then we want our ISR to execute.

- bit 31 Encoder Start Capture
- bit 30 Encoder EOS
- bit 29 Encoder VBI capture
- bit 28 Encoder Video Input Module reset event
- bit 27 Encoder DMA complete
- bit 24 Decoder audio mode change detection event (through event notification)
- bit 22 Decoder data request
- bit 20 Decoder DMA complete
- bit 19 Decoder VBI re-insertion
- bit 18 Decoder DMA err (linked-list bad)

Missing documentation

- Encoder API post(?)
- Decoder API post(?)
- Decoder VTRACE event

The cx2341x firmware upload

This document describes how to upload the cx2341x firmware to the card.

How to find

See the web pages of the various projects that uses this chip for information on how to obtain the firmware.

The firmware stored in a Windows driver can be detected as follows:

- Each firmware image is 256k bytes.
- The 1st 32-bit word of the Encoder image is 0x0000da7
- The 1st 32-bit word of the Decoder image is 0x00003a7
- The 2nd 32-bit word of both images is 0xaa55bb66

How to load

- Issue the FWapi command to stop the encoder if it is running. Wait for the command to complete.
- Issue the FWapi command to stop the decoder if it is running. Wait for the command to complete.
- Issue the I2C command to the digitizer to stop emitting VSYNC events.
- Issue the FWapi command to halt the encoder's firmware.
- Sleep for 10ms.
- Issue the FWapi command to halt the decoder's firmware.
- Sleep for 10ms.
- Write 0x00000000 to register 0x2800 to stop the Video Display Module.
- Write 0x00000005 to register 0x2D00 to stop the AO (audio output?).
- Write 0x00000000 to register 0xA064 to ping? the APU.
- Write 0xFFFFFFFFE to register 0x9058 to stop the VPU.
- Write 0xFFFFFFFF to register 0x9054 to reset the HW blocks.
- Write 0x00000001 to register 0x9050 to stop the SPU.
- Sleep for 10ms.
- Write 0x0000001A to register 0x07FC to init the Encoder SDRAM's pre-charge.
- Write 0x80000640 to register 0x07F8 to init the Encoder SDRAM's refresh to 1us.
- Write 0x0000001A to register 0x08FC to init the Decoder SDRAM's pre-charge.
- Write 0x80000640 to register 0x08F8 to init the Decoder SDRAM's refresh to 1us.
- Sleep for 512ms. (600ms is recommended)
- Transfer the encoder's firmware image to offset 0 in Encoder memory space.
- Transfer the decoder's firmware image to offset 0 in Decoder memory space.
- Use a read-modify-write operation to Clear bit 0 of register 0x9050 to re-enable the SPU.

- Sleep for 1 second.
- Use a read-modify-write operation to Clear bits 3 and 0 of register 0x9058 to re-enable the VPU.
- Sleep for 1 second.
- Issue status API commands to both firmware images to verify.

How to call the firmware API

The preferred calling convention is known as the firmware mailbox. The mailboxes are basically a fixed length array that serves as the call-stack.

Firmware mailboxes can be located by searching the encoder and decoder memory for a 16 byte signature. That signature will be located on a 256-byte boundary.

Signature:

```
0x78, 0x56, 0x34, 0x12, 0x12, 0x78, 0x56, 0x34,
0x34, 0x12, 0x78, 0x56, 0x56, 0x34, 0x12, 0x78
```

The firmware implements 20 mailboxes of 20 32-bit words. The first 10 are reserved for API calls. The second 10 are used by the firmware for event notification.

Index	Name
0	Flags
1	Command
2	Return value
3	Timeout
4-19	Parameter/Result

The flags are defined in the following table. The direction is from the perspective of the firmware.

Bit	Direction	Purpose
2	O	Firmware has processed the command.
1	I	Driver has finished setting the parameters.
0	I	Driver is using this mailbox.

The command is a 32-bit enumerator. The API specifics may be found in this chapter.

The return value is a 32-bit enumerator. Only two values are currently defined:

- 0=success
- -1=command undefined.

There are 16 parameters/results 32-bit fields. The driver populates these fields with values for all the parameters required by the call. The driver overwrites these fields with result values returned by the call.

The timeout value protects the card from a hung driver thread. If the driver doesn't handle the completed call within the timeout specified, the firmware will reset that mailbox.

To make an API call, the driver iterates over each mailbox looking for the first one available (bit 0 has been cleared). The driver sets that bit, fills in the command enumerator, the timeout value and any required parameters. The driver then sets the parameter ready bit (bit 1). The firmware scans the mailboxes for pending commands, processes them, sets the result code, populates the result value array with that call's return values and sets the call complete bit (bit 2). Once bit 2 is set, the driver should retrieve the results and clear all the flags. If the driver does not perform this task within the time set in the timeout register, the firmware will reset that mailbox.

Event notifications are sent from the firmware to the host. The host tells the firmware which events it is interested in via an API call. That call tells the firmware which notification mailbox to use. The firmware signals the host via an interrupt. Only the 16 Results fields are used, the Flags, Command, Return value and Timeout words are not used.

OSD firmware API description

Note: this API is part of the decoder firmware, so it's cx23415 only.

CX2341X_OSD_GET_FRAMEBUFFER

Enum: 65/0x41

Description

Return base and length of contiguous OSD memory.

Result[0]

OSD base address

Result[1]

OSD length

CX2341X_OSD_GET_PIXEL_FORMAT

Enum: 66/0x42

Description

Query OSD format

Result[0]

0=8bit index 1=16bit RGB 5:6:5 2=16bit ARGB 1:5:5:5 3=16bit ARGB 1:4:4:4 4=32bit ARGB 8:8:8:8

CX2341X_OSD_SET_PIXEL_FORMAT

Enum: 67/0x43

Description

Assign pixel format

Param[0]

- 0=8bit index
- 1=16bit RGB 5:6:5
- 2=16bit ARGB 1:5:5:5
- 3=16bit ARGB 1:4:4:4
- 4=32bit ARGB 8:8:8:8

CX2341X_OSD_GET_STATE

Enum: 68/0x44

Description

Query OSD state

Result[0]

- Bit 0 0=off, 1=on
- Bits 1:2 alpha control
- Bits 3:5 pixel format

CX2341X_OSD_SET_STATE

Enum: 69/0x45

Description

OSD switch

Param[0]

0=off, 1=on

CX2341X_OSD_GET_OSD_COORDS

Enum: 70/0x46

Description

Retrieve coordinates of OSD area blended with video

Result[0]

OSD buffer address

Result[1]

Stride in pixels

Result[2]

Lines in OSD buffer

Result[3]

Horizontal offset in buffer

Result[4]

Vertical offset in buffer

CX2341X_OSD_SET_OSD_COORDS

Enum: 71/0x47

Description

Assign the coordinates of the OSD area to blend with video

Param[0]

buffer address

Param[1]

buffer stride in pixels

Param[2]

lines in buffer

Param[3]

horizontal offset

Param[4]

vertical offset

CX2341X_OSD_GET_SCREEN_COORDS

Enum: 72/0x48

Description

Retrieve OSD screen area coordinates

Result[0]

top left horizontal offset

Result[1]

top left vertical offset

Result[2]

bottom right horizontal offset

Result[3]

bottom right vertical offset

CX2341X_OSD_SET_SCREEN_COORDS

Enum: 73/0x49

Description

Assign the coordinates of the screen area to blend with video

Param[0]

top left horizontal offset

Param[1]

top left vertical offset

Param[2]

bottom left horizontal offset

Param[3]

bottom left vertical offset

CX2341X_OSD_GET_GLOBAL_ALPHA

Enum: 74/0x4A

Description

Retrieve OSD global alpha

Result[0]

global alpha: 0=off, 1=on

Result[1]

bits 0:7 global alpha

CX2341X_OSD_SET_GLOBAL_ALPHA

Enum: 75/0x4B

Description

Update global alpha

Param[0]

global alpha: 0=off, 1=on

Param[1]

global alpha (8 bits)

Param[2]

local alpha: 0=on, 1=off

CX2341X_OSD_SET_BLEND_COORDS

Enum: 78/0x4C

Description

Move start of blending area within display buffer

Param[0]

horizontal offset in buffer

Param[1]

vertical offset in buffer

CX2341X_OSD_GET_FLICKER_STATE

Enum: 79/0x4F

Description

Retrieve flicker reduction module state

Result[0]

flicker state: 0=off, 1=on

CX2341X_OSD_SET_FLICKER_STATE

Enum: 80/0x50

Description

Set flicker reduction module state

Param[0]

State: 0=off, 1=on

CX2341X_OSD_BLT_COPY

Enum: 82/0x52

Description

BLT copy

Param[0]

```
'0000' zero
'0001' ~destination AND ~source
'0010' ~destination AND source
'0011' ~destination
'0100' destination AND ~source
'0101' ~source
'0110' destination XOR source
'0111' ~destination OR ~source
'1000' ~destination AND ~source
'1001' destination XNOR source
'1010' source
'1011' ~destination OR source
'1100' destination
'1101' destination OR ~source
'1110' destination OR source
'1111' one
```

Param[1]

Resulting alpha blending

- '01' source_alpha
- '10' destination_alpha
- '11' source_alpha*destination_alpha+1 (zero if both source and destination alpha are zero)

Param[2]

```
'00' output_pixel = source_pixel
'01' if source_alpha=0:
    output_pixel = destination_pixel
    if 256 > source_alpha > 1:
        output_pixel = ((source_alpha + 1)*source_pixel +
            (255 - source_alpha)*destination_pixel)/256
'10' if destination_alpha=0:
    output_pixel = source_pixel
    if 255 > destination_alpha > 0:
        output_pixel = ((255 - destination_alpha)*source_pixel +
            (destination_alpha + 1)*destination_pixel)/256
'11' if source_alpha=0:
    source_temp = 0
    if source_alpha=255:
        source_temp = source_pixel*256
    if 255 > source_alpha > 0:
        source_temp = source_pixel*(source_alpha + 1)
    if destination_alpha=0:
        destination_temp = 0
    if destination_alpha=255:
        destination_temp = destination_pixel*256
    if 255 > destination_alpha > 0:
        destination_temp = destination_pixel*(destination_alpha + 1)
    output_pixel = (source_temp + destination_temp)/256
```

Param[3]

width

Param[4]

height

Param[5]

destination pixel mask

Param[6]

destination rectangle start address

Param[7]

destination stride in dwords

Param[8]

source stride in dwords

Param[9]

source rectangle start address

CX2341X_OSD_BLT_FILL

Enum: 83/0x53

Description

BLT fill color

Param[0]

Same as Param[0] on API 0x52

Param[1]

Same as Param[1] on API 0x52

Param[2]

Same as Param[2] on API 0x52

Param[3]

width

Param[4]

height

Param[5]

destination pixel mask

Param[6]

destination rectangle start address

Param[7]

destination stride in dwords

Param[8]

color fill value

CX2341X_OSD_BLT_TEXT

Enum: 84/0x54

Description

BLT for 8 bit alpha text source

Param[0]

Same as Param[0] on API 0x52

Param[1]

Same as Param[1] on API 0x52

Param[2]

Same as Param[2] on API 0x52

Param[3]

width

Param[4]

height

Param[5]

destination pixel mask

Param[6]

destination rectangle start address

Param[7]

destination stride in dwords

Param[8]

source stride in dwords

Param[9]

source rectangle start address

Param[10]

color fill value

CX2341X_OSD_SET_FRAMEBUFFER_WINDOW

Enum: 86/0x56

Description

Positions the main output window on the screen. The coordinates must be such that the entire window fits on the screen.

Param[0]

window width

Param[1]

window height

Param[2]

top left window corner horizontal offset

Param[3]

top left window corner vertical offset

CX2341X_OSD_SET_CHROMA_KEY

Enum: 96/0x60

Description

Chroma key switch and color

Param[0]

state: 0=off, 1=on

Param[1]

color

CX2341X_OSD_GET_ALPHA_CONTENT_INDEX

Enum: 97/0x61

Description

Retrieve alpha content index

Result[0]

alpha content index, Range 0:15

CX2341X_OSD_SET_ALPHA_CONTENT_INDEX

Enum: 98/0x62

Description

Assign alpha content index

Param[0]

alpha content index, range 0:15

Encoder firmware API description

CX2341X_ENC_PING_FW

Enum: 128/0x80

Description

Does nothing. Can be used to check if the firmware is responding.

CX2341X_ENC_START_CAPTURE

Enum: 129/0x81

Description

Commences the capture of video, audio and/or VBI data. All encoding parameters must be initialized prior to this API call. Captures frames continuously or until a predefined number of frames have been captured.

Param[0]

Capture stream type:

- 0=MPEG
- 1=Raw
- 2=Raw passthrough
- 3=VBI

Param[1]

Bitmask:

- Bit 0 when set, captures YUV
- Bit 1 when set, captures PCM audio
- Bit 2 when set, captures VBI (same as param[0]=3)
- Bit 3 when set, the capture destination is the decoder (same as param[0]=2)
- Bit 4 when set, the capture destination is the host

Note: this parameter is only meaningful for RAW capture type.

CX2341X_ENC_STOP_CAPTURE

Enum: 130/0x82

Description

Ends a capture in progress

Param[0]

- 0=stop at end of GOP (generates IRQ)
- 1=stop immediate (no IRQ)

Param[1]

Stream type to stop, see param[0] of API 0x81

Param[2]

Subtype, see param[1] of API 0x81

CX2341X_ENC_SET_AUDIO_ID

Enum: 137/0x89

Description

Assigns the transport stream ID of the encoded audio stream

Param[0]

Audio Stream ID

CX2341X_ENC_SET_VIDEO_ID

Enum: 139/0x8B

Description

Set video transport stream ID

Param[0]

Video stream ID

CX2341X_ENC_SET_PCR_ID

Enum: 141/0x8D

Description

Assigns the transport stream ID for PCR packets

Param[0]

PCR Stream ID

CX2341X_ENC_SET_FRAME_RATE

Enum: 143/0x8F

Description

Set video frames per second. Change occurs at start of new GOP.

Param[0]

- 0=30fps
- 1=25fps

CX2341X_ENC_SET_FRAME_SIZE

Enum: 145/0x91

Description

Select video stream encoding resolution.

Param[0]

Height in lines. Default 480

Param[1]

Width in pixels. Default 720

CX2341X_ENC_SET_BIT_RATE

Enum: 149/0x95

Description

Assign average video stream bitrate.

Param[0]

0=variable bitrate, 1=constant bitrate

Param[1]

bitrate in bits per second

Param[2]

peak bitrate in bits per second, divided by 400

Param[3]

Mux bitrate in bits per second, divided by 400. May be 0 (default).

Param[4]

Rate Control VBR Padding

Param[5]

VBV Buffer used by encoder

Note:

- 1) Param[3] and Param[4] seem to be always 0
 - 2) Param[5] doesn't seem to be used.
-

CX2341X_ENC_SET_GOP_PROPERTIES

Enum: 151/0x97

Description

Setup the GOP structure

Param[0]

GOP size (maximum is 34)

Param[1]

Number of B frames between the I and P frame, plus 1. For example: IBBPBBPBBPBB -> GOP size: 12, number of B frames: 2+1 = 3

Note: GOP size must be a multiple of (B-frames + 1).

CX2341X_ENC_SET_ASPECT_RATIO

Enum: 153/0x99

Description

Sets the encoding aspect ratio. Changes in the aspect ratio take effect at the start of the next GOP.

Param[0]

- '0000' forbidden
- '0001' 1:1 square
- '0010' 4:3
- '0011' 16:9
- '0100' 2.21:1
- '0101' to '1111' reserved

CX2341X_ENC_SET_DNR_FILTER_MODE

Enum: 155/0x9B

Description

Assign Dynamic Noise Reduction operating mode

Param[0]

Bit0: Spatial filter, set=auto, clear=manual Bit1: Temporal filter, set=auto, clear=manual

Param[1]

Median filter:

- 0=Disabled
- 1=Horizontal
- 2=Vertical
- 3=Horiz/Vert
- 4=Diagonal

CX2341X_ENC_SET_DNR_FILTER_PROPS

Enum: 157/0x9D

Description

These Dynamic Noise Reduction filter values are only meaningful when the respective filter is set to “manual” (See API 0x9B)

Param[0]

Spatial filter: default 0, range 0:15

Param[1]

Temporal filter: default 0, range 0:31

CX2341X_ENC_SET_CORING_LEVELS

Enum: 159/0x9F

Description

Assign Dynamic Noise Reduction median filter properties.

Param[0]

Threshold above which the luminance median filter is enabled. Default: 0, range 0:255

Param[1]

Threshold below which the luminance median filter is enabled. Default: 255, range 0:255

Param[2]

Threshold above which the chrominance median filter is enabled. Default: 0, range 0:255

Param[3]

Threshold below which the chrominance median filter is enabled. Default: 255, range 0:255

CX2341X_ENC_SET_SPATIAL_FILTER_TYPE

Enum: 161/0xA1

Description

Assign spatial prefilter parameters

Param[0]

Luminance filter

- 0=Off
- 1=1D Horizontal
- 2=1D Vertical
- 3=2D H/V Separable (default)
- 4=2D Symmetric non-separable

Param[1]

Chrominance filter

- 0=Off
- 1=1D Horizontal (default)

CX2341X_ENC_SET_VBI_LINE

Enum: 183/0xB7

Description

Selects VBI line number.

Param[0]

- Bits 0:4 line number
- Bit 31 0=top_field, 1=bottom_field
- Bits 0:31 all set specifies “all lines”

Param[1]

VBI line information features: 0=disabled, 1=enabled

Param[2]

Slicing: 0=None, 1=Closed Caption Almost certainly not implemented. Set to 0.

Param[3]

Luminance samples in this line. Almost certainly not implemented. Set to 0.

Param[4]

Chrominance samples in this line Almost certainly not implemented. Set to 0.

CX2341X_ENC_SET_STREAM_TYPE

Enum: 185/0xB9

Description

Assign stream type

Note: Transport stream is not working in recent firmwares. And in older firmwares the timestamps in the TS seem to be unreliable.

Param[0]

- 0=Program stream
- 1=Transport stream
- 2=MPEG1 stream
- 3=PES A/V stream
- 5=PES Video stream
- 7=PES Audio stream
- 10=DVD stream
- 11=VCD stream
- 12=SVCD stream
- 13=DVD_S1 stream
- 14=DVD_S2 stream

CX2341X_ENC_SET_OUTPUT_PORT

Enum: 187/0xBB

Description

Assign stream output port. Normally 0 when the data is copied through the PCI bus (DMA), and 1 when the data is streamed to another chip (pvrusb and cx88-blackbird).

Param[0]

- 0=Memory (default)
- 1=Streaming
- 2=Serial

Param[1]

Unknown, but leaving this to 0 seems to work best. Indications are that this might have to do with USB support, although passing anything but 0 only breaks things.

CX2341X_ENC_SET_AUDIO_PROPERTIES

Enum: 189/0xBD

Description

Set audio stream properties, may be called while encoding is in progress.

Note: All bitfields are consistent with ISO11172 documentation except bits 2:3 which ISO docs define as:

- '11' Layer I
- '10' Layer II
- '01' Layer III
- '00' Undefined

This discrepancy may indicate a possible error in the documentation. Testing indicated that only Layer II is actually working, and that the minimum bitrate should be 192 kbps.

Param[0]

Bitmask:

0:1	'00'	44.1Khz		
	'01'	48Khz		
	'10'	32Khz		
	'11'	reserved		
2:3	'01'	=Layer I		
	'10'	=Layer II		
4:7	Bitrate:			
		Index Layer I	Layer II	
		-----+-----+-----		
	'0000'	free format	free format	

'0001'	32 kbit/s	32 kbit/s
'0010'	64 kbit/s	48 kbit/s
'0011'	96 kbit/s	56 kbit/s
'0100'	128 kbit/s	64 kbit/s
'0101'	160 kbit/s	80 kbit/s
'0110'	192 kbit/s	96 kbit/s
'0111'	224 kbit/s	112 kbit/s
'1000'	256 kbit/s	128 kbit/s
'1001'	288 kbit/s	160 kbit/s
'1010'	320 kbit/s	192 kbit/s
'1011'	352 kbit/s	224 kbit/s
'1100'	384 kbit/s	256 kbit/s
'1101'	416 kbit/s	320 kbit/s
'1110'	448 kbit/s	384 kbit/s

.. note::

For Layer II, not all combinations of total bitrate and mode are allowed. See ISO11172-3 3-Annex B, Table 3-B.2

8:9 '00'=Stereo
 '01'=JointStereo
 '10'=Dual
 '11'=Mono

.. note::

The cx23415 cannot decode Joint Stereo properly.

10:11 Mode Extension used in joint_stereo mode.
 In Layer I and II they indicate which subbands are in intensity_stereo. All other subbands are coded in stereo.
 '00' subbands 4-31 in intensity_stereo, bound==4
 '01' subbands 8-31 in intensity_stereo, bound==8
 '10' subbands 12-31 in intensity_stereo, bound==12
 '11' subbands 16-31 in intensity_stereo, bound==16

12:13 Emphasis:
 '00' None
 '01' 50/15uS
 '10' reserved
 '11' CCITT J.17

14 CRC:
 '0' off
 '1' on

15 Copyright:
 '0' off
 '1' on

16 Generation:
 '0' copy
 '1' original

CX2341X_ENC_HALT_FW

Enum: 195/0xC3

Description

The firmware is halted and no further API calls are serviced until the firmware is uploaded again.

CX2341X_ENC_GET_VERSION

Enum: 196/0xC4

Description

Returns the version of the encoder firmware.

Result[0]

Version bitmask: - Bits 0:15 build - Bits 16:23 minor - Bits 24:31 major

CX2341X_ENC_SET_GOP_CLOSURE

Enum: 197/0xC5

Description

Assigns the GOP open/close property.

Param[0]

- 0=Open
- 1=Closed

CX2341X_ENC_GET_SEQ_END

Enum: 198/0xC6

Description

Obtains the sequence end code of the encoder's buffer. When a capture is started a number of interrupts are still generated, the last of which will have Result[0] set to 1 and Result[1] will contain the size of the buffer.

Result[0]

State of the transfer (1 if last buffer)

Result[1]

If Result[0] is 1, this contains the size of the last buffer, undefined otherwise.

CX2341X_ENC_SET_PGM_INDEX_INFO

Enum: 199/0xC7

Description

Sets the Program Index Information. The information is stored as follows:

```

struct info {
    u32 length;           // Length of this frame
    u32 offset_low;      // Offset in the file of the
    u32 offset_high;    // start of this frame
    u32 mask1;           // Bits 0-2 are the type mask:
                        // 1=I, 2=P, 4=B
                        // 0=End of Program Index, other fields
                        // are invalid.
    u32 pts;             // The PTS of the frame
    u32 mask2;           // Bit 0 is bit 32 of the pts.
};
u32 table_ptr;
struct info index[400];

```

The table_ptr is the encoder memory address in the table were *new* entries will be written.

Note: This is a ringbuffer, so the table_ptr will wraparound.

Param[0]

Picture Mask: - 0=No index capture - 1=I frames - 3=I,P frames - 7=I,P,B frames
(Seems to be ignored, it always indexes I, P and B frames)

Param[1]

Elements requested (up to 400)

Result[0]

Offset in the encoder memory of the start of the table.

Result[1]

Number of allocated elements up to a maximum of Param[1]

CX2341X_ENC_SET_VBI_CONFIG

Enum: 200/0xC8

Description

Configure VBI settings

Param[0]

Bitmap:

```
0    Mode '0' Sliced, '1' Raw
1:3  Insertion:
      '000' insert in extension & user data
      '001' insert in private packets
      '010' separate stream and user data
      '111' separate stream and private data
8:15 Stream ID (normally 0xBD)
```


Param[1]

Frames per interrupt (max 8). Only valid in raw mode.

Param[2]

Total raw VBI frames. Only valid in raw mode.

Param[3]

Start codes

Param[4]

Stop codes

Param[5]

Lines per frame

Param[6]

Byte per line

Result[0]

Observed frames per interrupt in raw mode only. Range 1 to Param[1]

Result[1]

Observed number of frames in raw mode. Range 1 to Param[2]

Result[2]

Memory offset to start of raw VBI data

CX2341X_ENC_SET_DMA_BLOCK_SIZE

Enum: 201/0xC9

Description

Set DMA transfer block size

Param[0]

DMA transfer block size in bytes or frames. When unit is bytes, supported block sizes are 2^7 , 2^8 and 2^9 bytes.

Param[1]

Unit: 0=bytes, 1=frames

CX2341X_ENC_GET_PREV_DMA_INFO_MB_10

Enum: 202/0xCA

Description

Returns information on the previous DMA transfer in conjunction with bit 27 of the interrupt mask. Uses mailbox 10.

Result[0]

Type of stream

Result[1]

Address Offset

Result[2]

Maximum size of transfer

CX2341X_ENC_GET_PREV_DMA_INFO_MB_9

Enum: 203/0xCB

Description

Returns information on the previous DMA transfer in conjunction with bit 27 or 18 of the interrupt mask. Uses mailbox 9.

Result[0]

Status bits: - 0 read completed - 1 write completed - 2 DMA read error - 3 DMA write error - 4 Scatter-Gather array error

Result[1]

DMA type

Result[2]

Presentation Time Stamp bits 0..31

Result[3]

Presentation Time Stamp bit 32

CX2341X_ENC_SCHED_DMA_TO_HOST

Enum: 204/0xCC

Description

Setup DMA to host operation

Param[0]

Memory address of link list

Param[1]

Length of link list (wtf: what units ???)

Param[2]

DMA type (0=MPEG)

CX2341X_ENC_INITIALIZE_INPUT

Enum: 205/0xCD

Description

Initializes the video input

CX2341X_ENC_SET_FRAME_DROP_RATE

Enum: 208/0xD0

Description

For each frame captured, skip specified number of frames.

Param[0]

Number of frames to skip

CX2341X_ENC_PAUSE_ENCODER

Enum: 210/0xD2

Description

During a pause condition, all frames are dropped instead of being encoded.

Param[0]

- 0=Pause encoding
- 1=Continue encoding

CX2341X_ENC_REFRESH_INPUT

Enum: 211/0xD3

Description

Refreshes the video input

CX2341X_ENC_SET_COPYRIGHT

Enum: 212/0xD4

Description

Sets stream copyright property

Param[0]

- 0=Stream is not copyrighted
- 1=Stream is copyrighted

CX2341X_ENC_SET_EVENT_NOTIFICATION

Enum: 213/0xD5

Description

Setup firmware to notify the host about a particular event. Host must unmask the interrupt bit.

Param[0]

Event (0=refresh encoder input)

Param[1]

Notification 0=disabled 1=enabled

Param[2]

Interrupt bit

Param[3]

Mailbox slot, -1 if no mailbox required.

CX2341X_ENC_SET_NUM_VSYNC_LINES

Enum: 214/0xD6

Description

Depending on the analog video decoder used, this assigns the number of lines for field 1 and 2.

Param[0]

Field 1 number of lines: - 0x00EF for SAA7114 - 0x00F0 for SAA7115 - 0x0105 for Micronas

Param[1]

Field 2 number of lines: - 0x00EF for SAA7114 - 0x00F0 for SAA7115 - 0x0106 for Micronas

CX2341X_ENC_SET_PLACEHOLDER

Enum: 215/0xD7

Description

Provides a mechanism of inserting custom user data in the MPEG stream.

Param[0]

- 0=extension & user data
- 1=private packet with stream ID 0xBD

Param[1]

Rate at which to insert data, in units of frames (for private packet) or GOPs (for ext. & user data)

Param[2]

Number of data DWORDs (below) to insert

Param[3]

Custom data 0

Param[4]

Custom data 1

Param[5]

Custom data 2

Param[6]

Custom data 3

Param[7]

Custom data 4

Param[8]

Custom data 5

Param[9]

Custom data 6

Param[10]

Custom data 7

Param[11]

Custom data 8

CX2341X_ENC_MUTE_VIDEO

Enum: 217/0xD9

Description

Video muting

Param[0]

Bit usage:

0	'0'=video not muted
	'1'=video muted, creates frames with the YUV color defined below
1:7	Unused
8:15	V chrominance information
16:23	U chrominance information
24:31	Y luminance information

CX2341X_ENC_MUTE_AUDIO

Enum: 218/0xDA

Description

Audio muting

Param[0]

- 0=audio not muted
- 1=audio muted (produces silent mpeg audio stream)

CX2341X_ENC_SET_VERT_CROP_LINE

Enum: 219/0xDB

Description

Something to do with 'Vertical Crop Line'

Param[0]

If saa7114 and raw VBI capture and 60 Hz, then set to 10001. Else 0.

CX2341X_ENC_MISC

Enum: 220/0xDC

Description

Miscellaneous actions. Not known for 100% what it does. It's really a sort of ioctl call. The first parameter is a command number, the second the value.

Param[0]

Command number:

```

1=set initial SCR value when starting encoding (works).
2=set quality mode (apparently some test setting).
3=setup advanced VIM protection handling.
   Always 1 for the cx23416 and 0 for cx23415.
4=generate DVD compatible PTS timestamps
5=USB flush mode
6=something to do with the quantization matrix
7=set navigation pack insertion for DVD: adds 0xbf (private stream 2)
   packets to the MPEG. The size of these packets is 2048 bytes (including
   the header of 6 bytes: 0x000001bf + length). The payload is zeroed and
   it is up to the application to fill them in. These packets are apparently

```

```
inserted every four frames.  
8=enable scene change detection (seems to be a failure)  
9=set history parameters of the video input module  
10=set input field order of VIM  
11=set quantization matrix  
12=reset audio interface after channel change or input switch (has no argument).  
    Needed for the cx2584x, not needed for the msp4xx, but it doesn't seem to  
    do any harm calling it regardless.  
13=set audio volume delay  
14=set audio delay
```

Param[1]

Command value.

Decoder firmware API description

Note: this API is part of the decoder firmware, so it's cx23415 only.

CX2341X_DEC_PING_FW

Enum: 0/0x00

Description

This API call does nothing. It may be used to check if the firmware is responding.

CX2341X_DEC_START_PLAYBACK

Enum: 1/0x01

Description

Begin or resume playback.

Param[0]

0 based frame number in GOP to begin playback from.

Param[1]

Specifies the number of muted audio frames to play before normal audio resumes. (This is not implemented in the firmware, leave at 0)

CX2341X_DEC_STOP_PLAYBACK

Enum: 2/0x02

Description

Ends playback and clears all decoder buffers. If PTS is not zero, playback stops at specified PTS.

Param[0]

Display 0=last frame, 1=black

Note: this takes effect immediately, so if you want to wait for a PTS, then use '0', otherwise the screen goes to black at once. You can call this later (even if there is no playback) with a 1 value to set the screen to black.

Param[1]

PTS low

Param[2]

PTS high

CX2341X_DEC_SET_PLAYBACK_SPEED

Enum: 3/0x03

Description

Playback stream at speed other than normal. There are two modes of operation:

- Smooth: host transfers entire stream and firmware drops unused frames.
- Coarse: host drops frames based on indexing as required to achieve desired speed.

Param[0]

```
Bitmap:
 0:7  0 normal
      1 fast only "1.5 times"
      n nX fast, 1/nX slow
30   Framedrop:
      '0' during 1.5 times play, every other B frame is dropped
      '1' during 1.5 times play, stream is unchanged (bitrate
          must not exceed 8mbps)
31   Speed:
      '0' slow
      '1' fast
```

Note: n is limited to 2. Anything higher does not result in faster playback. Instead the host should start dropping frames.

Param[1]

Direction: 0=forward, 1=reverse

Note: to make reverse playback work you have to write full GOPs in reverse order.

Param[2]

```
Picture mask:
 1=I frames
 3=I, P frames
 7=I, P, B frames
```

Param[3]

B frames per GOP (for reverse play only)

Note: for reverse playback the Picture Mask should be set to I or I, P. Adding B frames to the mask will result in corrupt video. This field has to be set to the correct value in order to keep the timing correct.

Param[4]

Mute audio: 0=disable, 1=enable

Param[5]

Display 0=frame, 1=field

Param[6]

Specifies the number of muted audio frames to play before normal audio resumes. (Not implemented in the firmware, leave at 0)

CX2341X_DEC_STEP_VIDEO

Enum: 5/0x05

Description

Each call to this API steps the playback to the next unit defined below in the current playback direction.

Param[0]

0=frame, 1=top field, 2=bottom field

CX2341X_DEC_SET_DMA_BLOCK_SIZE

Enum: 8/0x08

Description

Set DMA transfer block size. Counterpart to API 0xC9

Param[0]

DMA transfer block size in bytes. A different size may be specified when issuing the DMA transfer command.

CX2341X_DEC_GET_XFER_INFO

Enum: 9/0x09

Description

This API call may be used to detect an end of stream condition.

Result[0]

Stream type

Result[1]

Address offset

Result[2]

Maximum bytes to transfer

Result[3]

Buffer fullness

CX2341X_DEC_GET_DMA_STATUS

Enum: 10/0x0A

Description

Status of the last DMA transfer

Result[0]

Bit 1 set means transfer complete Bit 2 set means DMA error Bit 3 set means linked list error

Result[1]

DMA type: 0=MPEG, 1=OSD, 2=YUV

CX2341X_DEC_SCHED_DMA_FROM_HOST

Enum: 11/0x0B

Description

Setup DMA from host operation. Counterpart to API 0xCC

Param[0]

Memory address of link list

Param[1]

Total # of bytes to transfer

Param[2]

DMA type (0=MPEG, 1=OSD, 2=YUV)

CX2341X_DEC_PAUSE_PLAYBACK

Enum: 13/0x0D

Description

Freeze playback immediately. In this mode, when internal buffers are full, no more data will be accepted and data request IRQs will be masked.

Param[0]

Display: 0=last frame, 1=black

CX2341X_DEC_HALT_FW

Enum: 14/0x0E

Description

The firmware is halted and no further API calls are serviced until the firmware is uploaded again.

CX2341X_DEC_SET_STANDARD

Enum: 16/0x10

Description

Selects display standard

Param[0]

0=NTSC, 1=PAL

CX2341X_DEC_GET_VERSION

Enum: 17/0x11

Description

Returns decoder firmware version information

Result[0]**Version bitmask:**

- Bits 0:15 build
- Bits 16:23 minor
- Bits 24:31 major

CX2341X_DEC_SET_STREAM_INPUT

Enum: 20/0x14

Description

Select decoder stream input port

Param[0]

0=memory (default), 1=streaming

CX2341X_DEC_GET_TIMING_INFO

Enum: 21/0x15

Description

Returns timing information from start of playback

Result[0]

Frame count by decode order

Result[1]

Video PTS bits 0:31 by display order

Result[2]

Video PTS bit 32 by display order

Result[3]

SCR bits 0:31 by display order

Result[4]

SCR bit 32 by display order

CX2341X_DEC_SET_AUDIO_MODE

Enum: 22/0x16

Description

Select audio mode

Param[0]

Dual mono mode action 0=Stereo, 1=Left, 2=Right, 3=Mono, 4=Swap, -1=Unchanged

Param[1]

Stereo mode action: 0=Stereo, 1=Left, 2=Right, 3=Mono, 4=Swap, -1=Unchanged

CX2341X_DEC_SET_EVENT_NOTIFICATION

Enum: 23/0x17

Description

Setup firmware to notify the host about a particular event. Counterpart to API 0xD5

Param[0]**Event:**

- 0=Audio mode change between mono, (joint) stereo and dual channel.
- 3=Decoder started
- 4=Unknown: goes off 10-15 times per second while decoding.
- 5=Some sync event: goes off once per frame.

Param[1]

Notification 0=disabled, 1=enabled

Param[2]

Interrupt bit

Param[3]

Mailbox slot, -1 if no mailbox required.

CX2341X_DEC_SET_DISPLAY_BUFFERS

Enum: 24/0x18

Description

Number of display buffers. To decode all frames in reverse playback you must use nine buffers.

Param[0]

0=six buffers, 1=nine buffers

CX2341X_DEC_EXTRACT_VBI

Enum: 25/0x19

Description

Extracts VBI data

Param[0]

0=extract from extension & user data, 1=extract from private packets

Result[0]

VBI table location

Result[1]

VBI table size

CX2341X_DEC_SET_DECODER_SOURCE

Enum: 26/0x1A

Description

Selects decoder source. Ensure that the parameters passed to this API match the encoder settings.

Param[0]

Mode: 0=MPEG from host, 1=YUV from encoder, 2=YUV from host

Param[1]

YUV picture width

Param[2]

YUV picture height

Param[3]

Bitmap: see Param[0] of API 0xBD

CX2341X_DEC_SET_PREBUFFERING

Enum: 30/0x1E

Description

Decoder prebuffering, when enabled up to 128KB are buffered for streams <8mpbs or 640KB for streams >8mbps

Param[0]

0=off, 1=on

PVR350 Video decoder registers 0x02002800 -> 0x02002B00

Author: Ian Armstrong <ian@iarmst.demon.co.uk>

Version: v0.4

Date: 12 March 2007

This list has been worked out through trial and error. There will be mistakes and omissions. Some registers have no obvious effect so it's hard to say what they do, while others interact with each other, or require a certain load sequence. Horizontal filter setup is one example, with six registers working in unison and requiring a certain load sequence to correctly configure. The indexed colour palette is much easier to set at just two registers, but again it requires a certain load sequence.

Some registers are fussy about what they are set to. Load in a bad value & the decoder will fail. A firmware reload will often recover, but sometimes a reset is required. For registers containing size information, setting them to 0 is generally a bad idea. For other control registers i.e. 2878, you'll only find out what values are bad when it hangs.

```

-----
2800
bit 0
    Decoder enable
    0 = disable
    1 = enable
-----
2804
bits 0:31
    Decoder horizontal Y alias register 1
-----
2808
bits 0:31
    Decoder horizontal Y alias register 2

```

280C
bits 0:31
Decoder horizontal Y alias register 3

2810
bits 0:31
Decoder horizontal Y alias register 4

2814
bits 0:31
Decoder horizontal Y alias register 5

2818
bits 0:31
Decoder horizontal Y alias trigger

These six registers control the horizontal aliasing filter for the Y plane. The first five registers must all be loaded before accessing the trigger (2818), as this register actually clocks the data through for the first five.

To correctly program set the filter, this whole procedure must be done 16 times. The actual register contents are copied from a lookup-table in the firmware which contains 4 different filter settings.

281C
bits 0:31
Decoder horizontal UV alias register 1

2820
bits 0:31
Decoder horizontal UV alias register 2

2824
bits 0:31
Decoder horizontal UV alias register 3

2828
bits 0:31
Decoder horizontal UV alias register 4

282C
bits 0:31
Decoder horizontal UV alias register 5

2830
bits 0:31
Decoder horizontal UV alias trigger

These six registers control the horizontal aliasing for the UV plane. Operation is the same as the Y filter, with 2830 being the trigger register.

2834

bits 0:15
Decoder Y source width in pixels

bits 16:31
Decoder Y destination width in pixels

2838
bits 0:15
Decoder UV source width in pixels

bits 16:31
Decoder UV destination width in pixels

NOTE: For both registers, the resulting image must be fully visible on screen. If the image exceeds the right edge both the source and destination size must be adjusted to reflect the visible portion. For the source width, you must take into account the scaling when calculating the new value.

283C
bits 0:31
Decoder Y horizontal scaling
Normally = Reg 2854 >> 2

2840
bits 0:31
Decoder ?? unknown - horizontal scaling
Usually 0x00080514

2844
bits 0:31
Decoder UV horizontal scaling
Normally = Reg 2854 >> 2

2848
bits 0:31
Decoder ?? unknown - horizontal scaling
Usually 0x00100514

284C
bits 0:31
Decoder ?? unknown - Y plane
Usually 0x00200020

2850
bits 0:31
Decoder ?? unknown - UV plane
Usually 0x00200020

2854
bits 0:31
Decoder 'master' value for horizontal scaling

2858
bits 0:31
Decoder ?? unknown
Usually 0

285C
bits 0:31
 Decoder ?? unknown
 Normally = Reg 2854 >> 1

2860
bits 0:31
 Decoder ?? unknown
 Usually 0

2864
bits 0:31
 Decoder ?? unknown
 Normally = Reg 2854 >> 1

2868
bits 0:31
 Decoder ?? unknown
 Usually 0

Most of these registers either control horizontal scaling, or appear linked to it in some way. Register 2854 contains the 'master' value & the other registers can be calculated from that one. You must also remember to correctly set the divider in Reg 2874.

To enlarge:
 Reg 2854 = (source_width * 0x00200000) / destination_width
 Reg 2874 = No divide

To reduce from full size down to half size:
 Reg 2854 = (source_width/2 * 0x00200000) / destination width
 Reg 2874 = Divide by 2

To reduce from half size down to quarter size:
 Reg 2854 = (source_width/4 * 0x00200000) / destination width
 Reg 2874 = Divide by 4

The result is always rounded up.

286C
bits 0:15
 Decoder horizontal Y buffer offset

bits 15:31
 Decoder horizontal UV buffer offset

Offset into the video image buffer. If the offset is gradually incremented, the on screen image will move left & wrap around higher up on the right.

2870
bits 0:15
 Decoder horizontal Y output offset

bits 16:31

Decoder horizontal UV output offset

Offsets the actual video output. Controls output alignment of the Y & UV planes. The higher the value, the greater the shift to the left. Use reg 2890 to move the image right.

2874

bits 0:1

Decoder horizontal Y output size divider

00 = No divide

01 = Divide by 2

10 = Divide by 3

bits 4:5

Decoder horizontal UV output size divider

00 = No divide

01 = Divide by 2

10 = Divide by 3

bit 8

Decoder ?? unknown

0 = Normal

1 = Affects video output levels

bit 16

Decoder ?? unknown

0 = Normal

1 = Disable horizontal filter

2878

bit 0

?? unknown

bit 1

osd on/off

0 = osd off

1 = osd on

bit 2

Decoder + osd video timing

0 = NTSC

1 = PAL

bits 3:4

?? unknown

bit 5

Decoder + osd

Swaps upper & lower fields

287C

bits 0:10

Decoder & osd ?? unknown

Moves entire screen horizontally. Starts at 0x005 with the screen

shifted heavily to the right. Incrementing in steps of 0x004 will gradually shift the screen to the left.

bits 11:31
?? unknown

Normally contents are 0x00101111 (NTSC) or 0x1010111d (PAL)

2880 ----- ?? unknown
2884 ----- ?? unknown

2888
bit 0
Decoder + osd ?? unknown
0 = Normal
1 = Misaligned fields (Correctable through 289C & 28A4)

bit 4
?? unknown

bit 8
?? unknown

Warning: Bad values will require a firmware reload to recover.
Known to be bad are 0x000,0x011,0x100,0x111

288C
bits 0:15
osd ?? unknown
Appears to affect the osd position stability. The higher the value the more unstable it becomes. Decoder output remains stable.

bits 16:31
osd ?? unknown
Same as bits 0:15

2890
bits 0:11
Decoder output horizontal offset.

Horizontal offset moves the video image right. A small left shift is possible, but it's better to use reg 2870 for that due to its greater range.

NOTE: Video corruption will occur if video window is shifted off the right edge. To avoid this read the notes for 2834 & 2838.

2894
bits 0:23
Decoder output video surround colour.

Contains the colour (in yuv) used to fill the screen when the video is running in a window.

2898

bits 0:23

Decoder video window colour
Contains the colour (in yuv) used to fill the video window when the video is turned off.

bit 24

Decoder video output
0 = Video on
1 = Video off

bit 28

Decoder plane order
0 = Y,UV
1 = UV,Y

bit 29

Decoder second plane byte order
0 = Normal (UV)
1 = Swapped (VU)

In normal usage, the first plane is Y & the second plane is UV. Though the order of the planes can be swapped, only the byte order of the second plane can be swapped. This isn't much use for the Y plane, but can be useful for the UV plane.

289C

bits 0:15

Decoder vertical field offset 1

bits 16:31

Decoder vertical field offset 2

Controls field output vertical alignment. The higher the number, the lower the image on screen. Known starting values are 0x011E0017 (NTSC) & 0x01500017 (PAL)

28A0

bits 0:15

Decoder & osd width in pixels

bits 16:31

Decoder & osd height in pixels

All output from the decoder & osd are disabled beyond this area. Decoder output will simply go black outside of this region. If the osd tries to exceed this area it will become corrupt.

28A4

bits 0:11

osd left shift.

Has a range of 0x770->0x7FF. With the exception of 0, any value outside of this range corrupts the osd.

28A8

bits 0:15

```
osd vertical field offset 1
```

```
bits 16:31
```

```
osd vertical field offset 2
```

Controls field output vertical alignment. The higher the number, the lower the image on screen. Known starting values are 0x011E0017 (NTSC) & 0x01500017 (PAL)

```
-----
28AC ----- ?? unknown
```

```
|
V
```

```
28BC ----- ?? unknown
-----
```

```
28C0
```

```
bit 0
```

```
Current output field
```

```
0 = first field
```

```
1 = second field
```

```
bits 16:31
```

```
Current scanline
```

```
The scanline counts from the top line of the first field
through to the last line of the second field.
```

```
-----
28C4 ----- ?? unknown
```

```
|
V
```

```
28F8 ----- ?? unknown
-----
```

```
28FC
```

```
bit 0
```

```
?? unknown
```

```
0 = Normal
```

```
1 = Breaks decoder & osd output
```

```
-----
2900
```

```
bits 0:31
```

```
Decoder vertical Y alias register 1
```

```
-----
2904
```

```
bits 0:31
```

```
Decoder vertical Y alias register 2
```

```
-----
2908
```

```
bits 0:31
```

```
Decoder vertical Y alias trigger
```

These three registers control the vertical aliasing filter for the Y plane. Operation is similar to the horizontal Y filter (2804). The only real difference is that there are only two registers to set before accessing the trigger register (2908). As for the horizontal filter, the values are taken from a lookup table in the firmware, and the procedure must be repeated 16 times to fully program the filter.

```
-----
290C
```

```
bits 0:31
```

Decoder vertical UV alias register 1

2910

bits 0:31

Decoder vertical UV alias register 2

2914

bits 0:31

Decoder vertical UV alias trigger

These three registers control the vertical aliasing filter for the UV plane. Operation is the same as the Y filter, with 2914 being the trigger.

2918

bits 0:15

Decoder Y source height in pixels

bits 16:31

Decoder Y destination height in pixels

291C

bits 0:15

Decoder UV source height in pixels divided by 2

bits 16:31

Decoder UV destination height in pixels

NOTE: For both registers, the resulting image must be fully visible on screen. If the image exceeds the bottom edge both the source and destination size must be adjusted to reflect the visible portion. For the source height, you must take into account the scaling when calculating the new value.

2920

bits 0:31

Decoder Y vertical scaling
Normally = Reg 2930 >> 2

2924

bits 0:31

Decoder Y vertical scaling
Normally = Reg 2920 + 0x514

2928

bits 0:31

Decoder UV vertical scaling
When enlarging = Reg 2930 >> 2
When reducing = Reg 2930 >> 3

292C

bits 0:31

Decoder UV vertical scaling
Normally = Reg 2928 + 0x514

2930

bits 0:31

Decoder 'master' value for vertical scaling

2934
bits 0:31
Decoder ?? unknown - Y vertical scaling

2938
bits 0:31
Decoder Y vertical scaling
Normally = Reg 2930

293C
bits 0:31
Decoder ?? unknown - Y vertical scaling

2940
bits 0:31
Decoder UV vertical scaling
When enlarging = Reg 2930 >> 1
When reducing = Reg 2930

2944
bits 0:31
Decoder ?? unknown - UV vertical scaling

2948
bits 0:31
Decoder UV vertical scaling
Normally = Reg 2940

294C
bits 0:31
Decoder ?? unknown - UV vertical scaling

Most of these registers either control vertical scaling, or appear linked to it in some way. Register 2930 contains the 'master' value & all other registers can be calculated from that one. You must also remember to correctly set the divider in Reg 296C

To enlarge:
Reg 2930 = (source_height * 0x00200000) / destination_height
Reg 296C = No divide

To reduce from full size down to half size:
Reg 2930 = (source_height/2 * 0x00200000) / destination height
Reg 296C = Divide by 2

To reduce from half down to quarter.
Reg 2930 = (source_height/4 * 0x00200000) / destination height
Reg 296C = Divide by 4

2950
bits 0:15
Decoder Y line index into display buffer, first field

bits 16:31
Decoder Y vertical line skip, first field

 2954
 bits 0:15
 Decoder Y line index into display buffer, second field

bits 16:31
 Decoder Y vertical line skip, second field

2958
 bits 0:15
 Decoder UV line index into display buffer, first field

bits 16:31
 Decoder UV vertical line skip, first field

295C
 bits 0:15
 Decoder UV line index into display buffer, second field

bits 16:31
 Decoder UV vertical line skip, second field

2960
 bits 0:15
 Decoder destination height minus 1

bits 16:31
 Decoder destination height divided by 2

2964
 bits 0:15
 Decoder Y vertical offset, second field

bits 16:31
 Decoder Y vertical offset, first field

These two registers shift the Y plane up. The higher the number, the greater the shift.

2968
 bits 0:15
 Decoder UV vertical offset, second field

bits 16:31
 Decoder UV vertical offset, first field

These two registers shift the UV plane up. The higher the number, the greater the shift.

296C
 bits 0:1
 Decoder vertical Y output size divider
 00 = No divide
 01 = Divide by 2
 10 = Divide by 4

bits 8:9

Decoder vertical UV output size divider
00 = No divide
01 = Divide by 2
10 = Divide by 4

2970

bit 0

Decoder ?? unknown
0 = Normal
1 = Affect video output levels

bit 16

Decoder ?? unknown
0 = Normal
1 = Disable vertical filter

2974 ----- ?? unknown

|
V

29EF ----- ?? unknown

2A00

bits 0:2

osd colour mode
000 = 8 bit indexed
001 = 16 bit (565)
010 = 15 bit (555)
011 = 12 bit (444)
100 = 32 bit (8888)

bits 4:5

osd display bpp
01 = 8 bit
10 = 16 bit
11 = 32 bit

bit 8

osd global alpha
0 = Off
1 = On

bit 9

osd local alpha
0 = Off
1 = On

bit 10

osd colour key
0 = Off
1 = On

bit 11

osd ?? unknown
Must be 1

bit 13


```

osd colour space
0 = ARGB
1 = AYVU

```

```
bits 16:31
```

```

osd ?? unknown
Must be 0x001B (some kind of buffer pointer ?)

```

When the bits-per-pixel is set to 8, the colour mode is ignored and assumed to be 8 bit indexed. For 16 & 32 bits-per-pixel the colour depth is honoured, and when using a colour depth that requires fewer bytes than allocated the extra bytes are used as padding. So for a 32 bpp with 8 bit index colour, there are 3 padding bytes per pixel. It's also possible to select 16bpp with a 32 bit colour mode. This results in the pixel width being doubled, but the color key will not work as expected in this mode.

Colour key is as it suggests. You designate a colour which will become completely transparent. When using 565, 555 or 444 colour modes, the colour key is always 16 bits wide. The colour to key on is set in Reg 2A18.

Local alpha works differently depending on the colour mode. For 32bpp & 8 bit indexed, local alpha is a per-pixel 256 step transparency, with 0 being transparent and 255 being solid. For the 16bpp modes 555 & 444, the unused bit(s) act as a simple transparency switch, with 0 being solid & 1 being fully transparent. There is no local alpha support for 16bit 565.

Global alpha is a 256 step transparency that applies to the entire osd, with 0 being transparent & 255 being solid.

It's possible to combine colour key, local alpha & global alpha.

```
-----
2A04
```

```
bits 0:15
    osd x coord for left edge
```

```
bits 16:31
    osd y coord for top edge
-----
```

```
2A08
```

```
bits 0:15
    osd x coord for right edge
```

```
bits 16:31
    osd y coord for bottom edge
```

For both registers, (0,0) = top left corner of the display area. These registers do not control the osd size, only where it's positioned & how much is visible. The visible osd area cannot exceed the right edge of the display, otherwise the osd will become corrupt. See reg 2A10 for setting osd width.

```
-----
2A0C
```

```
bits 0:31
    osd buffer index
```

An index into the osd buffer. Slowly incrementing this moves the osd left, wrapping around onto the right edge

2A10
bits 0:11
 osd buffer 32 bit word width

Contains the width of the osd measured in 32 bit words. This means that all colour modes are restricted to a byte width which is divisible by 4.

2A14
bits 0:15
 osd height in pixels

bits 16:32
 osd line index into buffer
 osd will start displaying from this line.

2A18
bits 0:31
 osd colour key

Contains the colour value which will be transparent.

2A1C
bits 0:7
 osd global alpha

Contains the global alpha value (equiv ivtvfbctl --alpha XX)

2A20 ----- ?? unknown
|
V
2A2C ----- ?? unknown

2A30
bits 0:7
 osd colour to change in indexed palette

2A34
bits 0:31
 osd colour for indexed palette

To set the new palette, first load the index of the colour to change into 2A30, then load the new colour into 2A34. The full palette is 256 colours, so the index range is 0x00-0xFF

2A38 ----- ?? unknown
2A3C ----- ?? unknown

2A40
bits 0:31
 osd ?? unknown

Affects overall brightness, wrapping around to black

2A44
bits 0:31
 osd ?? unknown

Green tint

```
-----
2A48
bits 0:31
    osd ?? unknown
```

Red tint

```
-----
2A4C
bits 0:31
    osd ?? unknown
```

Affects overall brightness, wrapping around to black

```
-----
```

```
2A50
bits 0:31
    osd ?? unknown
```

Colour shift

```
-----
2A54
bits 0:31
    osd ?? unknown
```

Colour shift

```
-----
2A58 ----- ?? unknown
|
V
2AFC ----- ?? unknown
-----
```

```
2B00
bit 0
    osd filter control
    0 = filter off
    1 = filter on
```

```
bits 1:4
    osd ?? unknown
-----
```

The cx231xx DMA engine

This page describes the structures and procedures used by the cx2341x DMA engine.

Introduction

The cx2341x PCI interface is busmaster capable. This means it has a DMA engine to efficiently transfer large volumes of data between the card and main memory without requiring help from a CPU. Like most hardware, it must operate on contiguous physical memory. This is difficult to come by in large quantities on virtual memory machines.

Therefore, it also supports a technique called “scatter-gather”. The card can transfer multiple buffers in one operation. Instead of allocating one large contiguous buffer, the driver can allocate several smaller buffers.

In practice, I’ve seen the average transfer to be roughly 80K, but transfers above 128K were not uncommon, particularly at startup. The 128K figure is important, because that is the largest block that the kernel can normally allocate. Even still, 128K blocks are hard to come by, so the driver writer is urged to choose a smaller block size and learn the scatter-gather technique.

Mailbox #10 is reserved for DMA transfer information.

Note: the hardware expects little-endian data (‘intel format’).

Flow

This section describes, in general, the order of events when handling DMA transfers. Detailed information follows this section.

- The card raises the Encoder interrupt.
- The driver reads the transfer type, offset and size from Mailbox #10.
- The driver constructs the scatter-gather array from enough free dma buffers to cover the size.
- The driver schedules the DMA transfer via the ScheduleDMAtoHost API call.
- The card raises the DMA Complete interrupt.
- The driver checks the DMA status register for any errors.
- The driver post-processes the newly transferred buffers.

NOTE! It is possible that the Encoder and DMA Complete interrupts get raised simultaneously. (End of the last, start of the next, etc.)

Mailbox #10

The Flags, Command, Return Value and Timeout fields are ignored.

- Name: Mailbox #10
- Results[0]: Type: 0: MPEG.
- Results[1]: Offset: The position relative to the card’s memory space.
- Results[2]: Size: The exact number of bytes to transfer.

My speculation is that since the StartCapture API has a capture type of “RAW” available, that the type field will have other values that correspond to YUV and PCM data.

Scatter-Gather Array

The scatter-gather array is a contiguously allocated block of memory that tells the card the source and destination of each data-block to transfer. Card “addresses” are derived from the offset supplied by Mailbox #10. Host addresses are the physical memory location of the target DMA buffer.

Each S-G array element is a struct of three 32-bit words. The first word is the source address, the second is the destination address. Both take up the entire 32 bits. The lowest 18 bits of the third word is the transfer byte count. The high-bit of the third word is the “last” flag. The last-flag tells the card to raise the DMA_DONE interrupt. From hard personal experience, if you forget to set this bit, the card will still “work” but the stream will most likely get corrupted.

The transfer count must be a multiple of 256. Therefore, the driver will need to track how much data in the target buffer is valid and deal with it accordingly.

Array Element:

- 32-bit Source Address
- 32-bit Destination Address
- 14-bit reserved (high bit is the last flag)
- 18-bit byte count

DMA Transfer Status

Register 0x0004 holds the DMA Transfer Status:

- bit 0: read completed
- bit 1: write completed
- bit 2: DMA read error
- bit 3: DMA write error
- bit 4: Scatter-Gather array error

2.9.1.4 The cx88 driver

Author: Gerd Hoffmann

Documentation missing at the cx88 datasheet

MO_OUTPUT_FORMAT (0x310164)

Previous default from DScaler: 0x1c1f0008

Digit 8: 31-28

28: PREVEMOD = 1

Digit 7: 27-24 (0xc = 12 = b1100)

27: COMBALT = 1

26: PAL_INV_PHASE

```
(DScaler apparently set this to 1, resulted in sucky picture)

Digits 6,5: 23-16
25-16: COMB_RANGE = 0x1f [default] (9 bits -> max 512)

Digit 4: 15-12
15: DISIFX = 0
14: INVCBF = 0
13: DISADAPT = 0
12: NARROWADAPT = 0

Digit 3: 11-8
11: FORCE2H
10: FORCEREMD
9: NCHROMAEN
8: NREMODEN

Digit 2: 7-4
7-6: YCORE
5-4: CCORE

Digit 1: 3-0
3: RANGE = 1
2: HACTEXT
1: HSFMT
```

0x47 is the sync byte for MPEG-2 transport stream packets. Datasheet incorrectly states to use 47 decimal. 188 is the length. All DVB compliant frontends output packets with this start code.

Hauppauge WinTV cx88 IR information

The controls for the mux are GPIO [0,1] for source, and GPIO 2 for muting.

GPIO0	GPIO1	
0	0	TV Audio
1	0	FM radio
0	1	Line-In
1	1	Mono tuner bypass or CD passthru (tuner specific)

GPIO 16(I believe) is tied to the IR port (if present).

From the data sheet:

- Register 24'h20004 PCI Interrupt Status
- bit [18] IR_SMP_INT Set when 32 input samples have been collected over
- gpio[16] pin into GP_SAMPLE register.

What's missing from the data sheet:

- Setup 4KHz sampling rate (roughly 2x oversampled; good enough for our RC5 compat remote)
- set register 0x35C050 to 0xa80a80

- enable sampling
- set register 0x35C054 to 0x5
- enable the IRQ bit 18 in the interrupt mask register (and provide for a handler)

GP_SAMPLE register is at 0x35C058

Bits are then right shifted into the GP_SAMPLE register at the specified rate; you get an interrupt when a full DWORD is received. You need to recover the actual RC5 bits out of the (oversampled) IR sensor bits. (Hint: look for the 0/1 and 1/0 crossings of the RC5 bi-phase data) An actual raw RC5 code will span 2-3 DWORDS, depending on the actual alignment.

I'm pretty sure when no IR signal is present the receiver is always in a marking state(1); but stray light, etc can cause intermittent noise values as well. Remember, this is a free running sample of the IR receiver state over time, so don't assume any sample starts at any particular place.

Additional info

This data sheet (google search) seems to have a lovely description of the RC5 basics: http://www.atmel.com/dyn/resources/prod_documents/doc2817.pdf

This document has more data: <http://www.nenya.be/beor/electronics/rc5.htm>

This document has a how to decode a bi-phase data stream: http://www.ee.washington.edu/circuit_archive/text/ir_decode.txt

This document has still more info: <http://www.xs4all.nl/~sbp/knowledge/ir/rc5.htm>

2.9.1.5 The VPBE V4L2 driver design

File partitioning

V4L2 display device driver drivers/media/platform/davinci/vpbe_display.c
drivers/media/platform/davinci/vpbe_display.h

VPBE display controller drivers/media/platform/davinci/vpbe.c
drivers/media/platform/davinci/vpbe.h

VPBE venc sub device driver drivers/media/platform/davinci/vpbe_venc.c
drivers/media/platform/davinci/vpbe_venc.h drivers/media/platform/davinci/vpbe_venc_regs.h

VPBE osd driver drivers/media/platform/davinci/vpbe_osd.c
drivers/media/platform/davinci/vpbe_osd.h drivers/media/platform/davinci/vpbe_osd_regs.h

To be done

vpbe display controller

- Add support for external encoders.
- add support for selecting external encoder as default at probe time.

vpbe venc sub device

- add timings for supporting ths8200
- add support for LogicPD LCD.

FB drivers

- Add support for fbdev drivers.- Ready and part of subsequent patches.

2.9.1.6 The Samsung S5P/EXYNOS4 FIMC driver

Copyright © 2012 - 2013 Samsung Electronics Co., Ltd.

Files partitioning

- media device driver
drivers/media/platform/exynos4-is/media-dev.[ch]
- camera capture video device driver
drivers/media/platform/exynos4-is/fimc-capture.c
- MIPI-CSI2 receiver subdev
drivers/media/platform/exynos4-is/mipi-csis.[ch]
- video post-processor (mem-to-mem)
drivers/media/platform/exynos4-is/fimc-core.c
- common files
drivers/media/platform/exynos4-is/fimc-core.h drivers/media/platform/exynos4-is/fimc-reg.h
drivers/media/platform/exynos4-is/regs-fimc.h

2.9.1.7 The pvrusb2 driver

Author: Mike Isely <isely@pobox.com>

Background

This driver is intended for the “Hauppauge WinTV PVR USB 2.0”, which is a USB 2.0 hosted TV Tuner. This driver is a work in progress. Its history started with the reverse-engineering effort by Björn Danielsson <pvusb2@dax.nu> whose web page can be found here: <http://pvusb2.dax.nu/>

From there Aurelien Alleaume <slts@free.fr> began an effort to create a video4linux compatible driver. I began with Aurelien’s last known snapshot and evolved the driver to the state it is in here.

More information on this driver can be found at: <https://www.isely.net/pvusb2.html>

This driver has a strong separation of layers. They are very roughly:

1. Low level wire-protocol implementation with the device.
2. I2C adaptor implementation and corresponding I2C client drivers implemented elsewhere in V4L.
3. High level hardware driver implementation which coordinates all activities that ensure correct operation of the device.
4. A “context” layer which manages instancing of driver, setup, tear-down, arbitration, and interaction with high level interfaces appropriately as devices are hotplugged in the system.
5. High level interfaces which glue the driver to various published Linux APIs (V4L, sysfs, maybe DVB in the future).

The most important shearing layer is between the top 2 layers. A lot of work went into the driver to ensure that any kind of conceivable API can be laid on top of the core driver. (Yes, the driver internally leverages V4L to do its work but that really has nothing to do with the API published by the driver to the outside world.) The architecture allows for different APIs to simultaneously access the driver. I have a strong sense of fairness about APIs and also feel that it is a good design principle to keep implementation and interface isolated from each other. Thus while right now the V4L high level interface is the most complete, the sysfs high level interface will work equally well for similar functions, and there’s no reason I see right now why it shouldn’t be possible to produce a DVB high level interface that can sit right alongside V4L.

Building

To build these modules essentially amounts to just running “Make”, but you need the kernel source tree nearby and you will likely also want to set a few controlling environment variables first in order to link things up with that source tree. Please see the Makefile here for comments that explain how to do that.

Source file list / functional overview

(Note: The term “module” used below generally refers to loosely defined functional units within the pvrusb2 driver and bears no relation to the Linux kernel’s concept of a loadable module.)

pvrusb2-audio.[ch] - This is glue logic that resides between this driver and the msp3400.ko I2C client driver (which is found elsewhere in V4L).

pvrusb2-context.[ch] - This module implements the context for an instance of the driver. Everything else eventually ties back to or is otherwise instanced within the data structures implemented here. Hotplugging is ultimately coordinated here. All high level interfaces tie into the driver through this module. This module helps arbitrate each interface’s access to the actual driver core, and is designed to allow concurrent access through multiple instances of multiple interfaces (thus you can for example change the tuner’s frequency through sysfs while simultaneously streaming video through V4L out to an instance of mplayer).

pvrusb2-debug.h - This header defines a `printk()` wrapper and a mask of debugging bit definitions for the various kinds of debug messages that can be enabled within the driver.

pvrusb2-debugifc.[ch] - This module implements a crude command line oriented debug interface into the driver. Aside from being part of the process for implementing manual firmware extraction (see the pvrusb2 web site mentioned earlier), probably I’m the only one who has ever used this. It is mainly a debugging aid.

pvrusb2-eeeprom.[ch] - This is glue logic that resides between this driver the tveeprom.ko module, which is itself implemented elsewhere in V4L.

pvrusb2-encoder.[ch] - This module implements all protocol needed to interact with the Conexant mpeg2 encoder chip within the pvrusb2 device. It is a crude echo of corresponding logic in ivtv, however the design goals (strict isolation) and physical layer (proxy through USB instead of PCI) are enough different that this implementation had to be completely different.

pvrusb2-hdw-internal.h - This header defines the core data structure in the driver used to track ALL internal state related to control of the hardware. Nobody outside of the core hardware-handling modules should have any business using this header. All external access to the driver should be through one of the high level interfaces (e.g. V4L, sysfs, etc), and in fact even those high level interfaces are restricted to the API defined in pvrusb2-hdw.h and NOT this header.

pvrusb2-hdw.h - This header defines the full internal API for controlling the hardware. High level interfaces (e.g. V4L, sysfs) will work through here.

pvrusb2-hdw.c - This module implements all the various bits of logic that handle overall control of a specific pvrusb2 device. (Policy, instantiation, and arbitration of pvrusb2 devices fall within the jurisdiction of pvrusb2-context not here).

pvrusb2-i2c-chips-*.c - These modules implement the glue logic to tie together and configure various I2C modules as they attach to the I2C bus. There are two versions of this file. The “v4l2” version is intended to be used in-tree alongside V4L, where we implement just the logic that makes sense for a pure V4L environment. The “all” version is intended for use outside of V4L, where we might encounter other possibly “challenging” modules from ivtv or older kernel snapshots (or even the support modules in the standalone snapshot).

pvrusb2-i2c-cmd-v4l1.[ch] - This module implements generic V4L1 compatible commands to the I2C modules. It is here where state changes inside the pvrusb2 driver are translated into V4L1 commands that are in turn send to the various I2C modules.

pvrusb2-i2c-cmd-v4l2.[ch] - This module implements generic V4L2 compatible commands to the I2C modules. It is here where state changes inside the pvrusb2 driver are translated into V4L2 commands that are in turn send to the various I2C modules.

pvrusb2-i2c-core.[ch] - This module provides an implementation of a kernel-friendly I2C adaptor driver, through which other external I2C client drivers (e.g. msp3400, tuner, lirc) may connect and operate corresponding chips within the pvrusb2 device. It is through here that other V4L modules can reach into this driver to operate specific pieces (and those modules are in turn driven by glue logic which is coordinated by pvrusb2-hdw, doled out by pvrusb2-context, and then ultimately made available to users through one of the high level interfaces).

pvrusb2-io.[ch] - This module implements a very low level ring of transfer buffers, required in order to stream data from the device. This module is very low level. It only operates the buffers and makes no attempt to define any policy or mechanism for how such buffers might be used.

pvrusb2-ioread.[ch] - This module layers on top of pvrusb2-io.[ch] to provide a streaming API usable by a read() system call style of I/O. Right now this is the only layer on top of pvrusb2-io.[ch], however the underlying architecture here was intended to allow for other styles of I/O to be implemented with additional modules, like mmap()'ed buffers or something even more exotic.

pvrusb2-main.c - This is the top level of the driver. Module level and USB core entry points are here. This is our "main".

pvrusb2-sysfs.[ch] - This is the high level interface which ties the pvrusb2 driver into sysfs. Through this interface you can do everything with the driver except actually stream data.

pvrusb2-tuner.[ch] - This is glue logic that resides between this driver and the tuner.ko I2C client driver (which is found elsewhere in V4L).

pvrusb2-util.h - This header defines some common macros used throughout the driver. These macros are not really specific to the driver, but they had to go somewhere.

pvrusb2-v4l2.[ch] - This is the high level interface which ties the pvrusb2 driver into video4linux. It is through here that V4L applications can open and operate the driver in the usual V4L ways. Note that ALL V4L functionality is published only through here and nowhere else.

pvrusb2-video-*.ch] - This is glue logic that resides between this driver and the saa711x.ko I2C client driver (which is found elsewhere in V4L). Note that saa711x.ko used to be known as saa7115.ko in ivtv. There are two versions of this; one is selected depending on the particular saa711[5x].ko that is found.

pvrusb2.h - This header contains compile time tunable parameters (and at the moment the driver has very little that needs to be tuned).

2.9.1.8 PXA-Camera Host Driver

Author: Robert Jarzmik <robert.jarzmik@free.fr>

Constraints

- a) Image size for YUV422P format All YUV422P images are enforced to have width x height % 16 = 0. This is due to DMA constraints, which transfers only planes of 8 byte multiples.

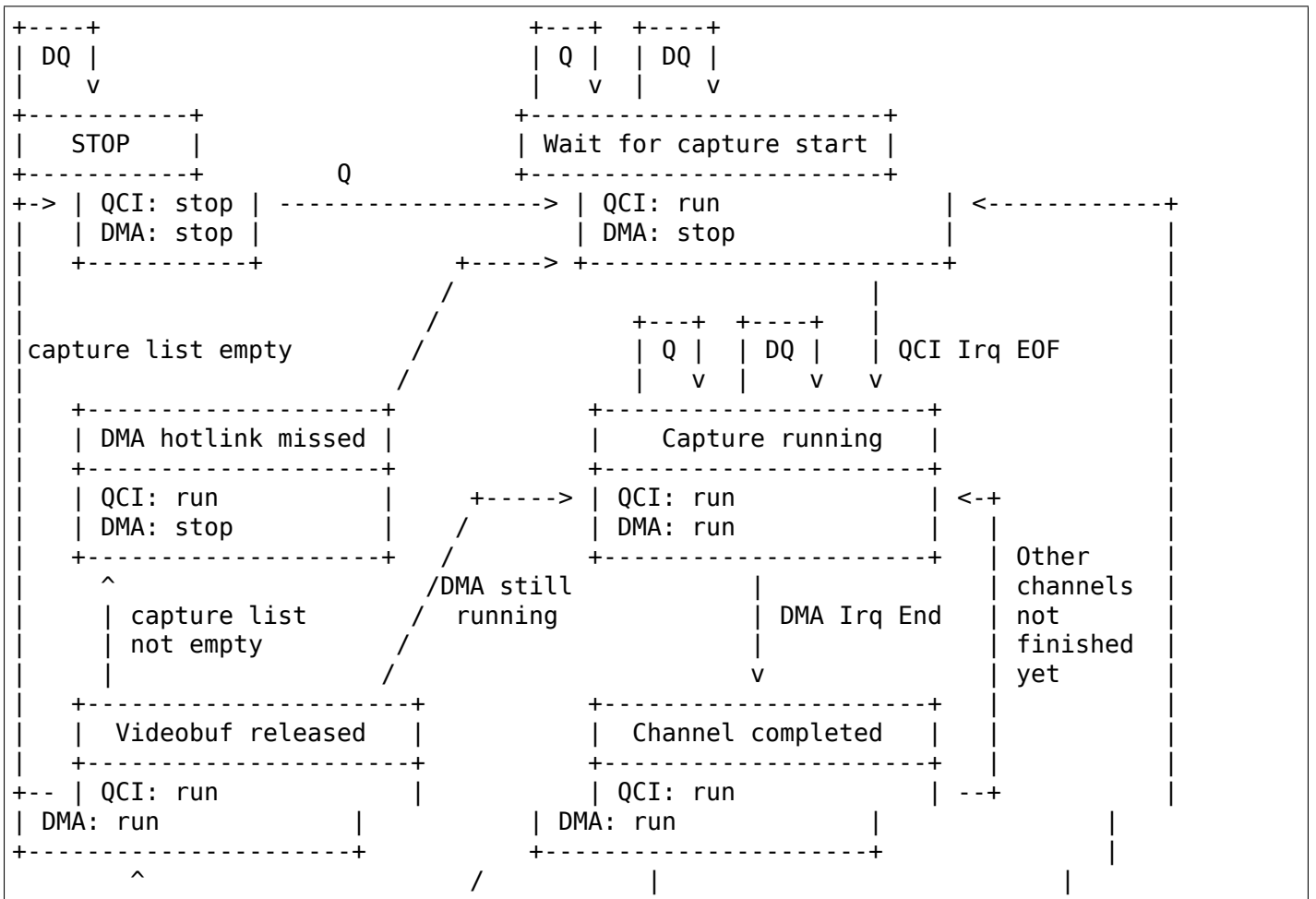
Global video workflow

- a) QCI stopped Initially, the QCI interface is stopped. When a buffer is queued (pxa_videobuf_ops->buf_queue), the QCI starts.
- b) QCI started More buffers can be queued while the QCI is started without halting the capture. The new buffers are "appended" at the tail of the DMA chain, and smoothly captured one frame after the other.

Once a buffer is filled in the QCI interface, it is marked as "DONE" and removed from the active buffers list. It can be then requeued or dequeued by userland application.

Once the last buffer is filled in, the QCI interface stops.

- c) Capture global finite state machine schema



Note: If DMA stops just after `pxa_camera_check_link_miss()` reads `DDADR()` value, we have the guarantee that the DMA irq handler will be called back when the DMA will finish the buffer, and `pxa_camera_check_link_miss()` will be called again, to reschedule `Videobuffer3`.

2.9.1.9 The Radiotrack radio driver

Author: Stephen M. Benoit <benoits@servicepro.com>

Date: Dec 14, 1996

ACKNOWLEDGMENTS

This document was made based on 'C' code for Linux from Gideon le Grange (legrang@active.co.za or legrang@cs.sun.ac.za) in 1994, and elaborations from Frans Brinkman (brinkman@esd.nl) in 1996. The results reported here are from experiments that the author performed on his own setup, so your mileage may vary... I make no guarantees, claims or warranties to the suitability or validity of this information. No other documentation on the AIMS Lab (<http://www.aimslab.com/>) RadioTrack card was made available to the author. This document is offered in the hopes that it might help users who want to use the RadioTrack card in an environment other than MS Windows.

WHY THIS DOCUMENT?

I have a RadioTrack card from back when I ran an MS-Windows platform. After converting to Linux, I found Gideon le Grange's command-line software for running the card, and found that it was good! Frans Brinkman made a comfortable X-windows interface, and added a scanning feature. For hack value, I wanted to see if the tuner could be tuned beyond the usual FM radio broadcast band, so I could pick up the audio carriers from North American broadcast TV channels, situated just below and above the 87.0-109.0 MHz range. I did not get much success, but I learned about programming ioports under Linux and gained some insights about the hardware design used for the card.

So, without further delay, here are the details.

PHYSICAL DESCRIPTION

The RadioTrack card is an ISA 8-bit FM radio card. The radio frequency (RF) input is simply an antenna lead, and the output is a power audio signal available through a miniature phone plug. Its RF frequencies of operation are more or less limited from 87.0 to 109.0 MHz (the commercial FM broadcast band). Although the registers can be programmed to request frequencies beyond these limits, experiments did not give promising results. The variable frequency oscillator (VFO) that demodulates the intermediate frequency (IF) signal probably has a small range of useful frequencies, and wraps around or gets clipped beyond the limits mentioned above.

CONTROLLING THE CARD WITH IOPORT

The RadioTrack (base) ioport is configurable for 0x30c or 0x20c. Only one ioport seems to be involved. The ioport decoding circuitry must be pretty simple, as individual ioport bits are directly matched to specific functions (or blocks) of the radio card. This way, many functions can be changed in parallel with one write to the ioport. The only feedback available through the ioport appears to be the “Stereo Detect” bit.

The bits of the ioport are arranged as follows:

MSb				LSb			
VolA	VolB	????	Stereo	Radio	TuneA	TuneB	Tune
(+)	(-)		Detect	Audio	(bit)	(latch)	Update
			Enable	Enable			Enable

VolA	VolB	Description
0	0	audio mute
0	1	volume + (some delay required)
1	0	volume - (some delay required)
1	1	stay at present volume

Stereo Detect Enable	Description
0	No Detect
1	Detect

Results available by reading ioport >60 msec after last port write.

0xff ==> no stereo detected, 0xfd ==> stereo detected.

Radio to Audio (path) Enable	Description
0	Disable path (silence)
1	Enable path (audio produced)

TuneA	TuneB	Description
0	0	“zero” bit phase 1
0	1	“zero” bit phase 2
1	0	“one” bit phase 1
1	1	“one” bit phase 2

24-bit code, where bits = (freq*40) + 10486188. The Most Significant 11 bits must be 1010 xxxx 0x0 to be valid. The bits are shifted in LSb first.

Tune Update Enable	Description
0	Tuner held constant
1	Tuner updating in progress

PROGRAMMING EXAMPLES

```

Default:      BASE <-- 0xc8 (current volume, no stereo detect,
                    radio enable, tuner adjust disable)

Card Off:     BASE <-- 0x00 (audio mute, no stereo detect,
                    radio disable, tuner adjust disable)

Card On:      BASE <-- 0x00 (see "Card Off", clears any unfinished business)
                    BASE <-- 0xc8 (see "Default")

Volume Down:  BASE <-- 0x48 (volume down, no stereo detect,
                    radio enable, tuner adjust disable)
                    wait 10 msec
                    BASE <-- 0xc8 (see "Default")

Volume Up:    BASE <-- 0x88 (volume up, no stereo detect,
                    radio enable, tuner adjust disable)
                    wait 10 msec
                    BASE <-- 0xc8 (see "Default")

Check Stereo: BASE <-- 0xd8 (current volume, stereo detect,
                    radio enable, tuner adjust disable)
                    wait 100 msec
                    x <-- BASE (read ioport)
                    BASE <-- 0xc8 (see "Default")

                    x=0xff ==> "not stereo", x=0xfd ==> "stereo detected"

Set Frequency: code = (freq*40) + 10486188
                    foreach of the 24 bits in code,
                    (from Least to Most Significant):
                    to write a "zero" bit,
                    BASE <-- 0x01 (audio mute, no stereo detect, radio
                                disable, "zero" bit phase 1, tuner adjust)
                    BASE <-- 0x03 (audio mute, no stereo detect, radio
                                disable, "zero" bit phase 2, tuner adjust)
                    to write a "one" bit,
                    BASE <-- 0x05 (audio mute, no stereo detect, radio
                                disable, "one" bit phase 1, tuner adjust)
                    BASE <-- 0x07 (audio mute, no stereo detect, radio
                                disable, "one" bit phase 2, tuner adjust)

```

2.9.1.10 The Rockchip Image Signal Processor Driver (rkisp1)

Versions and their differences

The rkisp1 block underwent some changes between SoC implementations. The vendor designates them as:

- V10: used at least in rk3288 and rk3399
- V11: declared in the original vendor code, but not used
- V12: used at least in rk3326 and px30

- V13: used at least in rk1808
- V20: used in rk3568 and beyond

Right now the kernel supports rkisp1 implementations based on V10 and V12 variants. V11 does not seem to be actually used and V13 will need some more additions but isn't researched yet, especially as it seems to be limited to the rk1808 which hasn't reached much market spread.

V20 on the other hand will probably be used in future SoCs and has seen really big changes in the vendor kernel, so will need quite a bit of research.

Changes from V10 to V12

- V12 supports a new CSI-host implementation but can still also use the same implementation from V10
- The module for lens shading correction got changed from 12bit to 13bit width
- The AWB and AEC modules got replaced to support finer grained data collection

Changes from V12 to V13

The list for V13 is incomplete and needs further investigation.

- V13 does not support the old CSI-host implementation anymore

2.9.1.11 The saa7134 driver

Author Gerd Hoffmann

Card Variations:

Cards can use either of these two crystals (xtal):

- 32.11 MHz -> `.audio_clock=0x187de7`
- 24.576MHz -> `.audio_clock=0x200000 (xtal * .audio_clock = 51539600)`

Some details about 30/34/35:

- saa7130 - low-price chip, doesn't have mute, that is why all those cards should have `.mute` field defined in their tuner structure.
- saa7134 - usual chip
- saa7133/35 - saa7135 is probably a marketing decision, since all those chips identifies itself as 33 on pci.

LifeView GPIOs

This section was authored by: Peter Missel <peter.missel@onlinehome.de>

- LifeView FlyTV Platinum FM (LR214WF)
 - GP27 MDT2005 PB4 pin 10
 - GP26 MDT2005 PB3 pin 9
 - GP25 MDT2005 PB2 pin 8
 - GP23 MDT2005 PB1 pin 7
 - GP22 MDT2005 PB0 pin 6
 - GP21 MDT2005 PB5 pin 11
 - GP20 MDT2005 PB6 pin 12
 - GP19 MDT2005 PB7 pin 13
 - nc MDT2005 PA3 pin 2
 - Remote MDT2005 PA2 pin 1
 - GP18 MDT2005 PA1 pin 18
 - nc MDT2005 PA0 pin 17 strap low
 - GP17 Strap "GP7"=High
 - GP16 Strap "GP6"=High
 - * 0=Radio 1=TV
 - * Drives SA630D ENCH1 and HEF4052 A1 pinsto do FM radio through SIF input
 - GP15 nc
 - GP14 nc
 - GP13 nc
 - GP12 Strap "GP5" = High
 - GP11 Strap "GP4" = High
 - GP10 Strap "GP3" = High
 - GP09 Strap "GP2" = Low
 - GP08 Strap "GP1" = Low
 - GP07.00 nc

Credits

andrew.stevens@philips.com + werner.leebe@philips.com for providing saa7134 hardware specs and sample board.

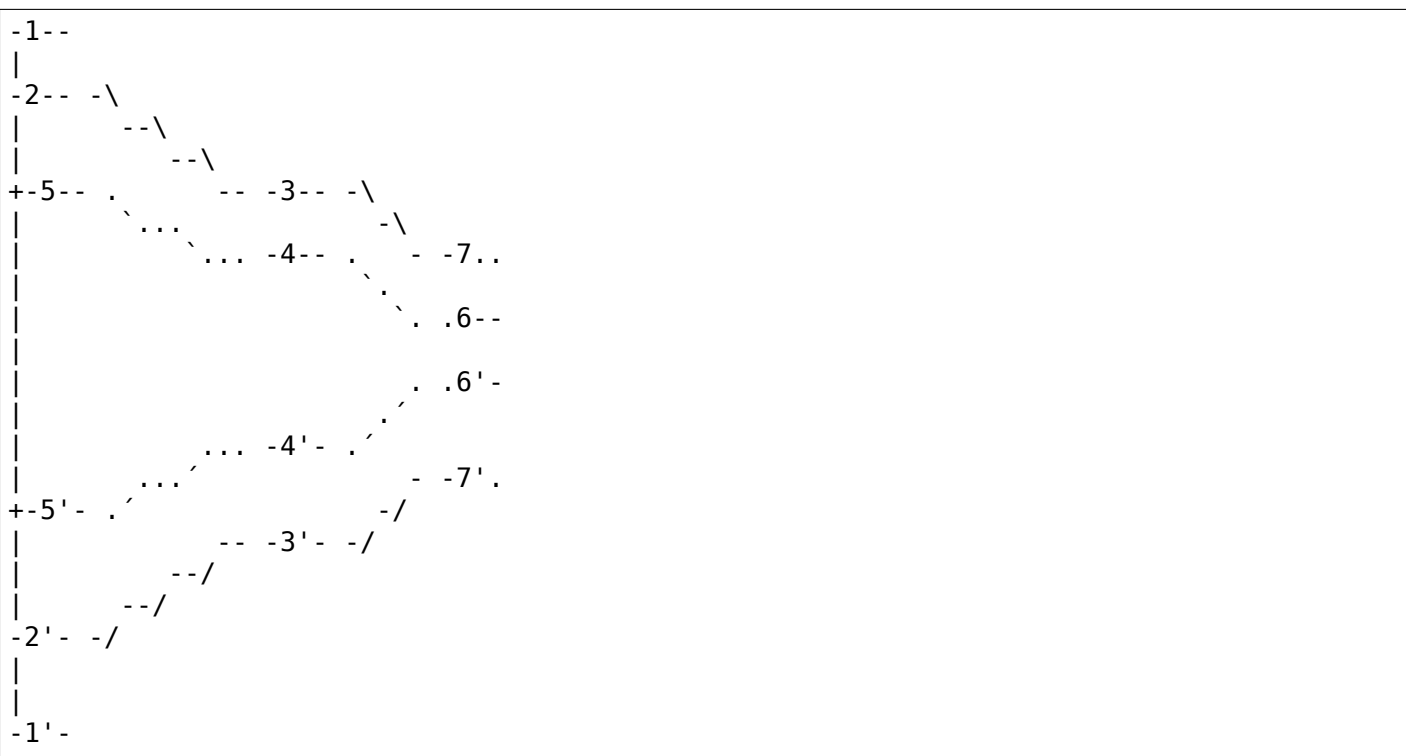
2.9.1.12 Cropping and Scaling algorithm, used in the sh_mobile_ceu_camera driver

Author: Guennadi Liakhovetski <g.liakhovetski@gmx.de>

Terminology

sensor scales: horizontal and vertical scales, configured by the sensor driver host scales: -"-
 host driver combined scales: sensor_scale * host_scale

Generic scaling / cropping scheme



In the above chart minuses and slashes represent “real” data amounts, points and accents represent “useful” data, basically, CEU scaled and cropped output, mapped back onto the client’s source plane.

Such a configuration can be produced by user requests:

S_CROP(left / top = (5) - (1), width / height = (5') - (5)) S_FMT(width / height = (6') - (6))

Here:

- (1) to (1') - whole max width or height
- (1) to (2) - sensor cropped left or top
- (2) to (2') - sensor cropped width or height
- (3) to (3') - sensor scale
- (3) to (4) - CEU cropped left or top
- (4) to (4') - CEU cropped width or height
- (5) to (5') - reverse sensor scale applied to CEU cropped width

or height (2) to (5) - reverse sensor scale applied to CEU cropped left or top (6) to (6') - CEU scale - user window

S_FMT

Do not touch input rectangle - it is already optimal.

1. Calculate current sensor scales:

$$\text{scale}_s = ((2') - (2)) / ((3') - (3))$$

2. Calculate "effective" input crop (sensor subwindow) - CEU crop scaled back at current sensor scales onto input window - this is user S_CROP:

$$\text{width}_u = (5') - (5) = ((4') - (4)) * \text{scale}_s$$

3. Calculate new combined scales from "effective" input window to requested user window:

$$\text{scale}_{\text{comb}} = \text{width}_u / ((6') - (6))$$

4. Calculate sensor output window by applying combined scales to real input window:

$$\text{width}_{s_out} = ((7') - (7)) = ((2') - (2)) / \text{scale}_{\text{comb}}$$

5. Apply iterative sensor S_FMT for sensor output window.

$$\text{subdev->video_ops->s_fmt}(\text{.width} = \text{width}_{s_out})$$

6. Retrieve sensor output window (g_fmt)

7. Calculate new sensor scales:

$$\text{scale}_{s_new} = ((3')_{\text{new}} - (3)_{\text{new}}) / ((2') - (2))$$

8. Calculate new CEU crop - apply sensor scales to previously calculated "effective" crop:

$$\begin{aligned} \text{width}_{\text{ceu}} &= (4')_{\text{new}} - (4)_{\text{new}} = \text{width}_u / \text{scale}_{s_new} \\ \text{left}_{\text{ceu}} &= (4)_{\text{new}} - (3)_{\text{new}} \\ &= ((5) - (2)) / \text{scale}_{s_new} \end{aligned}$$

9. Use CEU cropping to crop to the new window:

$$\text{ceu_crop}(\text{.width} = \text{width}_{\text{ceu}}, \text{.left} = \text{left}_{\text{ceu}})$$

10. Use CEU scaling to scale to the requested user window:

$$\text{scale}_{\text{ceu}} = \text{width}_{\text{ceu}} / \text{width}$$

S_CROP

The *V4L2 crop API* says:

"...specification does not define an origin or units. However by convention drivers should horizontally count unscaled samples relative to 0H."

We choose to follow the advise and interpret cropping units as client input pixels.

Cropping is performed in the following 6 steps:

1. Request exactly user rectangle from the sensor.

2. If smaller - iterate until a larger one is obtained. Result: sensor cropped to 2 : 2', target crop 5 : 5', current output format 6' - 6.
3. In the previous step the sensor has tried to preserve its output frame as good as possible, but it could have changed. Retrieve it again.
4. Sensor scaled to 3 : 3'. Sensor's scale is $(2' - 2) / (3' - 3)$. Calculate intermediate window: $4' - 4 = (5' - 5) * (3' - 3) / (2' - 2)$
5. Calculate and apply host scale = $(6' - 6) / (4' - 4)$
6. Calculate and apply host crop: $6 - 7 = (5 - 2) * (6' - 6) / (5' - 5)$

2.9.1.13 Tuner drivers

Simple tuner Programming

There are some flavors of Tuner programming APIs. These differ mainly by the bandswitch byte.

- L= LG_API (VHF_LO=0x01, VHF_HI=0x02, UHF=0x08, radio=0x04)
- P= PHILIPS_API (VHF_LO=0xA0, VHF_HI=0x90, UHF=0x30, radio=0x04)
- T= TEMIC_API (VHF_LO=0x02, VHF_HI=0x04, UHF=0x01)
- A= ALPS_API (VHF_LO=0x14, VHF_HI=0x12, UHF=0x11)
- M= PHILIPS_MK3 (VHF_LO=0x01, VHF_HI=0x02, UHF=0x04, radio=0x19)

Tuner Manufacturers

- Samsung Tuner identification: (e.g. TCPM9091PD27)

```
TCP [ABCJLMNQ] 90[89][125] [DP] [ACD] 27 [ABCD]
[ABCJLMNQ]:
A= BG+DK
B= BG
C= I+DK
J= NTSC-Japan
L= Secam LL
M= BG+I+DK
N= NTSC
Q= BG+I+DK+LL
[89]: ?
[125]:
2: No FM
5: With FM
[DP]:
D= NTSC
P= PAL
[ACD]:
A= F-connector
C= Phono connector
D= Din Jack
[ABCD]:
3-wire/I2C tuning, 2-band/3-band
```

These Tuners are PHILIPS_API compatible.

Philips Tuner identification: (e.g. FM1216MF)

```
F[IRMQ]12[1345]6{MF|ME|MP}
F[IRMQ]:
  FI12x6: Tuner Series
  FR12x6: Tuner + Radio IF
  FM12x6: Tuner + FM
  FQ12x6: special
  FMR12x6: special
  TD15xx: Digital Tuner ATSC
12[1345]6:
  1216: PAL BG
  1236: NTSC
  1246: PAL I
  1256: Pal DK
{MF|ME|MP}
MF: BG LL w/ Secam (Multi France)
ME: BG DK I LL (Multi Europe)
MP: BG DK I (Multi PAL)
MR: BG DK M (?)
MG: BG DKI M (?)
MK2 series PHILIPS_API, most tuners are compatible to this one !
MK3 series introduced in 2002 w/ PHILIPS_MK3_API
```

Temec Tuner identification: (.e.g 4006FH5)

```
4[01][0136][269]F[HYNR]5
  40x2: Tuner (5V/33V), TEMIC_API.
  40x6: Tuner 5V
  41xx: Tuner compact
  40x9: Tuner+FM compact
[0136]
  xx0x: PAL BG
  xx1x: Pal DK, Secam LL
  xx3x: NTSC
  xx6x: PAL I
F[HYNR]5
  FH5: Pal BG
  FY5: others
  FN5: multistandard
  FR5: w/ FM radio
3X xxxx: order number with specific connector
Note: Only 40x2 series has TEMIC_API, all newer tuners have PHILIPS_API.
```

LG Innotek Tuner:

- TPI8NSR11 : NTSC J/M (TPI8NSR01 w/FM) (P,210/497)
- TPI8PSB11 : PAL B/G (TPI8PSB01 w/FM) (P,170/450)
- TAPC-I701 : PAL I (TAPC-I001 w/FM) (P,170/450)
- TPI8PSB12 : PAL D/K+B/G (TPI8PSB02 w/FM) (P,170/450)
- TAPC-H701P: NTSC_JP (TAPC-H001P w/FM) (L,170/450)

- TAPC-G701P: PAL B/G (TAPC-G001P w/FM) (L,170/450)
- TAPC-W701P: PAL I (TAPC-W001P w/FM) (L,170/450)
- TAPC-Q703P: PAL D/K (TAPC-Q001P w/FM) (L,170/450)
- TAPC-Q704P: PAL D/K+I (L,170/450)
- TAPC-G702P: PAL D/K+B/G (L,170/450)
- TADC-H002F: NTSC (L,175/410?; 2-B, C-W+11, W+12-69)
- TADC-M201D: PAL D/K+B/G+I (L,143/425) (sound control at I2C address 0xc8)
- TADC-T003F: NTSC Taiwan (L,175/410?; 2-B, C-W+11, W+12-69)

Suffix:

- P= Standard phono female socket
- D= IEC female socket
- F= F-connector

Other Tuners:

- TCL2002MB-1 : PAL BG + DK =TUNER_LG_PAL_NEW_TAPC
- TCL2002MB-1F: PAL BG + DK w/FM =PHILIPS_PAL
- TCL2002MI-2 : PAL I = ??

ALPS Tuners:

- Most are LG_API compatible
- TSCH6 has ALPS_API (TSCH5 ?)
- TSBE1 has extra API 05,02,08 Control_byte=0xCB Source:¹

2.9.1.14 The Virtual Media Controller Driver (vimc)

Source code documentation

vimc-streamer

struct `vimc_stream`

struct that represents a stream in the pipeline

Definition

```
struct vimc_stream {
    struct media_pipeline pipe;
    struct vimc_ent_device *ved_pipeline[VIMC_STREAMER_PIPELINE_MAX_SIZE];
    unsigned int pipe_size;
    struct task_struct *kthread;
};
```

Members

¹ conexant100029b-PCI-Decoder-ApplicationNote.pdf

pipe the media pipeline object associated with this stream

ved_pipeline array containing all the entities participating in the stream. The order is from a video device (usually a capture device) where `stream_on` was called, to the entity generating the first base image to be processed in the pipeline.

pipe_size size of **ved_pipeline**

kthread thread that generates the frames of the stream.

Description

When the user call `stream_on` in a video device, *struct vimc_stream* is used to keep track of all entities and subdevices that generates and process frames for the stream.

```
struct media_entity * vimc_get_source_entity(struct media_entity *ent)
    get the entity connected with the first sink pad
```

Parameters

struct media_entity *ent reference media_entity

Description

Helper function that returns the media entity containing the source pad linked with the first sink pad from the given media entity pad list.

Return

The source pad or NULL, if it wasn't found.

```
void vimc_streamer_pipeline_terminate(struct vimc_stream *stream)
    Disable stream in all ved in stream
```

Parameters

struct vimc_stream *stream the pointer to the stream structure with the pipeline to be disabled.

Description

Calls `s_stream` to disable the stream in each entity of the pipeline

```
int vimc_streamer_pipeline_init(struct vimc_stream *stream, struct
                               vimc_ent_device *ved)
    Initializes the stream structure
```

Parameters

struct vimc_stream *stream the pointer to the stream structure to be initialized

struct vimc_ent_device *ved the pointer to the vimc entity initializing the stream

Description

Initializes the stream structure. Walks through the entity graph to construct the pipeline used later on the streamer thread. Calls `vimc_streamer_s_stream()` to enable stream in all entities of the pipeline.

Return

0 if success, error code otherwise.

int **vimc_streamer_thread**(void **data*)
Process frames through the pipeline

Parameters

void *data vimc_stream struct of the current stream

Description

From the source to the sink, gets a frame from each subdevice and send to the next one of the pipeline at a fixed framerate.

Return

Always zero (created as int instead of void to comply with kthread API).

int **vimc_streamer_s_stream**(struct *vimc_stream* **stream*, struct vimc_ent_device **ved*,
int *enable*)
Start/stop the streaming on the media pipeline

Parameters

struct vimc_stream **stream* the pointer to the stream structure of the current stream

struct vimc_ent_device **ved* pointer to the vimc entity of the entity of the stream

int enable flag to determine if stream should start/stop

Description

When starting, check if there is no stream->kthread allocated. This should indicate that a stream is already running. Then, it initializes the pipeline, creates and runs a kthread to consume buffers through the pipeline. When stopping, analogously check if there is a stream running, stop the thread and terminates the pipeline.

Return

0 if success, error code otherwise.

2.9.1.15 The Zoran driver

unified zoran driver (zr360x7, zoran, buz, dc10(+), dc30(+), lml33)

website: <http://mjpeg.sourceforge.net/driver-zoran/>

Frequently Asked Questions

What cards are supported

Iomega Buz, Linux Media Labs LML33/LML33R10, Pinnacle/Miro DC10/DC10+/DC30/DC30+ and related boards (available under various names).

Iomega Buz

- Zoran zr36067 PCI controller
- Zoran zr36060 MJPEG codec
- Philips saa7111 TV decoder
- Philips saa7185 TV encoder

Drivers to use: videodev, i2c-core, i2c-algo-bit, videocodec, saa7111, saa7185, zr36060, zr36067

Inputs/outputs: Composite and S-video

Norms: PAL, SECAM (720x576 @ 25 fps), NTSC (720x480 @ 29.97 fps)

Card number: 7

AverMedia 6 Eyes AVS6EYES

- Zoran zr36067 PCI controller
- Zoran zr36060 MJPEG codec
- Samsung ks0127 TV decoder
- Conexant bt866 TV encoder

Drivers to use: videodev, i2c-core, i2c-algo-bit, videocodec, ks0127, bt866, zr36060, zr36067

Inputs/outputs: Six physical inputs. 1-6 are composite, 1-2, 3-4, 5-6 doubles as S-video, 1-3 triples as component. One composite output.

Norms: PAL, SECAM (720x576 @ 25 fps), NTSC (720x480 @ 29.97 fps)

Card number: 8

Note: Not autodetected, card=8 is necessary.

Linux Media Labs LML33

- Zoran zr36067 PCI controller
- Zoran zr36060 MJPEG codec
- Brooktree bt819 TV decoder
- Brooktree bt856 TV encoder

Drivers to use: videodev, i2c-core, i2c-algo-bit, videocodec, bt819, bt856, zr36060, zr36067

Inputs/outputs: Composite and S-video

Norms: PAL (720x576 @ 25 fps), NTSC (720x480 @ 29.97 fps)

Card number: 5

Linux Media Labs LML33R10

- Zoran zr36067 PCI controller
- Zoran zr36060 MJPEG codec
- Philips saa7114 TV decoder
- Analog Devices adv7170 TV encoder

Drivers to use: videodev, i2c-core, i2c-algo-bit, videocodec, saa7114, adv7170, zr36060, zr36067

Inputs/outputs: Composite and S-video

Norms: PAL (720x576 @ 25 fps), NTSC (720x480 @ 29.97 fps)

Card number: 6

Pinnacle/Miro DC10(new)

- Zoran zr36057 PCI controller
- Zoran zr36060 MJPEG codec
- Philips saa7110a TV decoder
- Analog Devices adv7176 TV encoder

Drivers to use: videodev, i2c-core, i2c-algo-bit, videocodec, saa7110, adv7175, zr36060, zr36067

Inputs/outputs: Composite, S-video and Internal

Norms: PAL, SECAM (768x576 @ 25 fps), NTSC (640x480 @ 29.97 fps)

Card number: 1

Pinnacle/Miro DC10+

- Zoran zr36067 PCI controller
- Zoran zr36060 MJPEG codec
- Philips saa7110a TV decoder
- Analog Devices adv7176 TV encoder

Drivers to use: videodev, i2c-core, i2c-algo-bit, videocodec, saa7110, adv7175, zr36060, zr36067

Inputs/outputs: Composite, S-video and Internal

Norms: PAL, SECAM (768x576 @ 25 fps), NTSC (640x480 @ 29.97 fps)

Card number: 2

Pinnacle/Miro DC10(old)

- Zoran zr36057 PCI controller
- Zoran zr36050 MJPEG codec
- Zoran zr36016 Video Front End or Fuji md0211 Video Front End (clone?)
- Micronas vpx3220a TV decoder
- mse3000 TV encoder or Analog Devices adv7176 TV encoder

Drivers to use: videodev, i2c-core, i2c-algo-bit, videocodec, vpx3220, mse3000/adv7175, zr36050, zr36016, zr36067

Inputs/outputs: Composite, S-video and Internal

Norms: PAL, SECAM (768x576 @ 25 fps), NTSC (640x480 @ 29.97 fps)

Card number: 0

Pinnacle/Miro DC30

- Zoran zr36057 PCI controller
- Zoran zr36050 MJPEG codec
- Zoran zr36016 Video Front End
- Micronas vpx3225d/vpx3220a/vpx3216b TV decoder
- Analog Devices adv7176 TV encoder

Drivers to use: videodev, i2c-core, i2c-algo-bit, videocodec, vpx3220/vpx3224, adv7175, zr36050, zr36016, zr36067

Inputs/outputs: Composite, S-video and Internal

Norms: PAL, SECAM (768x576 @ 25 fps), NTSC (640x480 @ 29.97 fps)

Card number: 3

Pinnacle/Miro DC30+

- Zoran zr36067 PCI controller
- Zoran zr36050 MJPEG codec
- Zoran zr36016 Video Front End
- Micronas vpx3225d/vpx3220a/vpx3216b TV decoder
- Analog Devices adv7176 TV encoder

Drivers to use: videodev, i2c-core, i2c-algo-bit, videocodec, vpx3220/vpx3224, adv7175, zr36050, zr36015, zr36067

Inputs/outputs: Composite, S-video and Internal

Norms: PAL, SECAM (768x576 @ 25 fps), NTSC (640x480 @ 29.97 fps)

Card number: 4

Note:

- 1) No module for the mse3000 is available yet
 - 2) No module for the vpx3224 is available yet
-

1.1 What the TV decoder can do and what not

The best known TV standards are NTSC/PAL/SECAM. but for decoding a frame that information is not enough. There are several formats of the TV standards. And not every TV decoder is able to handle every format. Also the every combination is supported by the driver. There are currently 11 different tv broadcast formats all over the world.

The CCIR defines parameters needed for broadcasting the signal. The CCIR has defined different standards: A,B,D,E,F,G,H,I,K,K1,L,M,N,... The CCIR says not much about the colorsystem used !!! And talking about a colorsystem says not much about how it is broadcast.

The CCIR standards A,E,F are not used any more.

When you speak about NTSC, you usually mean the standard: CCIR - M using the NTSC colorsystem which is used in the USA, Japan, Mexico, Canada and a few others.

When you talk about PAL, you usually mean: CCIR - B/G using the PAL colorsystem which is used in many Countries.

When you talk about SECAM, you mean: CCIR - L using the SECAM Colorsystem which is used in France, and a few others.

There the other version of SECAM, CCIR - D/K is used in Bulgaria, China, Slovakia, Hungary, Korea (Rep.), Poland, Rumania and a others.

The CCIR - H uses the PAL colorsystem (sometimes SECAM) and is used in Egypt, Libya, Sri Lanka, Syrian Arab. Rep.

The CCIR - I uses the PAL colorsystem, and is used in Great Britain, Hong Kong, Ireland, Nigeria, South Africa.

The CCIR - N uses the PAL colorsystem and PAL frame size but the NTSC framerate, and is used in Argentina, Uruguay, and a few others

We do not talk about how the audio is broadcast !

A rather good sites about the TV standards are: <http://www.sony.jp/support/> http://info.electronicwerkstatt.de/bereiche/fernsehtechnik/frequenzen_und_normen/Fernsehnormen/ and <http://www.cabl.com/restaurant/channel.html>

Other weird things around: NTSC 4.43 is a modified NTSC, which is mainly used in PAL VCR's that are able to play back NTSC. PAL 60 seems to be the same as NTSC 4.43 . The Datasheets also talk about NTSC 44, It seems as if it would be the same as NTSC 4.43. NTSC Comb seems to be a decoder mode where the decoder uses a comb filter to split chroma and luma instead of a Delay line.

But I did not defiantly find out what NTSC Comb is.

Philips saa7111 TV decoder

- was introduced in 1997, is used in the BUZ and
- can handle: PAL B/G/H/I, PAL N, PAL M, NTSC M, NTSC N, NTSC 4.43 and SECAM

Philips saa7110a TV decoder

- was introduced in 1995, is used in the Pinnacle/Miro DC10(new), DC10+ and
- can handle: PAL B/G, NTSC M and SECAM

Philips saa7114 TV decoder

- was introduced in 2000, is used in the LML33R10 and
- can handle: PAL B/G/D/H/I/N, PAL N, PAL M, NTSC M, NTSC 4.43 and SECAM

Brooktree bt819 TV decoder

- was introduced in 1996, and is used in the LML33 and
- can handle: PAL B/D/G/H/I, NTSC M

Micronas vpx3220a TV decoder

- was introduced in 1996, is used in the DC30 and DC30+ and
- can handle: PAL B/G/H/I, PAL N, PAL M, NTSC M, NTSC 44, PAL 60, SECAM,NTSC Comb

Samsung ks0127 TV decoder

- is used in the AVS6EYES card and
- can handle: NTSC-M/N/44, PAL-M/N/B/G/H/I/D/K/L and SECAM

What the TV encoder can do an what not

The TV encoder is doing the “same” as the decoder, but in the other direction. You feed them digital data and the generate a Composite or SVHS signal. For information about the colorsystems and TV norm take a look in the TV decoder section.

Philips saa7185 TV Encoder

- was introduced in 1996, is used in the BUZ
- can generate: PAL B/G, NTSC M

Brooktree bt856 TV Encoder

- was introduced in 1994, is used in the LML33
- can generate: PAL B/D/G/H/I/N, PAL M, NTSC M, PAL-N (Argentina)

Analog Devices adv7170 TV Encoder

- was introduced in 2000, is used in the LML300R10
- can generate: PAL B/D/G/H/I/N, PAL M, NTSC M, PAL 60

Analog Devices adv7175 TV Encoder

- was introduced in 1996, is used in the DC10, DC10+, DC10 old, DC30, DC30+
- can generate: PAL B/D/G/H/I/N, PAL M, NTSC M

ITT mse3000 TV encoder

- was introduced in 1991, is used in the DC10 old
- can generate: PAL , NTSC , SECAM

Conexant bt866 TV encoder

- is used in AVS6EYES, and
- can generate: NTSC/PAL, PAL-M, PAL-N

The adv717x, should be able to produce PAL N. But you find nothing PAL N specific in the registers. Seem that you have to reuse a other standard to generate PAL N, maybe it would work if you use the PAL M settings.

How do I get this damn thing to work

Load zr36067.o. If it can't autodetect your card, use the card=X insmod option with X being the card number as given in the previous section. To have more than one card, use card=X1[,X2[,X3[,X4[.]]]]

To automate this, add the following to your /etc/modprobe.d/zoran.conf:

```
options zr36067 card=X1[,X2[,X3[,X4[.]]] alias char-major-81-0 zr36067
```


One thing to keep in mind is that this doesn't load zr36067.o itself yet. It just automates loading. If you start using xawtv, the device won't load on some systems, since you're trying to load modules as a user, which is not allowed ("permission denied"). A quick workaround is to add 'Load "v4l"' to XF86Config-4 when you use X by default, or to run 'v4l-conf -c <device>' in one of your startup scripts (normally rc.local) if you don't use X. Both make sure that the modules are loaded on startup, under the root account.

What mainboard should I use (or why doesn't my card work)

<insert lousy disclaimer here>. In short: good=SiS/Intel, bad=VIA.

Experience tells us that people with a Buz, on average, have more problems than users with a DC10+/LML33. Also, it tells us that people owning a VIA- based mainboard (ktXXX, MVP3) have more problems than users with a mainboard based on a different chipset. Here's some notes from Andrew Stevens:

Here's my experience of using LML33 and Buz on various motherboards:

- **VIA MVP3**
 - Forget it. Pointless. Doesn't work.
- **Intel 430FX (Pentium 200)**
 - LML33 perfect, Buz tolerable (3 or 4 frames dropped per movie)
- **Intel 440BX (early stepping)**
 - LML33 tolerable. Buz starting to get annoying (6-10 frames/hour)
- **Intel 440BX (late stepping)**
 - Buz tolerable, LML3 almost perfect (occasional single frame drops)
- **SiS735**
 - LML33 perfect, Buz tolerable.
- **VIA KT133(*)**
 - LML33 starting to get annoying, Buz poor enough that I have up.
- Both 440BX boards were dual CPU versions.

Bernhard Praschinger later added:

- **AMD 751**
 - Buz perfect-tolerable
- **AMD 760**
 - Buz perfect-tolerable

In general, people on the user mailinglist won't give you much of a chance if you have a VIA-based motherboard. They may be cheap, but sometimes, you'd rather want to spend some more money on better boards. In general, VIA mainboard's IDE/PCI performance will also suck badly compared to others. You'll noticed the DC10+/DC30+ aren't mentioned anywhere in the overview. Basically, you can assume that if the Buz works, the LML33 will work too. If the LML33 works, the DC10+/DC30+ will work too. They're most tolerant to different mainboard chipsets from all of the supported cards.

If you experience timeouts during capture, buy a better mainboard or lower the quality/buffersize during capture (see ‘Concerning buffer sizes, quality, output size etc.’). If it hangs, there’s little we can do as of now. Check your IRQs and make sure the card has its own interrupts.

Programming interface

This driver conforms to video4linux2. Support for V4L1 and for the custom zoran ioctls has been removed in kernel 2.6.38.

For programming example, please, look at lavrec.c and lavplay.c code in the MJPEG-tools (<http://mjpeg.sf.net/>).

Additional notes for software developers:

The driver returns maxwidth and maxheight parameters according to the current TV standard (norm). Therefore, the software which communicates with the driver and “asks” for these parameters should first set the correct norm. Well, it seems logically correct: TV standard is “more constant” for current country than geometry settings of a variety of TV capture cards which may work in ITU or square pixel format.

Applications

Applications known to work with this driver:

TV viewing:

- xawtv
- kwintv
- probably any TV application that supports video4linux or video4linux2.

MJPEG capture/playback:

- mjpegtools/lavtools (or Linux Video Studio)
- gstreamer
- mplayer

General raw capture:

- xawtv
- gstreamer
- probably any application that supports video4linux or video4linux2

Video editing:

- Cinelerra
- MainActor
- mjpegtools (or Linux Video Studio)

Concerning buffer sizes, quality, output size etc.

The zr36060 can do 1:2 JPEG compression. This is really the theoretical maximum that the chipset can reach. The driver can, however, limit compression to a maximum (size) of 1:4. The reason for this is that some cards (e.g. Buz) can't handle 1:2 compression without stopping capture after only a few minutes. With 1:4, it'll mostly work. If you have a Buz, use 'low_bitrate=1' to go into 1:4 max. compression mode.

100% JPEG quality is thus 1:2 compression in practice. So for a full PAL frame (size 720x576). The JPEG fields are stored in YUY2 format, so the size of the fields are 720x288x16/2 bits/field (2 fields/frame) = 207360 bytes/field x 2 = 414720 bytes/frame (add some more bytes for headers and DHT (huffman)/DQT (quantization) tables, and you'll get to something like 512kB per frame for 1:2 compression. For 1:4 compression, you'd have frames of half this size.

Some additional explanation by Martin Samuelsson, which also explains the importance of buffer sizes: - > Hmm, I do not think it is really that way. With the current (downloaded > at 18:00 Monday) driver I get that output sizes for 10 sec: > -q 50 -b 128 : 24.283.332 Bytes > -q 50 -b 256 : 48.442.368 > -q 25 -b 128 : 24.655.992 > -q 25 -b 256 : 25.859.820

I woke up, and can't go to sleep again. I'll kill some time explaining why this doesn't look strange to me.

Let's do some math using a width of 704 pixels. I'm not sure whether the Buz actually use that number or not, but that's not too important right now.

704x288 pixels, one field, is 202752 pixels. Divided by 64 pixels per block; 3168 blocks per field. Each pixel consist of two bytes; 128 bytes per block; 1024 bits per block. 100% in the new driver mean 1:2 compression; the maximum output becomes 512 bits per block. Actually 510, but 512 is simpler to use for calculations.

Let's say that we specify d1q50. We thus want 256 bits per block; times 3168 becomes 811008 bits; 101376 bytes per field. We're talking raw bits and bytes here, so we don't need to do any fancy corrections for bits-per-pixel or such things. 101376 bytes per field.

d1 video contains two fields per frame. Those sum up to 202752 bytes per frame, and one of those frames goes into each buffer.

But wait a second! -b128 gives 128kB buffers! It's not possible to cram 202752 bytes of JPEG data into 128kB!

This is what the driver notice and automatically compensate for in your examples. Let's do some math using this information:

128kB is 131072 bytes. In this buffer, we want to store two fields, which leaves 65536 bytes for each field. Using 3168 blocks per field, we get 20.68686868... available bytes per block; 165 bits. We can't allow the request for 256 bits per block when there's only 165 bits available! The -q50 option is silently overridden, and the -b128 option takes precedence, leaving us with the equivalence of -q32.

This gives us a data rate of 165 bits per block, which, times 3168, sums up to 65340 bytes per field, out of the allowed 65536. The current driver has another level of rate limiting; it won't accept -q values that fill more than 6/8 of the specified buffers. (I'm not sure why. "Playing it safe" seem to be a safe bet. Personally, I think I would have lowered requested-bits-per-block by one, or something like that.) We can't use 165 bits per block, but have to lower it again, to 6/8 of the available buffer space: We end up with 124 bits per block, the equivalence of -q24. With 128kB buffers, you can't use greater than -q24 at -d1. (And PAL, and 704 pixels width...)

The third example is limited to -q24 through the same process. The second example, using very similar calculations, is limited to -q48. The only example that actually grab at the specified -q value is the last one, which is clearly visible, looking at the file size. -

Conclusion: the quality of the resulting movie depends on buffer size, quality, whether or not you use 'low_bitrate=1' as insmod option for the zr36060.c module to do 1:4 instead of 1:2 compression, etc.

If you experience timeouts, lowering the quality/buffersize or using 'low_bitrate=1' as insmod option for zr36060.o might actually help, as is proven by the Buz.

It hangs/crashes/fails/whatevers! Help!

Make sure that the card has its own interrupts (see /proc/interrupts), check the output of dmesg at high verbosity (load zr36067.o with debug=2, load all other modules with debug=1). Check that your mainboard is favorable (see question 2) and if not, test the card in another computer. Also see the notes given in question 3 and try lowering quality/buffersize/capturesize if recording fails after a period of time.

If all this doesn't help, give a clear description of the problem including detailed hardware information (memory+brand, mainboard+chipset+brand, which MJPEG card, processor, other PCI cards that might be of interest), give the system PnP information (/proc/interrupts, /proc/dma, /proc/devices), and give the kernel version, driver version, glibc version, gcc version and any other information that might possibly be of interest. Also provide the dmesg output at high verbosity. See 'Contacting' on how to contact the developers.

Maintainers/Contacting

Previous maintainers/developers of this driver are - Laurent Pinchart <laurent.pinchart@skynet.be> - Ronald Bultje <rbultje@ronald.bitfreak.net> - Serguei Miridonov <mirsev@cicese.mx> - Wolfgang Scherr <scherr@net4you.net> - Dave Perks <dperks@ibm.net> - Rainer Johanni <Rainer@Johanni.de>

Driver's License

This driver is distributed under the terms of the General Public License.

This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

See <http://www.gnu.org/> for more information.

2.9.1.16 MIPI CCS camera sensor driver

The MIPI CCS camera sensor driver is a generic driver for [MIPI CCS](#) compliant camera sensors. It exposes three sub-devices representing the pixel array, the binner and the scaler.

As the capabilities of individual devices vary, the driver exposes interfaces based on the capabilities that exist in hardware.

Pixel Array sub-device

The pixel array sub-device represents the camera sensor's pixel matrix, as well as analogue crop functionality present in many compliant devices. The analogue crop is configured using the `V4L2_SEL_TGT_CROP` on the source pad (0) of the entity. The size of the pixel matrix can be obtained by getting the `V4L2_SEL_TGT_NATIVE_SIZE` target.

Binner

The binner sub-device represents the binning functionality on the sensor. For that purpose, selection target `V4L2_SEL_TGT_COMPOSE` is supported on the sink pad (0).

Additionally, if a device has no scaler or digital crop functionality, the source pad (1) exposes another digital crop selection rectangle that can only crop at the end of the lines and frames.

Scaler

The scaler sub-device represents the digital crop and scaling functionality of the sensor. The V4L2 selection target `V4L2_SEL_TGT_CROP` is used to configure the digital crop on the sink pad (0) when digital crop is supported. Scaling is configured using selection target `V4L2_SEL_TGT_COMPOSE` on the sink pad (0) as well.

Additionally, if the scaler sub-device exists, its source pad (1) exposes another digital crop selection rectangle that can only crop at the end of the lines and frames.

Digital and analogue crop

Digital crop functionality is referred to as cropping that effectively works by dropping some data on the floor. Analogue crop, on the other hand, means that the cropped information is never retrieved. In case of camera sensors, the analogue data is never read from the pixel matrix that are outside the configured selection rectangle that designates crop. The difference has an effect in device timing and likely also in power consumption.

Register definition generator

The `ccs-regs.asc` file contains MIPI CCS register definitions that are used to produce C source code files for definitions that can be better used by programs written in C language. As there are many dependencies between the produced files, please do not modify them manually as it's error-prone and in vain, but instead change the script producing them.

Usage

Conventionally the script is called this way to update the CCS driver definitions:

```
$ Documentation/driver-api/media/drivers/ccs/mk-ccs-regs -k \  
-e drivers/media/i2c/ccs/ccs-regs.h \  
-L drivers/media/i2c/ccs/ccs-limits.h \  
-l drivers/media/i2c/ccs/ccs-limits.c \  
-c Documentation/driver-api/media/drivers/ccs/ccs-regs.asc
```

2.9.1.17 CCS PLL calculator

The CCS PLL calculator is used to compute the PLL configuration, given sensor's capabilities as well as board configuration and user specified configuration. As the configuration space that encompasses all these configurations is vast, the PLL calculator isn't entirely trivial. Yet it is relatively simple to use for a driver.

The PLL model implemented by the PLL calculator corresponds to MIPI CCS 1.1.

struct **ccs_pll_branch_fr**
 CCS PLL configuration (front)

Definition

```
struct ccs_pll_branch_fr {  
    u16 pre_pll_clk_div;  
    u16 pll_multiplier;  
    u32 pll_ip_clk_freq_hz;  
    u32 pll_op_clk_freq_hz;  
};
```

Members

pre_pll_clk_div Pre-PLL clock divisor

pll_multiplier PLL multiplier

pll_ip_clk_freq_hz PLL input clock frequency

pll_op_clk_freq_hz PLL output clock frequency

Description

A single branch front-end of the CCS PLL tree.

struct **ccs_pll_branch_bk**
 CCS PLL configuration (back)

Definition

```

struct ccs_pll_branch_bk {
    u16 sys_clk_div;
    u16 pix_clk_div;
    u32 sys_clk_freq_hz;
    u32 pix_clk_freq_hz;
};

```

Members

sys_clk_div System clock divider

pix_clk_div Pixel clock divider

sys_clk_freq_hz System clock frequency

pix_clk_freq_hz Pixel clock frequency

Description

A single branch back-end of the CCS PLL tree.

struct **ccs_pll**
 Full CCS PLL configuration

Definition

```

struct ccs_pll {
    u8 bus_type;
    u8 op_lanes;
    u8 vt_lanes;
    struct {
        u8 lanes;
    } csi2;
    u8 binning_horizontal;
    u8 binning_vertical;
    u8 scale_m;
    u8 scale_n;
    u8 bits_per_pixel;
    u8 op_bits_per_lane;
    u16 flags;
    u32 link_freq;
    u32 ext_clk_freq_hz;
    struct ccs_pll_branch_fr vt_fr;
    struct ccs_pll_branch_bk vt_bk;
    struct ccs_pll_branch_fr op_fr;
    struct ccs_pll_branch_bk op_bk;
    u32 pixel_rate_csi;
    u32 pixel_rate_pixel_array;
};

```

Members

bus_type Type of the data bus, CCS_PLL_BUS_TYPE_* (input)

op_lanes Number of operational lanes (input)

vt_lanes Number of video timing lanes (input)

csi2 CSI-2 related parameters

csi2.lanes The number of the CSI-2 data lanes (input)

- binning_horizontal** Horizontal binning factor (input)
- binning_vertical** Vertical binning factor (input)
- scale_m** Downscaling factor, M component, [16, max] (input)
- scale_n** Downscaling factor, N component, typically 16 (input)
- bits_per_pixel** Bits per pixel on the output data bus (input)
- op_bits_per_lane** Number of bits per OP lane (input)
- flags** CCS_PLL_FLAG_* (input)
- link_freq** Chosen link frequency (input)
- ext_clk_freq_hz** External clock frequency, i.e. the sensor's input clock (input)
- vt_fr** Video timing front-end configuration (output)
- vt_bk** Video timing back-end configuration (output)
- op_fr** Operational timing front-end configuration (output)
- op_bk** Operational timing back-end configuration (output)
- pixel_rate_csi** Pixel rate on the output data bus (output)
- pixel_rate_pixel_array** Nominal pixel rate in the sensor's pixel array (output)

Description

All information required to calculate CCS PLL configuration.

struct **ccs_pll_branch_limits_fr**
 CCS PLL front-end limits

Definition

```
struct ccs_pll_branch_limits_fr {
    u16 min_pre_pll_clk_div;
    u16 max_pre_pll_clk_div;
    u32 min_pll_ip_clk_freq_hz;
    u32 max_pll_ip_clk_freq_hz;
    u16 min_pll_multiplier;
    u16 max_pll_multiplier;
    u32 min_pll_op_clk_freq_hz;
    u32 max_pll_op_clk_freq_hz;
};
```

Members

- min_pre_pll_clk_div** Minimum pre-PLL clock divider
- max_pre_pll_clk_div** Maximum pre-PLL clock divider
- min_pll_ip_clk_freq_hz** Minimum PLL input clock frequency
- max_pll_ip_clk_freq_hz** Maximum PLL input clock frequency
- min_pll_multiplier** Minimum PLL multiplier
- max_pll_multiplier** Maximum PLL multiplier
- min_pll_op_clk_freq_hz** Minimum PLL output clock frequency

max_pll_op_clk_freq_hz Maximum PLL output clock frequency

struct **ccs_pll_branch_limits_bk**
CCS PLL back-end limits

Definition

```
struct ccs_pll_branch_limits_bk {
    u16 min_sys_clk_div;
    u16 max_sys_clk_div;
    u32 min_sys_clk_freq_hz;
    u32 max_sys_clk_freq_hz;
    u16 min_pix_clk_div;
    u16 max_pix_clk_div;
    u32 min_pix_clk_freq_hz;
    u32 max_pix_clk_freq_hz;
};
```

Members

min_sys_clk_div Minimum system clock divider

max_sys_clk_div Maximum system clock divider

min_sys_clk_freq_hz Minimum system clock frequency

max_sys_clk_freq_hz Maximum system clock frequency

min_pix_clk_div Minimum pixel clock divider

max_pix_clk_div Maximum pixel clock divider

min_pix_clk_freq_hz Minimum pixel clock frequency

max_pix_clk_freq_hz Maximum pixel clock frequency

struct **ccs_pll_limits**
CCS PLL limits

Definition

```
struct ccs_pll_limits {
    u32 min_ext_clk_freq_hz;
    u32 max_ext_clk_freq_hz;
    struct ccs_pll_branch_limits_fr vt_fr;
    struct ccs_pll_branch_limits_bk vt_bk;
    struct ccs_pll_branch_limits_fr op_fr;
    struct ccs_pll_branch_limits_bk op_bk;
    u32 min_line_length_pck_bin;
    u32 min_line_length_pck;
};
```

Members

min_ext_clk_freq_hz Minimum external clock frequency

max_ext_clk_freq_hz Maximum external clock frequency

vt_fr Video timing front-end limits

vt_bk Video timing back-end limits

op_fr Operational timing front-end limits

op_bk Operational timing back-end limits

min_line_length_pck_bin Minimum line length in pixels, with binning

min_line_length_pck Minimum line length in pixels without binning

int **ccs_pll_calculate**(struct device *dev, const struct *ccs_pll_limits* *limits, struct *ccs_pll* *pll)

Calculate CCS PLL configuration based on input parameters

Parameters

struct device *dev Device pointer, used for printing messages

const struct ccs_pll_limits *limits Limits specific to the sensor

struct ccs_pll *pll Given PLL configuration

Description

Calculate the CCS PLL configuration based on the limits as well as given device specific, system specific or user configured input data.

Copyright © 2020 Intel Corporation

2.9.2 Digital TV drivers

2.9.2.1 Idea behind the dvb-usb-framework

Note:

- 1) This documentation is outdated. Please check at the DVB wiki at <https://linuxtv.org/wiki> for more updated info.
 - 2) **deprecated:** Newer DVB USB drivers should use the dvb-usb-v2 framework.
-

In March 2005 I got the new Twinhan USB2.0 DVB-T device. They provided specs and a firmware.

Quite keen I wanted to put the driver (with some quirks of course) into dibusb. After reading some specs and doing some USB snooping, it realized, that the dibusb-driver would be a complete mess afterwards. So I decided to do it in a different way: With the help of a dvb-usb-framework.

The framework provides generic functions (mostly kernel API calls), such as:

- Transport Stream URB handling in conjunction with dvb-demux-feed-control (bulk and isoc are supported)
- registering the device for the DVB-API
- registering an I2C-adapter if applicable
- remote-control/input-device handling
- firmware requesting and loading (currently just for the Cypress USB controllers)
- other functions/methods which can be shared by several drivers (such as functions for bulk-control-commands)

- TODO: a I2C-chunker. It creates device-specific chunks of register-accesses depending on length of a register and the number of values that can be multi-written and multi-read.

The source code of the particular DVB USB devices does just the communication with the device via the bus. The connection between the DVB-API-functionality is done via callbacks, assigned in a static device-description (struct `dvb_usb_device`) each device-driver has to have.

For an example have a look in `drivers/media/usb/dvb-usb/vp7045*`.

Objective is to migrate all the usb-devices (`dibusb`, `cinergyT2`, maybe the `ttusb`; `flexcop-usb` already benefits from the generic `flexcop-device`) to use the `dvb-usb-lib`.

TODO: dynamic enabling and disabling of the pid-filter in regard to number of feeds requested.

Supported devices

See the LinuxTV DVB Wiki at <https://linuxtv.org> for a complete list of cards/drivers/firmwares: https://linuxtv.org/wiki/index.php/DVB_USB

0. History & News:

2005-06-30

- added support for WideView WT-220U (Thanks to Steve Chang)

2005-05-30

- added basic isochronous support to the `dvb-usb-framework`
- **added support for Conexant Hybrid reference design and Nebula DigiTV USB**

2005-04-17

- all `dibusb` devices ported to make use of the `dvb-usb-framework`

2005-04-02

- re-enabled and improved remote control code.

2005-03-31

- ported the Yakumo/Hama/Typhoon DVB-T USB2.0 device to `dvb-usb`.

2005-03-30

- first commit of the `dvb-usb-module` based on the `dibusb-source`. First device is a new driver for the TwinhanDTV Alpha / MagicBox II USB2.0-only DVB-T device.
- (change from `dvb-dibusb` to `dvb-usb`)

2005-03-28

- added support for the AVerMedia AverTV DVB-T USB2.0 device (Thanks to Glen Harris and Jiun-Kuei Jung, AVerMedia)

2005-03-14

- added support for the Typhoon/Yakumo/HAMA DVB-T mobile USB2.0

2005-02-11

- added support for the KWorld/ADSTech Instant DVB-T USB2.0. Thanks a lot to Joachim von Caron

2005-02-02 - added support for the Hauppauge Win-TV Nova-T USB2

2005-01-31 - distorted streaming is gone for USB1.1 devices

2005-01-13

- moved the mirrored `pid_filter_table` back to `dvb-dibusb` first almost working version for HanfTek UMT-010 found out, that Yakumo/HAMA/Typhoon are predecessors of the HanfTek UMT-010

2005-01-10

- refactoring completed, now everything is very delightful
- tuner quirks for some weird devices (Artec T1 AN2235 device has sometimes a Panasonic Tuner assembled). Tunerprobing implemented. Thanks a lot to Gunnar Wittich.

2004-12-29

- after several days of struggling around bug of no returning URBs fixed.

2004-12-26

- refactored the `dibusb-driver`, split into separate files
- `i2c-probing` enabled

2004-12-06

- possibility for demod `i2c-address` probing
- new usb IDs (Compro, Artec)

2004-11-23

- merged changes from `DiB3000MC_ver2.1`
- revised the debugging
- possibility to deliver the complete TS for USB2.0

2004-11-21

- first working version of the `dib3000mc/p` frontend driver.

2004-11-12

- added additional remote control keys. Thanks to Uwe Hanke.

2004-11-07

- added remote control support. Thanks to David Matthews.

2004-11-05

- added support for a new devices (Grandtec/Avermedia/Artec)
- merged my changes (for `dib3000mb/dibusb`) to the `FE_REFACTORING`, because it became `HEAD`
- moved transfer control (`pid filter`, `fifo control`) from `usb driver` to `frontend`, it seems better settled there (added `xfer_ops-struct`)

- created a common files for frontends (mc/p/mb)

2004-09-28

- added support for a new device (Unknown, vendor ID is Hyper-Paltek)

2004-09-20

- added support for a new device (Compro DVB-U2000), thanks to Amaury Demol for reporting
- changed usb TS transfer method (several urbs, stopping transfer before setting a new pid)

2004-09-13

- added support for a new device (Artec T1 USB TVBOX), thanks to Christian Motschke for reporting

2004-09-05

- released the dibusb device and dib3000mb-frontend driver (old news for vp7041.c)

2004-07-15

- found out, by accident, that the device has a TUA6010XS for PLL

2004-07-12

- figured out, that the driver should also work with the CTS Portable (Chinese Television System)

2004-07-08

- firmware-extraction-2.422-problem solved, driver is now working properly with firmware extracted from 2.422
- #if for 2.6.4 (dVB), compile issue
- changed firmware handling, see vp7041.txt sec 1.1

2004-07-02

- some tuner modifications, v0.1, cleanups, first public

2004-06-28

- now using the dvb_dmx_swfilter_packets, everything runs fine now

2004-06-27

- able to watch and switching channels (pre-alpha)
- no section filtering yet

2004-06-06

- first TS received, but kernel oops :/

2004-05-14

- firmware loader is working

2004-05-11

- start writing the driver

How to use?

Firmware

Most of the USB drivers need to download a firmware to the device before start working.

Have a look at the Wikipage for the DVB-USB-drivers to find out, which firmware you need for your device:

https://linuxtv.org/wiki/index.php/DVB_USB

Compiling

Since the driver is in the linux kernel, activating the driver in your favorite config-environment should sufficient. I recommend to compile the driver as module. Hotplug does the rest.

If you use dvb-kernel enter the build-2.6 directory run 'make' and 'insmod.sh load' afterwards.

Loading the drivers

Hotplug is able to load the driver, when it is needed (because you plugged in the device).

If you want to enable debug output, you have to load the driver manually and from within the dvb-kernel cvs repository.

first have a look, which debug level are available:

```
# modinfo dvb-usb
# modinfo dvb-usb-vp7045

etc.
```

```
modprobe dvb-usb debug=<level>
modprobe dvb-usb-vp7045 debug=<level>
etc.
```

should do the trick.

When the driver is loaded successfully, the firmware file was in the right place and the device is connected, the "Power"-LED should be turned on.

At this point you should be able to start a dvb-capable application. I'm use (t|s)zap, mplayer and dvbscan to test the basics. VDR-xine provides the long-term test scenario.

Known problems and bugs

- Don't remove the USB device while running an DVB application, your system will go crazy or die most likely.

Adding support for devices

TODO

USB1.1 Bandwidth limitation

A lot of the currently supported devices are USB1.1 and thus they have a maximum bandwidth of about 5-6 MBit/s when connected to a USB2.0 hub. This is not enough for receiving the complete transport stream of a DVB-T channel (which is about 16 MBit/s). Normally this is not a problem, if you only want to watch TV (this does not apply for HDTV), but watching a channel while recording another channel on the same frequency simply does not work very well. This applies to all USB1.1 DVB-T devices, not just the dvb-usb-devices)

The bug, where the TS is distorted by a heavy usage of the device is gone definitely. All dvb-usb-devices I was using (Twinhan, Kworld, DiBcom) are working like charm now with VDR. Sometimes I even was able to record a channel and watch another one.

Comments

Patches, comments and suggestions are very very welcome.

3. Acknowledgements

Amaury Demol (Amaury.Demol@parrot.com) and Francois Kanounnikoff from DiBcom for providing specs, code and help, on which the dvb-dibusb, dib3000mb and dib3000mc are based.

David Matthews for identifying a new device type (Artec T1 with AN2235) and for extending dibusb with remote control event handling. Thank you.

Alex Woods for frequently answering question about usb and dvb stuff, a big thank you.

Bernd Wagner for helping with huge bug reports and discussions.

Gunnar Wittich and Joachim von Caron for their trust for providing root-shells on their machines to implement support for new devices.

Allan Third and Michael Hutchinson for their help to write the Nebula digitv-driver.

Glen Harris for bringing up, that there is a new dibusb-device and Jiun-Kuei Jung from AVerMedia who kindly provided a special firmware to get the device up and running in Linux.

Jennifer Chen, Jeff and Jack from Twinhan for kindly supporting by writing the vp7045-driver.

Steve Chang from WideView for providing information for new devices and firmware files.

Michael Paxton for submitting remote control keymaps.

Some guys on the linux-dvb mailing list for encouraging me.

Peter Schildmann >peter.schildmann-nospam-at-web.de< for his user-level firmware loader, which saves a lot of time (when writing the vp7041 driver)

Ulf Hermenau for helping me out with traditional chinese.

André Smoktun and Christian Frömmel for supporting me with hardware and listening to my problems very patiently.

2.9.2.2 Frontend drivers

Frontend attach headers

struct **a8293_platform_data**
Platform data for the a8293 driver

Definition

```
struct a8293_platform_data {
    struct dvb_frontend *dvb_frontend;
};
```

Members

dvb_frontend DVB frontend.

struct **af9013_platform_data**
Platform data for the af9013 driver

Definition

```
struct af9013_platform_data {
    u32 clk;
#define AF9013_TUNER_MXL5003D      3 ;
#define AF9013_TUNER_MXL5005D     13 ;
#define AF9013_TUNER_MXL5005R     30 ;
#define AF9013_TUNER_ENV77H11D5  129 ;
#define AF9013_TUNER_MT2060       130 ;
#define AF9013_TUNER_MC44S803     133 ;
#define AF9013_TUNER_QT1010       134 ;
#define AF9013_TUNER_UNKNOWN      140 ;
#define AF9013_TUNER_MT2060_2     147 ;
#define AF9013_TUNER_TDA18271     156 ;
#define AF9013_TUNER_QT1010A      162 ;
#define AF9013_TUNER_MXL5007T     177 ;
#define AF9013_TUNER_TDA18218     179 ;
    u8 tuner;
    u32 if_frequency;
#define AF9013_TS_MODE_USB        0;
#define AF9013_TS_MODE_PARALLEL  1;
#define AF9013_TS_MODE_SERIAL    2;
    u8 ts_mode;
```



```

u8 ts_output_pin;
bool spec_inv;
u8 api_version[4];
#define AF9013_GPIO_ON (1 << 0);
#define AF9013_GPIO_EN (1 << 1);
#define AF9013_GPIO_0 (1 << 2);
#define AF9013_GPIO_I (1 << 3);
#define AF9013_GPIO_LO (AF9013_GPIO_ON|AF9013_GPIO_EN);
#define AF9013_GPIO_HI (AF9013_GPIO_ON|AF9013_GPIO_EN|AF9013_GPIO_0);
#define AF9013_GPIO_TUNER_ON (AF9013_GPIO_ON|AF9013_GPIO_EN);
#define AF9013_GPIO_TUNER_OFF (AF9013_GPIO_ON|AF9013_GPIO_EN|AF9013_GPIO_0);
u8 gpio[4];
struct dvb_frontend* (*get_dvb_frontend)(struct i2c_client *);
struct i2c_adapter* (*get_i2c_adapter)(struct i2c_client *);
int (*pid_filter_ctrl)(struct dvb_frontend *, int);
int (*pid_filter)(struct dvb_frontend *, u8, u16, int);
};

```

Members

clk Clock frequency.

tuner Used tuner model.

if_frequency IF frequency.

ts_mode TS mode.

ts_output_pin TS output pin.

spec_inv Input spectrum inverted.

api_version Firmware API version.

gpio GPIOs.

get_dvb_frontend Get DVB frontend callback.

get_i2c_adapter Get I2C adapter.

pid_filter_ctrl Control PID filter.

pid_filter Set PID to PID filter.

struct **ascot2e_config**

the configuration of Ascot2E tuner driver

Definition

```

struct ascot2e_config {
    u8 i2c_address;
    u8 xtal_freq_mhz;
    void *set_tuner_priv;
    int (*set_tuner_callback)(void *, int);
};

```

Members

i2c_address I2C address of the tuner

xtal_freq_mhz Oscillator frequency, MHz

set_tuner_priv Callback function private context

set_tuner_callback Callback function that notifies the parent driver which tuner is active now

struct *dvb_frontend* * **ascot2e_attach**(struct *dvb_frontend* **fe*, const struct *ascot2e_config* **config*, struct i2c_adapter **i2c*)

Attach an ascot2e tuner

Parameters

struct dvb_frontend *fe frontend to be attached

const struct ascot2e_config *config pointer to *struct ascot2e_config* with tuner configuration.

struct i2c_adapter *i2c i2c adapter to use.

Return

FE pointer on success, NULL on failure.

struct **cxd2820r_platform_data**
Platform data for the cxd2820r driver

Definition

```
struct cxd2820r_platform_data {
    u8 ts_mode;
    bool ts_clk_inv;
    bool if_agc_polarity;
    bool spec_inv;
    int **gpio_chip_base;
    struct dvb_frontend* (*get_dvb_frontend)(struct i2c_client *);
};
```

Members

ts_mode TS mode.

ts_clk_inv TS clock inverted.

if_agc_polarity IF AGC polarity.

spec_inv Input spectrum inverted.

gpio_chip_base GPIO.

get_dvb_frontend Get DVB frontend.

struct **cxd2820r_config**
configuration for cxd2020r demod

Definition

```
struct cxd2820r_config {
    u8 i2c_address;
    u8 ts_mode;
    bool ts_clock_inv;
    bool if_agc_polarity;
    bool spec_inv;
};
```

Members

i2c_address Demodulator I2C address. Driver determines DVB-C slave I2C address automatically from master address. Default: none, must set. Values: 0x6c, 0x6d.

ts_mode TS output mode. Default: none, must set. Values: FIXME?

ts_clock_inv TS clock inverted. Default: 0. Values: 0, 1.

if_agc_polarity Default: 0. Values: 0, 1

spec_inv Spectrum inversion. Default: 0. Values: 0, 1.

struct *dvb_frontend* * **cxd2820r_attach**(const struct *cxd2820r_config* *config, struct *i2c_adapter* *i2c, int *gpio_chip_base)

Attach a cxd2820r demod

Parameters

const struct cxd2820r_config *config pointer to *struct cxd2820r_config* with demod configuration.

struct i2c_adapter *i2c i2c adapter to use.

int *gpio_chip_base if zero, disables GPIO setting. Otherwise, if CONFIG_GPIOLIB is set dynamically allocate gpio base; if is not set, use its value to setup the GPIO pins.

Return

FE pointer on success, NULL on failure.

struct **drxk_config**

Configure the initial parameters for DRX-K

Definition

```
struct drxk_config {
    u8 adr;
    bool single_master;
    bool no_i2c_bridge;
    bool parallel_ts;
    bool dynamic_clk;
    bool enable_merr_cfg;
    bool antenna_dvbt;
    u16 antenna_gpio;
    u8 mpeg_out_clk_strength;
    int chunk_size;
    const char *microcode_name;
    int qam_demod_parameter_count;
};
```

Members

adr I2C address of the DRX-K

single_master Device is on the single master mode

no_i2c_bridge Don't switch the I2C bridge to talk with tuner

parallel_ts True means that the device uses parallel TS, Serial otherwise.

dynamic_clk True means that the clock will be dynamically adjusted. Static clock otherwise.

enable_merr_cfg Enable SIO_PDR_PERR_CFG/SIO_PDR_MVAL_CFG.

antenna_dvbt GPIO bit for changing antenna to DVB-C. A value of 1 means that 1=DVBC, 0 = DVBT. Zero means the opposite.

antenna_gpio GPIO bit used to control the antenna

mpeg_out_clk_strength DRXK Mpeg output clock drive strength.

chunk_size maximum size for I2C messages

microcode_name Name of the firmware file with the microcode

qam_demod_parameter_count The number of parameters used for the command to set the demodulator parameters. All firmwares are using the 2-parameter command. An exception is the `drxk_a3.mc` firmware, which uses the 4-parameter command. A value of 0 (default) or lower indicates that the correct number of parameters will be automatically detected.

Description

On the `*_gpio` vars, bit 0 is UIO-1, bit 1 is UIO-2 and bit 2 is UIO-3.

```
struct dvb_frontend * drxk_attach(const struct drxk_config *config, struct  
                                i2c_adapter *i2c)  
    Attach a drxk demod
```

Parameters

const struct drxk_config *config pointer to `struct drxk_config` with demod configuration.

struct i2c_adapter *i2c i2c adapter to use.

Return

FE pointer on success, NULL on failure.

```
struct dvb_frontend * dvb_pll_attach(struct dvb_frontend *fe, int pll_addr, struct  
                                i2c_adapter *i2c, unsigned int pll_desc_id)  
    Attach a dvb-pll to the supplied frontend structure.
```

Parameters

struct dvb_frontend *fe Frontend to attach to.

int pll_addr i2c address of the PLL (if used).

struct i2c_adapter *i2c i2c adapter to use (set to NULL if not used).

unsigned int pll_desc_id dvb_pll_desc to use.

Return

Frontend pointer on success, NULL on failure

struct helene_config
the configuration of 'Helene' tuner driver

Definition

```
struct helene_config {  
    u8 i2c_address;  
    u8 xtal_freq_mhz;  
    void *set_tuner_priv;
```

```

int (*set_tuner_callback)(void *, int);
enum helene_xtal xtal;
struct dvb_frontend *fe;
};

```

Members

i2c_address I2C address of the tuner

xtal_freq_mhz Oscillator frequency, MHz

set_tuner_priv Callback function private context

set_tuner_callback Callback function that notifies the parent driver which tuner is active now

xtal Cristal frequency as described by enum `helene_xtal`

fe Frontend for which connects this tuner

struct *dvb_frontend* * **helene_attach**(struct *dvb_frontend* *fe, const struct *helene_config* *config, struct *i2c_adapter* *i2c)

Attach a helene tuner (terrestrial and cable standards)

Parameters

struct *dvb_frontend* *fe frontend to be attached

const struct *helene_config* *config pointer to *struct helene_config* with tuner configuration.

struct *i2c_adapter* *i2c i2c adapter to use.

Return

FE pointer on success, NULL on failure.

struct *dvb_frontend* * **helene_attach_s**(struct *dvb_frontend* *fe, const struct *helene_config* *config, struct *i2c_adapter* *i2c)

Attach a helene tuner (satellite standards)

Parameters

struct *dvb_frontend* *fe frontend to be attached

const struct *helene_config* *config pointer to *struct helene_config* with tuner configuration.

struct *i2c_adapter* *i2c i2c adapter to use.

Return

FE pointer on success, NULL on failure.

struct **horus3a_config**

the configuration of Horus3A tuner driver

Definition

```

struct horus3a_config {
    u8 i2c_address;
    u8 xtal_freq_mhz;
    void *set_tuner_priv;
};

```

```
int (*set_tuner_callback)(void *, int);
};
```

Members

i2c_address I2C address of the tuner

xtal_freq_mhz Oscillator frequency, MHz

set_tuner_priv Callback function private context

set_tuner_callback Callback function that notifies the parent driver which tuner is active now

struct *dvb_frontend* * **horus3a_attach**(struct *dvb_frontend* **fe*, const struct *horus3a_config* **config*, struct i2c_adapter **i2c*)
Attach a horus3a tuner

Parameters

struct dvb_frontend *fe frontend to be attached

const struct horus3a_config *config pointer to *struct helene_config* with tuner configuration.

struct i2c_adapter *i2c i2c adapter to use.

Return

FE pointer on success, NULL on failure.

struct **ix2505v_config**
ix2505 attachment configuration

Definition

```
struct ix2505v_config {
    u8 tuner_address;
    u8 tuner_gain;
    u8 tuner_changepump;
    int min_delay_ms;
    u8 tuner_write_only;
};
```

Members

tuner_address tuner address

tuner_gain Baseband AMP gain control 0/1=0dB(default) 2=-2dB 3=-4dB

tuner_changepump Charge pump output +/- 0=120 1=260 2=555 3=1200(default)

min_delay_ms delay after tune

tuner_write_only disables reads

struct *dvb_frontend* * **ix2505v_attach**(struct *dvb_frontend* **fe*, const struct *ix2505v_config* **config*, struct i2c_adapter **i2c*)
Attach a ix2505v tuner to the supplied frontend structure.

Parameters

struct dvb_frontend *fe Frontend to attach to.

const struct ix2505v_config *config pointer to *struct ix2505v_config*

struct i2c_adapter *i2c pointer to *struct i2c_adapter*.

Return

FE pointer on success, NULL on failure.

enum **m88ds3103_ts_mode**
TS connection mode

Constants

M88DS3103_TS_SERIAL TS output pin D0, normal

M88DS3103_TS_SERIAL_D7 TS output pin D7

M88DS3103_TS_PARALLEL TS Parallel mode

M88DS3103_TS_CI TS CI Mode

enum **m88ds3103_clock_out**

Constants

M88DS3103_CLOCK_OUT_DISABLED Clock output is disabled

M88DS3103_CLOCK_OUT_ENABLED Clock output is enabled with crystal clock.

M88DS3103_CLOCK_OUT_ENABLED_DIV2 Clock output is enabled with half crystal clock.

struct **m88ds3103_platform_data**
Platform data for the m88ds3103 driver

Definition

```
struct m88ds3103_platform_data {
    u32 clk;
    u16 i2c_wr_max;
    enum m88ds3103_ts_mode ts_mode;
    u32 ts_clk;
    enum m88ds3103_clock_out clk_out;
    u8 ts_clk_pol:1;
    u8 spec_inv:1;
    u8 agc;
    u8 agc_inv:1;
    u8 envelope_mode:1;
    u8 lnb_hv_pol:1;
    u8 lnb_en_pol:1;
    struct dvb_frontend* (*get_dvb_frontend)(struct i2c_client *);
    struct i2c_adapter* (*get_i2c_adapter)(struct i2c_client *);
};
```

Members

clk Clock frequency.

i2c_wr_max Max bytes I2C adapter can write at once.

ts_mode TS mode.

ts_clk TS clock (KHz).

clk_out Clock output.

ts_clk_pol TS clk polarity. 1-active at falling edge; 0-active at rising edge.

spec_inv Input spectrum inversion.

agc AGC configuration.

agc_inv AGC polarity.

envelope_mode DiSEqC envelope mode.

lnb_hv_pol LNB H/V pin polarity. 0: pin high set to VOLTAGE_18, pin low to set VOLTAGE_13.
1: pin high set to VOLTAGE_13, pin low to set VOLTAGE_18.

lnb_en_pol LNB enable pin polarity. 0: pin high to disable, pin low to enable. 1: pin high to enable, pin low to disable.

get_dvb_frontend Get DVB frontend.

get_i2c_adapter Get I2C adapter.

struct **m88ds3103_config**
m88ds3102 configuration

Definition

```
struct m88ds3103_config {
    u8 i2c_addr;
    u32 clock;
    u16 i2c_wr_max;
    u8 ts_mode;
    u32 ts_clk;
    u8 ts_clk_pol:1;
    u8 spec_inv:1;
    u8 agc_inv:1;
    u8 clock_out;
    u8 envelope_mode:1;
    u8 agc;
    u8 lnb_hv_pol:1;
    u8 lnb_en_pol:1;
};
```

Members

i2c_addr I2C address. Default: none, must set. Example: 0x68, ...

clock Device's clock. Default: none, must set. Example: 27000000

i2c_wr_max Max bytes I2C provider is asked to write at once. Default: none, must set. Example: 33, 65, ...

ts_mode TS output mode, as defined by [enum m88ds3103_ts_mode](#). Default: M88DS3103_TS_SERIAL.

ts_clk TS clk in KHz. Default: 0.

ts_clk_pol TS clk polarity. Default: 0. 1-active at falling edge; 0-active at rising edge.

spec_inv Spectrum inversion. Default: 0.

agc_inv AGC polarity. Default: 0.

clock_out Clock output, as defined by [enum m88ds3103_clock_out](#). Default: M88DS3103_CLOCK_OUT_DISABLED.

envelope_mode DiSEqC envelope mode. Default: 0.

agc AGC configuration. Default: none, must set.

lnb_hv_pol LNB H/V pin polarity. Default: 0. Values: 1: pin high set to VOLTAGE_13, pin low to set VOLTAGE_18; 0: pin high set to VOLTAGE_18, pin low to set VOLTAGE_13.

lnb_en_pol LNB enable pin polarity. Default: 0. Values: 1: pin high to enable, pin low to disable; 0: pin high to disable, pin low to enable.

```
struct dvb_frontend * m88ds3103_attach(const struct m88ds3103_config *config,
                                     struct i2c_adapter *i2c, struct
                                     i2c_adapter **tuner_i2c)
```

Attach a m88ds3103 demod

Parameters

const struct m88ds3103_config *config pointer to *struct m88ds3103_config* with demod configuration.

struct i2c_adapter *i2c i2c adapter to use.

struct i2c_adapter **tuner_i2c on success, returns the I2C adapter associated with m88ds3103 tuner.

Return

FE pointer on success, NULL on failure.

Note

Do not add new *m88ds3103_attach()* users! Use I2C bindings instead.

struct mb86a20s_config

Define the per-device attributes of the frontend

Definition

```
struct mb86a20s_config {
    u32 fclk;
    u8 demod_address;
    bool is_serial;
};
```

Members

fclk Clock frequency. If zero, assumes the default (32.57142 Mhz)

demod_address the demodulator's i2c address

is_serial if true, TS is serial. Otherwise, TS is parallel

```
struct dvb_frontend * mb86a20s_attach(const struct mb86a20s_config *config, struct
                                     i2c_adapter *i2c)
```

Attach a mb86a20s demod

Parameters

const struct mb86a20s_config *config pointer to *struct mb86a20s_config* with demod configuration.

struct i2c_adapter *i2c i2c adapter to use.

Return

FE pointer on success, NULL on failure.

struct `mn88472_config`

Platform data for the mn88472 driver

Definition

```
struct mn88472_config {
    unsigned int xtal;
#define MN88472_TS_MODE_SERIAL      0;
#define MN88472_TS_MODE_PARALLEL   1;
    int ts_mode;
#define MN88472_TS_CLK_FIXED        0;
#define MN88472_TS_CLK_VARIABLE    1;
    int ts_clock;
    u16 i2c_wr_max;
    struct dvb_frontend **fe;
    struct dvb_frontend* (*get_dvb_frontend)(struct i2c_client *);
};
```

Members

xtal Clock frequency.

ts_mode TS mode.

ts_clock TS clock config.

i2c_wr_max Max number of bytes driver writes to I2C at once.

fe pointer to a frontend pointer

get_dvb_frontend Get DVB frontend callback.

struct `rtl2830_platform_data`

Platform data for the rtl2830 driver

Definition

```
struct rtl2830_platform_data {
    u32 clk;
    bool spec_inv;
    u8 vtop;
    u8 krf;
    u8 agc_targ_val;
    struct dvb_frontend* (*get_dvb_frontend)(struct i2c_client *);
    struct i2c_adapter* (*get_i2c_adapter)(struct i2c_client *);
    int (*pid_filter)(struct dvb_frontend *, u8, u16, int);
    int (*pid_filter_ctrl)(struct dvb_frontend *, int);
};
```

Members

clk Clock frequency (4000000, 16000000, 25000000, 28800000).

spec_inv Spectrum inversion.

vtop AGC take-over point.

krf AGC ratio.

agc_targ_val AGC.

get_dvb_frontend Get DVB frontend.

get_i2c_adapter Get I2C adapter.

pid_filter Set PID to PID filter.

pid_filter_ctrl Control PID filter.

struct **rtl2832_platform_data**
Platform data for the rtl2832 driver

Definition

```
struct rtl2832_platform_data {
    u32 clk;
#define RTL2832_TUNER_FC2580    0x21;
#define RTL2832_TUNER_TUA9001  0x24;
#define RTL2832_TUNER_FC0012   0x26;
#define RTL2832_TUNER_E4000    0x27;
#define RTL2832_TUNER_FC0013   0x29;
#define RTL2832_TUNER_R820T    0x2a;
#define RTL2832_TUNER_R828D    0x2b;
#define RTL2832_TUNER_SI2157   0x2c;
    u8 tuner;
    struct dvb_frontend* (*get_dvb_frontend)(struct i2c_client *);
    struct i2c_adapter* (*get_i2c_adapter)(struct i2c_client *);
    int (*slave_ts_ctrl)(struct i2c_client *, bool);
    int (*pid_filter)(struct dvb_frontend *, u8, u16, int);
    int (*pid_filter_ctrl)(struct dvb_frontend *, int);
};
```

Members

clk Clock frequency (4000000, 16000000, 25000000, 28800000).

tuner Used tuner model.

get_dvb_frontend Get DVB frontend.

get_i2c_adapter Get I2C adapter.

slave_ts_ctrl Control slave TS interface.

pid_filter Set PID to PID filter.

pid_filter_ctrl Control PID filter.

struct **rtl2832_sdr_platform_data**
Platform data for the rtl2832_sdr driver

Definition

```
struct rtl2832_sdr_platform_data {
    u32 clk;
#define RTL2832_SDR_TUNER_FC2580    0x21;
#define RTL2832_SDR_TUNER_TUA9001  0x24;
#define RTL2832_SDR_TUNER_FC0012   0x26;
#define RTL2832_SDR_TUNER_E4000    0x27;
#define RTL2832_SDR_TUNER_FC0013   0x29;
#define RTL2832_SDR_TUNER_R820T    0x2a;
```

```
#define RTL2832_SDR_TUNER_R828D    0x2b;
    u8 tuner;
    struct regmap *regmap;
    struct dvb_frontend *dvb_frontend;
    struct v4l2_subdev *v4l2_subdev;
    struct dvb_usb_device *dvb_usb_device;
};
```

Members

clk Clock frequency (4000000, 16000000, 25000000, 28800000).

tuner Used tuner model.

regmap pointer to struct regmap.

dvb_frontend rtl2832 DVB frontend.

v4l2_subdev Tuner v4l2 controls.

dvb_usb_device DVB USB interface for USB streaming.

struct *dvb_frontend* * **stb6000_attach**(struct *dvb_frontend* **fe*, int *addr*, struct *i2c_adapter* **i2c*)
Attach a stb6000 tuner to the supplied frontend structure.

Parameters

struct dvb_frontend *fe Frontend to attach to.

int addr i2c address of the tuner.

struct i2c_adapter *i2c i2c adapter to use.

Return

FE pointer on success, NULL on failure.

struct **tda10071_platform_data**
Platform data for the tda10071 driver

Definition

```
struct tda10071_platform_data {
    u32 clk;
    u16 i2c_wr_max;
#define TDA10071_TS_SERIAL    0;
#define TDA10071_TS_PARALLEL  1;
    u8 ts_mode;
    bool spec_inv;
    u8 pll_multiplier;
    u8 tuner_i2c_addr;
    struct dvb_frontend* (*get_dvb_frontend)(struct i2c_client *);
};
```

Members

clk Clock frequency.

i2c_wr_max Max bytes I2C adapter can write at once.

ts_mode TS mode.

spec_inv Input spectrum inversion.

pll_multiplier PLL multiplier.

tuner_i2c_addr CX24118A tuner I2C address (0x14, 0x54, ...).

get_dvb_frontend Get DVB frontend.

```
struct dvb_frontend* tda826x_attach(struct dvb_frontend *fe, int addr, struct
                                i2c_adapter *i2c, int has_loophrough)
    Attach a tda826x tuner to the supplied frontend structure.
```

Parameters

struct dvb_frontend *fe Frontend to attach to.

int addr i2c address of the tuner.

struct i2c_adapter *i2c i2c adapter to use.

int has_loophrough Set to 1 if the card has a loopthrough RF connector.

Return

FE pointer on success, NULL on failure.

```
struct zd1301_demod_platform_data
    Platform data for the zd1301_demod driver
```

Definition

```
struct zd1301_demod_platform_data {
    void *reg_priv;
    int (*reg_read)(void *, u16, u8 *);
    int (*reg_write)(void *, u16, u8);
};
```

Members

reg_priv First argument of reg_read and reg_write callbacks.

reg_read Register read callback.

reg_write Register write callback.

```
struct dvb_frontend* zd1301_demod_get_dvb_frontend(struct platform_device *pdev)
    Get pointer to DVB frontend
```

Parameters

struct platform_device *pdev Pointer to platform device

Return

Pointer to DVB frontend which given platform device owns.

```
struct i2c_adapter* zd1301_demod_get_i2c_adapter(struct platform_device *pdev)
    Get pointer to I2C adapter
```

Parameters

struct platform_device *pdev Pointer to platform device

Return

Pointer to I2C adapter which given platform device owns.

```
struct dvb_frontend * zl10036_attach(struct dvb_frontend *fe, const struct  
                                zl10036_config *config, struct i2c_adapter *i2c)  
    Attach a zl10036 tuner to the supplied frontend structure.
```

Parameters

struct *dvb_frontend* **fe* Frontend to attach to.

const struct *zl10036_config* **config* *zl10036_config* structure.

struct *i2c_adapter* **i2c* pointer to struct *i2c_adapter*.

Return

FE pointer on success, NULL on failure.

2.9.2.3 vidtv: Virtual Digital TV driver

Author: Daniel W. S. Almeida <dwlsalmeida@gmail.com>, June 2020.

Background

Vidtv is a virtual DVB driver that aims to serve as a reference for driver writers by serving as a template. It also validates the existing media DVB APIs, thus helping userspace application writers.

Currently, it consists of:

- A fake tuner driver, which will report a bad signal quality if the chosen frequency is too far away from a table of valid frequencies for a particular delivery system.
- A fake demod driver, which will constantly poll the fake signal quality returned by the tuner, simulating a device that can lose/reacquire a lock on the signal depending on the CNR levels.
- A fake bridge driver, which is the module responsible for modprobing the fake tuner and demod modules and implementing the demux logic. This module takes parameters at initialization that will dictate how the simulation behaves.
- Code responsible for encoding a valid MPEG Transport Stream, which is then passed to the bridge driver. This fake stream contains some hardcoded content. For now, we have a single, audio-only channel containing a single MPEG Elementary Stream, which in turn contains a SMPTE 302m encoded sine-wave. Note that this particular encoder was chosen because it is the easiest way to encode PCM audio data in a MPEG Transport Stream.

Building vidtv

vidtv is a test driver and thus is **not** enabled by default when compiling the kernel.

In order to enable compilation of vidtv:

- Enable **DVB_TEST_DRIVERS**, then
- Enable **DVB_VIDTV**

When compiled as a module, expect the following .ko files:

- `dvb_vidtv_tuner.ko`
- `dvb_vidtv_demod.ko`
- `dvb_vidtv_bridge.ko`

Running vidtv

When compiled as a module, run:

```
modprobe vidtv
```

That's it! The bridge driver will initialize the tuner and demod drivers as part of its own initialization.

By default, it will accept the following frequencies:

- 474 MHz for DVB-T/T2/C;
- 11,362 GHz for DVB-S/S2.

For satellite systems, the driver simulates an universal extended LNBf, with frequencies at Ku-Band, ranging from 10.7 GHz to 12.75 GHz.

You can optionally define some command-line arguments to vidtv.

Command-line arguments to vidtv

Below is a list of all arguments that can be supplied to vidtv:

drop_tslock_prob_on_low_snr Probability of losing the TS lock if the signal quality is bad. This probability be used by the fake demodulator driver to eventually return a status of 0 when the signal quality is not good.

recover_tslock_prob_on_good_snr: Probability recovering the TS lock when the signal improves. This probability be used by the fake demodulator driver to eventually return a status of 0x1f when/if the signal quality improves.

mock_power_up_delay_msec Simulate a power up delay. Default: 0.

mock_tune_delay_msec Simulate a tune delay. Default 0.

vidtv_valid_dvb_t_freqs Valid DVB-T frequencies to simulate, in Hz.

vidtv_valid_dvb_c_freqs Valid DVB-C frequencies to simulate, in Hz.

vidtv_valid_dvb_s_freqs Valid DVB-S/S2 frequencies to simulate at Ku-Band, in kHz.

max_frequency_shift_hz, Maximum shift in HZ allowed when tuning in a channel.

si_period_msec How often to send SI packets. Default: 40ms.

pcr_period_msec How often to send PCR packets. Default: 40ms.

mux_rate_kbytes_sec Attempt to maintain this bit rate by inserting TS null packets, if necessary. Default: 4096.

pcr_pid, PCR PID for all channels. Default: 0x200.

mux_buf_sz_pkts, Size for the mux buffer in multiples of 188 bytes.

vidtv internal structure

The kernel modules are split in the following way:

vidtv_tuner.[ch] Implements a fake tuner DVB driver.

vidtv_demod.[ch] Implements a fake demodulator DVB driver.

vidtv_bridge.[ch] Implements a bridge driver.

The MPEG related code is split in the following way:

vidtv_ts.[ch] Code to work with MPEG TS packets, such as TS headers, adaptation fields, PCR packets and NULL packets.

vidtv_psi.[ch] This is the PSI generator. PSI packets contain general information about a MPEG Transport Stream. A PSI generator is needed so userspace apps can retrieve information about the Transport Stream and eventually tune into a (dummy) channel.

Because the generator is implemented in a separate file, it can be reused elsewhere in the media subsystem.

Currently vidtv supports working with 5 PSI tables: PAT, PMT, SDT, NIT and EIT.

The specification for PAT and PMT can be found in *ISO 13818-1: Systems*, while the specification for the SDT, NIT, EIT can be found in *ETSI EN 300 468: Specification for Service Information (SI) in DVB systems*.

It isn't strictly necessary, but using a real TS file helps when debugging PSI tables. Vidtv currently tries to replicate the PSI structure found in this file: [TS1Globo.ts](#).

A good way to visualize the structure of streams is by using [DVBIInspector](#).

vidtv_pes.[ch] Implements the PES logic to convert encoder data into MPEG TS packets. These can then be fed into a TS multiplexer and eventually into userspace.

vidtv_encoder.h An interface for vidtv encoders. New encoders can be added to this driver by implementing the calls in this file.

vidtv_s302m.[ch] Implements a S302M encoder to make it possible to insert PCM audio data in the generated MPEG Transport Stream. The relevant specification is available online as *SMPTE 302M-2007: Television - Mapping of AES3 Data into MPEG-2 Transport Stream*.

The resulting MPEG Elementary Stream is conveyed in a private stream with a S302M registration descriptor attached.

This shall enable passing an audio signal into userspace so it can be decoded and played by media software. The corresponding decoder in ffmpeg is located in 'libavcodec/s302m.c' and is experimental.

vidtv_channel.[ch] Implements a 'channel' abstraction.

When vidtv boots, it will create some hardcoded channels:

1. Their services will be concatenated to populate the SDT.
2. Their programs will be concatenated to populate the PAT
3. Their events will be concatenated to populate the EIT
4. For each program in the PAT, a PMT section will be created
5. The PMT section for a channel will be assigned its streams.
6. Every stream will have its corresponding encoder polled in a loop to produce TS packets. These packets may be interleaved by the muxer and then delivered to the bridge.

vidtv_mux.[ch] Implements a MPEG TS mux, loosely based on the ffmpeg implementation in "libavcodec/mpegtsenc.c"

The muxer runs a loop which is responsible for:

1. Keeping track of the amount of time elapsed since the last iteration.
2. Polling encoders in order to fetch 'elapsed_time' worth of data.
3. Inserting PSI and/or PCR packets, if needed.
4. Padding the resulting stream with NULL packets if necessary in order to maintain the chosen bit rate.
5. Delivering the resulting TS packets to the bridge driver so it can pass them to the demux.

Testing vidtv with v4l-utils

Using the tools in v4l-utils is a great way to test and inspect the output of vidtv. It is hosted here: [v4l-utils Documentation](#).

From its webpage:

```
The v4l-utils are a series of packages for handling media devices.
```

```
It is hosted at http://git.linuxtv.org/v4l-utils.git, and packaged on most distributions.
```

```
It provides a series of libraries and utilities to be used to control several aspect of the media boards.
```

Start by installing v4l-utils and then modprobing vidtv:

```
modprobe dvb_vidtv_bridge
```

If the driver is OK, it should load and its probing code will run. This will pull in the tuner and demod drivers.

Using dvb-fe-tool

The first step to check whether the demod loaded successfully is to run:

```
$ dvb-fe-tool
Device Dummy demod for DVB-T/T2/C/S/S2 (/dev/dvb/adapter0/frontend0) capabilities:
  CAN_FEC_1_2
  CAN_FEC_2_3
  CAN_FEC_3_4
  CAN_FEC_4_5
  CAN_FEC_5_6
  CAN_FEC_6_7
  CAN_FEC_7_8
  CAN_FEC_8_9
  CAN_FEC_AUTO
  CAN_GUARD_INTERVAL_AUTO
  CAN_HIERARCHY_AUTO
  CAN_INVERSION_AUTO
  CAN_QAM_16
  CAN_QAM_32
  CAN_QAM_64
  CAN_QAM_128
  CAN_QAM_256
  CAN_QAM_AUTO
  CAN_QPSK
  CAN_TRANSMISSION_MODE_AUTO
DVB API Version 5.11, Current v5 delivery system: DVBC/ANNEX_A
Supported delivery systems:
  DVBT
  DVBT2
  [DVBC/ANNEX_A]
  DVBS
  DVBS2
Frequency range for the current standard:
From:          51.0 MHz
To:            2.15 GHz
Step:          62.5 kHz
Tolerance:     29.5 MHz
Symbol rate ranges for the current standard:
From:          1.00 MBauds
To:            45.0 MBauds
```

This should return what is currently set up at the demod struct, i.e.:

```
static const struct dvb_frontend_ops vidtv_demod_ops = {
    .delsys = {
        SYS_DVBT,
        SYS_DVBT2,
        SYS_DVBC_ANNEX_A,
        SYS_DVBS,
        SYS_DVBS2,
    },

    .info = {
        .name = "Dummy demod for DVB-T/T2/C/S/S2",
        .frequency_min_hz = 51 * MHz,
        .frequency_max_hz = 2150 * MHz,
```

```

.frequency_stepsize_hz = 62500,
.frequency_tolerance_hz = 29500 * kHz,
.symbol_rate_min       = 1000000,
.symbol_rate_max       = 45000000,

.caps = FE_CAN_FEC_1_2 |
        FE_CAN_FEC_2_3 |
        FE_CAN_FEC_3_4 |
        FE_CAN_FEC_4_5 |
        FE_CAN_FEC_5_6 |
        FE_CAN_FEC_6_7 |
        FE_CAN_FEC_7_8 |
        FE_CAN_FEC_8_9 |
        FE_CAN_QAM_16 |
        FE_CAN_QAM_64 |
        FE_CAN_QAM_32 |
        FE_CAN_QAM_128 |
        FE_CAN_QAM_256 |
        FE_CAN_QAM_AUTO |
        FE_CAN_QPSK |
        FE_CAN_FEC_AUTO |
        FE_CAN_INVERSION_AUTO |
        FE_CAN_TRANSMISSION_MODE_AUTO |
        FE_CAN_GUARD_INTERVAL_AUTO |
        FE_CAN_HIERARCHY_AUTO,
}

....

```

For more information on dvb-fe-tools check its online documentation here: [dvb-fe-tool Documentation](#).

Using dvb-scan

In order to tune into a channel and read the PSI tables, we can use dvb-scan.

For this, one should provide a configuration file known as a 'scan file', here's an example:

```

[Channel]
FREQUENCY = 474000000
MODULATION = QAM/AUTO
SYMBOL_RATE = 6940000
INNER_FEC = AUTO
DELIVERY_SYSTEM = DVBC/ANNEX_A

```

Note: The parameters depend on the video standard you're testing.

Note: Vidtv is a fake driver and does not validate much of the information in the scan file. Just specifying 'FREQUENCY' and 'DELIVERY_SYSTEM' should be enough for DVB-T/DVB-T2. For DVB-S/DVB-C however, you should also provide 'SYMBOL_RATE'.

You can browse scan tables online here: [dvb-scan-tables](#).

Assuming this channel is named 'channel.conf', you can then run:

```
$ dvbv5-scan channel.conf
dvvbv5-scan ~/vidtv.conf
ERROR    command BANDWIDTH_HZ (5) not found during retrieve
Cannot calc frequency shift. Either bandwidth/symbol-rate is unavailable (yet).
Scanning frequency #1 330000000
  (0x00) Signal= -68.00dBm
Scanning frequency #2 474000000
Lock    (0x1f) Signal= -34.45dBm C/N= 33.74dB UCB= 0
Service Beethoven, provider LinuxTV.org: digital television
```

For more information on dvb-scan, check its documentation online here: [dvb-scan Documentation](#).

Using dvb-zap

dvvbv5-zap is a command line tool that can be used to record MPEG-TS to disk. The typical use is to tune into a channel and put it into record mode. The example below - which is taken from the documentation - illustrates that¹:

```
$ dvbv5-zap -c dvb_channel.conf "beethoven" -o music.ts -P -t 10
using demux 'dvb0.demux0'
reading channels from file 'dvb_channel.conf'
tuning to 474000000 Hz
pass all PID's to TS
dvb_set_pesfilter 8192
dvb_dev_set_bufsize: buffer set to 6160384
Lock    (0x1f) Quality= Good Signal= -34.66dBm C/N= 33.41dB UCB= 0 postBER= 0 preBER= 1.
↳05x10^-3 PER= 0
Lock    (0x1f) Quality= Good Signal= -34.57dBm C/N= 33.46dB UCB= 0 postBER= 0 preBER= 1.
↳05x10^-3 PER= 0
Record to file 'music.ts' started
received 24587768 bytes (2401 Kbytes/sec)
Lock    (0x1f) Quality= Good Signal= -34.42dBm C/N= 33.89dB UCB= 0 postBER= 0 preBER= 2.
↳44x10^-3 PER= 0
```

The channel can be watched by playing the contents of the stream with some player that recognizes the MPEG-TS format, such as mplayer or vlc.

By playing the contents of the stream one can visually inspect the workings of vidtv, e.g., to play a recorded TS file with:

```
$ mplayer music.ts
```

or, alternatively, running this command on one terminal:

```
$ dvbv5-zap -c dvb_channel.conf "beethoven" -P -r &
```

And, on a second terminal, playing the contents from DVR interface with:

```
$ mplayer /dev/dvb/adaptor0/dvr0
```

¹ In this example, it records 10 seconds with all program ID's stored at the music.ts file.

For more information on dvb-zap check its online documentation here: [dvb-zap Documentation](#). See also: [zap](#).

What can still be improved in vidtv

Add *debugfs* integration

Although frontend drivers provide DVBv5 statistics via the `.read_status` call, a nice addition would be to make additional statistics available to userspace via `debugfs`, which is a simple-to-use, RAM-based filesystem specifically designed for debug purposes.

The logic for this would be implemented on a separate file so as not to pollute the frontend driver. These statistics are driver-specific and can be useful during tests.

The Siano driver is one example of a driver using `debugfs` to convey driver-specific statistics to userspace and it can be used as a reference.

This should be further enabled and disabled via a Kconfig option for convenience.

Add a way to test video

Currently, vidtv can only encode PCM audio. It would be great to implement a barebones version of MPEG-2 video encoding so we can also test video. The first place to look into is *ISO 13818-2: Information technology — Generic coding of moving pictures and associated audio information — Part 2: Video*, which covers the encoding of compressed video in MPEG Transport Streams.

This might optionally use the Video4Linux2 Test Pattern Generator, `v4l2-tpg`, which resides at:

```
drivers/media/common/v4l2-tpg/
```

Add white noise simulation

The vidtv tuner already has code to identify whether the chosen frequency is too far away from a table of valid frequencies. For now, this means that the demodulator can eventually lose the lock on the signal, since the tuner will report a bad signal quality.

A nice addition is to simulate some noise when the signal quality is bad by:

- Randomly dropping some TS packets. This will trigger a continuity error if the continuity counter is updated but the packet is not passed on to the demux.
- Updating the error statistics accordingly (e.g. BER, etc).
- Simulating some noise in the encoded data.

Functions and structs used within vidtv

struct **vidtv_dvb**
Vidtv bridge state

Definition

```
struct vidtv_dvb {
    struct platform_device *pdev;
    struct dvb_frontend *fe[NUM_FE];
    struct dvb_adapter adapter;
    struct dvb_demux demux;
    struct dmxdev dmx_dev;
    struct dmx_frontend dmx_fe[NUM_FE];
    struct i2c_adapter i2c_adapter;
    struct i2c_client *i2c_client_demod[NUM_FE];
    struct i2c_client *i2c_client_tuner[NUM_FE];
    u32 nfeeds;
    struct mutex feed_lock;
    bool streaming;
    struct vidtv_mux *mux;
#ifdef CONFIG_MEDIA_CONTROLLER_DVB;
    struct media_device mdev;
#endif ;
};
```

Members

pdev The platform device. Obtained when the bridge is probed.

fe The frontends. Obtained when probing the demodulator modules.

adapter Represents a DTV adapter. See 'dvb_register_adapter'.

demux The demux used by the *dvb_dmx_swfilter_packets()* call.

dmx_dev Represents a demux device.

dmx_fe The frontends associated with the demux.

i2c_adapter The i2c_adapter associated with the bridge driver.

i2c_client_demod The i2c_clients associated with the demodulator modules.

i2c_client_tuner The i2c_clients associated with the tuner modules.

nfeeds The number of feeds active.

feed_lock Protects access to the start/stop stream logic/data.

streaming Whether we are streaming now.

mux The abstraction responsible for delivering MPEG TS packets to the bridge.

mdev The media_device struct for media controller support.

struct **vidtv_channel**
A 'channel' abstraction

Definition

```

struct vidtv_channel {
    char *name;
    u16 transport_stream_id;
    struct vidtv_psi_table_sdt_service *service;
    u16 program_num;
    struct vidtv_psi_table_pat_program *program;
    struct vidtv_psi_table_pmt_stream *streams;
    struct vidtv_encoder *encoders;
    struct vidtv_psi_table_eit_event *events;
    struct vidtv_channel *next;
};

```

Members

name name of the channel

transport_stream_id a number to identify the TS, chosen at will.

service A `_single_service`. Will be concatenated into the SDT.

program_num The link between PAT, PMT and SDT.

program A `_single_program` with one or more streams associated with it. Will be concatenated into the PAT.

streams A stream loop used to populate the PMT section for 'program'

encoders A encoder loop. There must be one encoder for each stream.

events Optional event information. This will feed into the EIT.

next Optionally chain this channel.

Description

When vidtv boots, it will create some hardcoded channels. Their services will be concatenated to populate the SDT. Their programs will be concatenated to populate the PAT. For each program in the PAT, a PMT section will be created. The PMT section for a channel will be assigned its streams. Every stream will have its corresponding encoder polled to produce TS packets. These packets may be interleaved by the mux and then delivered to the bridge.

int **vidtv_channel_si_init**(struct *vidtv_mux* *m)
Init the PSI tables from the channels in the mux

Parameters

struct vidtv_mux *m The mux containing the channels.

int **vidtv_channels_init**(struct *vidtv_mux* *m)
Init hardcoded, fake 'channels'.

Parameters

struct vidtv_mux *m The mux to store the channels into.

struct **vidtv_demod_cnr_to_qual_s**
Map CNR values to a given combination of modulation and fec_inner

Definition

```
struct vidtv_demod_cnr_to_qual_s {
    u32 modulation;
    u32 fec;
    u32 cnr_ok;
    u32 cnr_good;
};
```

Members

modulation see enum fe_modulation

fec see enum fe_fec_rate

cnr_ok S/N threshold to consider the signal as OK. Below that, there's a chance of losing sync.

cnr_good S/N threshold to consider the signal strong.

Description

This struct matches values for 'good' and 'ok' CNRs given the combination of modulation and fec_inner in use. We might simulate some noise if the signal quality is not too good.

The values were taken from libdvbv5.

struct **vidtv_demod_config**
Configuration used to init the demod

Definition

```
struct vidtv_demod_config {
    u8 drop_tslock_prob_on_low_snr;
    u8 recover_tslock_prob_on_good_snr;
};
```

Members

drop_tslock_prob_on_low_snr probability of losing the lock due to low snr

recover_tslock_prob_on_good_snr probability of recovering when the signal improves

Description

The configuration used to init the demodulator module, usually filled by a bridge driver. For vidtv, this is filled by vidtv_bridge before the demodulator module is probed.

struct **vidtv_demod_state**
The demodulator state

Definition

```
struct vidtv_demod_state {
    struct dvb_frontend frontend;
    struct vidtv_demod_config config;
    enum fe_status status;
    u16 tuner_cnr;
};
```

Members

frontend The frontend structure allocated by the demod.

config The config used to init the demod.

status the demod status.

tuner_cnr current S/N ratio for the signal carrier

struct **vidtv_encoder**

A generic encoder type.

Definition

```

struct vidtv_encoder {
    enum vidtv_encoder_id id;
    char *name;
    u8 *encoder_buf;
    u32 encoder_buf_sz;
    u32 encoder_buf_offset;
    u64 sample_count;
    struct vidtv_access_unit *access_units;
    void *src_buf;
    u32 src_buf_sz;
    u32 src_buf_offset;
    bool is_video_encoder;
    void *ctx;
    __be16 stream_id;
    __be16 es_pid;
    void *(*encode)(struct vidtv_encoder *e);
    u32 (*clear)(struct vidtv_encoder *e);
    struct vidtv_encoder *sync;
    u32 sampling_rate_hz;
    void (*last_sample_cb)(u32 sample_no);
    void (*destroy)(struct vidtv_encoder *e);
    struct vidtv_encoder *next;
};

```

Members

id So we can cast to a concrete implementation when needed.

name Usually the same as the stream name.

encoder_buf The encoder internal buffer for the access units.

encoder_buf_sz The encoder buffer size, in bytes

encoder_buf_offset Our byte position in the encoder buffer.

sample_count How many samples we have encoded in total.

access_units encoder payload units, used for clock references

src_buf The source of raw data to be encoded, encoder might set a default if null.

src_buf_sz size of **src_buf**.

src_buf_offset Our position in the source buffer.

is_video_encoder Whether this a video encoder (as opposed to audio)

ctx Encoder-specific state.

stream_id Examples: Audio streams (0xc0-0xdf), Video streams (0xe0-0xef).

es_pid The TS PID to use for the elementary stream in this encoder.

encode Prepare enough AUs for the given amount of time.

clear Clear the encoder output.

sync Attempt to synchronize with this encoder.

sampling_rate_hz The sampling rate (or fps, if video) used.

last_sample_cb Called when the encoder runs out of data. This is so the source can read data in a piecemeal fashion instead of having to provide it all at once.

destroy Destroy this encoder, freeing allocated resources.

next Next in the chain

struct **vidtv_mux_timing**
Timing related information

Definition

```
struct vidtv_mux_timing {
    u64 start_jiffies;
    u64 current_jiffies;
    u64 past_jiffies;
    u64 clk;
    u64 pcr_period_usecs;
    u64 si_period_usecs;
};
```

Members

start_jiffies The value of 'jiffies' when we started the mux thread.

current_jiffies The value of 'jiffies' for the current iteration.

past_jiffies The value of 'jiffies' for the past iteration.

clk A 27Mhz clock from which we will drive the PCR. Updated proportionally on every iteration.

pcr_period_usecs How often we should send PCR packets.

si_period_usecs How often we should send PSI packets.

Description

This is used to decide when PCR or PSI packets should be sent. This will also provide storage for the clock, which is used to compute the value for the PCR.

struct **vidtv_mux_si**
Store the PSI context.

Definition

```
struct vidtv_mux_si {
    struct vidtv_psi_table_pat *pat;
    struct vidtv_psi_table_pmt **pmt_secs;
    struct vidtv_psi_table_sdt *sdt;
    struct vidtv_psi_table_nit *nit;
    struct vidtv_psi_table_eit *eit;
};
```

Members

pat The PAT in use by the muxer.

pmt_secs The PMT sections in use by the muxer. One for each program in the PAT.

sdt The SDT in use by the muxer.

nit The NIT in use by the muxer.

eit the EIT in use by the muxer.

Description

This is used to store the PAT, PMT sections and SDT in use by the muxer.

The muxer acquire these by looking into the hardcoded channels in vidtv_channel and then periodically sends the TS packets for them>

struct **vidtv_mux_pid_ctx**

Store the context for a given TS PID.

Definition

```
struct vidtv_mux_pid_ctx {
    u16 pid;
    u8 cc;
    struct hlist_node h;
};
```

Members

pid The TS PID.

cc The continuity counter for this PID. It is incremented on every TS pack and it will wrap around at 0xf0. If the decoder notices a sudden jump in this counter this will trigger a discontinuity state.

h This is embedded in a hash table, mapping pid -> vidtv_mux_pid_ctx

struct **vidtv_mux**

A muxer abstraction loosely based in libavcodec/mpegtsenc.c

Definition

```
struct vidtv_mux {
    struct dvb_frontend *fe;
    struct device *dev;
    struct vidtv_mux_timing timing;
    u32 mux_rate_kbytes_sec;
    unsigned long pid_ctx[1 << ((3) - 1)];
    void (*on_new_packets_available_cb)(void *priv, u8 *buf, u32 npackets);
    u8 *mux_buf;
    u32 mux_buf_sz;
    u32 mux_buf_offset;
    struct vidtv_channel *channels;
    struct vidtv_mux_si si;
    u64 num_streamed_pcr;
    u64 num_streamed_si;
    struct work_struct mpeg_thread;
    bool streaming;
    u16 pcr_pid;
    u16 transport_stream_id;
};
```

```
u16 network_id;
char *network_name;
void *priv;
};
```

Members

fe The frontend structure allocated by the muxer.

dev pointer to struct device.

timing Keeps track of timing related information.

mux_rate_kbytes_sec The bit rate for the TS, in kbytes.

pid_ctx A hash table to keep track of per-PID metadata.

on_new_packets_available_cb A callback to inform of new TS packets ready.

mux_buf A pointer to a buffer for this muxer. TS packets are stored there and then passed on to the bridge driver.

mux_buf_sz The size for 'mux_buf'.

mux_buf_offset The current offset into 'mux_buf'.

channels The channels associated with this muxer.

si Keeps track of the PSI context.

num_streamed_pcr Number of PCR packets streamed.

num_streamed_si The number of PSI packets streamed.

mpeg_thread Thread responsible for the muxer loop.

streaming whether 'mpeg_thread' is running.

pcr_pid The TS PID used for the PSI packets. All channels will share the same PCR.

transport_stream_id The transport stream ID

network_id The network ID

network_name The network name

priv Private data.

struct **vidtv_mux_init_args**
Arguments used to inix the muxer.

Definition

```
struct vidtv_mux_init_args {
    u32 mux_rate_kbytes_sec;
    void (*on_new_packets_available_cb)(void *priv, u8 *buf, u32 npackets);
    u32 mux_buf_sz;
    u64 pcr_period_usecs;
    u64 si_period_usecs;
    u16 pcr_pid;
    u16 transport_stream_id;
    struct vidtv_channel *channels;
    u16 network_id;
};
```

```
char *network_name;
void *priv;
};
```

Members

mux_rate_kbytes_sec The bit rate for the TS, in kbytes.

on_new_packets_available_cb A callback to inform of new TS packets ready.

mux_buf_sz The size for 'mux_buf'.

pcr_period_usecs How often we should send PCR packets.

si_period_usecs How often we should send PSI packets.

pcr_pid The TS PID used for the PSI packets. All channels will share the same PCR.

transport_stream_id The transport stream ID

channels an optional list of channels to use

network_id The network ID

network_name The network name

priv Private data.

struct **pes_header_write_args**

Arguments to write a PES header.

Definition

```
struct pes_header_write_args {
    void *dest_buf;
    u32 dest_offset;
    u32 dest_buf_sz;
    u32 encoder_id;
    bool send_pts;
    u64 pts;
    bool send_dts;
    u64 dts;
    u16 stream_id;
    u32 n_pes_h_s_bytes;
    u32 access_unit_len;
};
```

Members

dest_buf The buffer to write into.

dest_offset where to start writing in the dest_buffer.

dest_buf_sz The size of the dest_buffer

encoder_id Encoder id (see vidtv_encoder.h)

send_pts Should we send PTS?

pts PTS value to send.

send_dts Should we send DTS?

dts DTS value to send.

stream_id The stream id to use. Ex: Audio streams (0xc0-0xdf), Video streams (0xe0-0xef).

n_pes_h_s_bytes Padding bytes. Might be used by an encoder if needed, gets discarded by the decoder.

access_unit_len The size of `_one_` access unit (with any headers it might need)

struct **pes_ts_header_write_args**

Arguments to write a TS header.

Definition

```
struct pes_ts_header_write_args {
    void *dest_buf;
    u32 dest_offset;
    u32 dest_buf_sz;
    u16 pid;
    u8 *continuity_counter;
    bool wrote_pes_header;
    u32 n_stuffing_bytes;
    u64 pcr;
};
```

Members

dest_buf The buffer to write into.

dest_offset where to start writing in the `dest_buffer`.

dest_buf_sz The size of the `dest_buffer`

pid The PID to use for the TS packets.

continuity_counter Incremented on every new TS packet.

wrote_pes_header Flag to indicate that the PES header was written

n_stuffing_bytes Padding bytes. Might be used by an encoder if needed, gets discarded by the decoder.

pcr counter driven by a 27Mhz clock.

struct **pes_write_args**

Arguments for the packetizer.

Definition

```
struct pes_write_args {
    void *dest_buf;
    void *from;
    u32 access_unit_len;
    u32 dest_offset;
    u32 dest_buf_sz;
    u16 pid;
    u32 encoder_id;
    u8 *continuity_counter;
    u16 stream_id;
    bool send_pts;
    u64 pts;
    bool send_dts;
};
```

```

u64 dts;
u32 n_pes_h_s_bytes;
u64 pcr;
};

```

Members

dest_buf The buffer to write into.

from A pointer to the encoder buffer containing one access unit.

access_unit_len The size of `_one_` access unit (with any headers it might need)

dest_offset where to start writing in the `dest_buffer`.

dest_buf_sz The size of the `dest_buffer`

pid The PID to use for the TS packets.

encoder_id Encoder id (see `vidtv_encoder.h`)

continuity_counter Incremented on every new TS packet.

stream_id The stream id to use. Ex: Audio streams (0xc0-0xdf), Video streams (0xe0-0xef).

send_pts Should we send PTS?

pts PTS value to send.

send_dts Should we send DTS?

dts DTS value to send.

n_pes_h_s_bytes Padding bytes. Might be used by an encoder if needed, gets discarded by the decoder.

pcr counter driven by a 27Mhz clock.

u32 **vidtv_pes_write_into**(struct *pes_write_args* *args)
Write a PES packet as MPEG-TS packets into a buffer.

Parameters

struct pes_write_args *args The args to use when writing

Description

This function translate the ES data for one access unit from an encoder into MPEG TS packets. It does so by first encapsulating it with a PES header and then splitting it into TS packets.

The data is then written into the buffer pointed to by 'args.buf'

Return

The number of bytes written into the buffer. This is usually NOT equal to the size of the access unit, since we need space for PES headers, TS headers and padding bytes, if any.

struct **psi_write_args**
Arguments for the PSI packetizer.

Definition

```
struct psi_write_args {
    void *dest_buf;
    void *from;
    size_t len;
    u32 dest_offset;
    u16 pid;
    bool new_psi_section;
    u8 *continuity_counter;
    bool is_crc;
    u32 dest_buf_sz;
    u32 *crc;
};
```

Members

dest_buf The buffer to write into.

from PSI data to be copied.

len How much to write.

dest_offset where to start writing in the dest_buffer.

pid TS packet ID.

new_psi_section Set when starting a table section.

continuity_counter Incremented on every new packet.

is_crc Set when writing the CRC at the end.

dest_buf_sz The size of the dest_buffer

crc a pointer to store the crc for this chunk

struct **desc_write_args**

Arguments in order to write a descriptor.

Definition

```
struct desc_write_args {
    void *dest_buf;
    u32 dest_offset;
    struct vidtv_psi_desc *desc;
    u16 pid;
    u8 *continuity_counter;
    u32 dest_buf_sz;
    u32 *crc;
};
```

Members

dest_buf The buffer to write into.

dest_offset where to start writing in the dest_buffer.

desc A pointer to the descriptor

pid TS packet ID.

continuity_counter Incremented on every new packet.

dest_buf_sz The size of the dest_buffer

crc a pointer to store the crc for this chunk

struct **crc32_write_args**

Arguments in order to write the CRC at the end of the PSI tables.

Definition

```
struct crc32_write_args {
    void *dest_buf;
    u32 dest_offset;
    __be32 crc;
    u16 pid;
    u8 *continuity_counter;
    u32 dest_buf_sz;
};
```

Members

dest_buf The buffer to write into.

dest_offset where to start writing in the dest_buffer.

crc the CRC value to write

pid TS packet ID.

continuity_counter Incremented on every new packet.

dest_buf_sz The size of the dest_buffer

struct **header_write_args**

Arguments in order to write the common table header

Definition

```
struct header_write_args {
    void *dest_buf;
    u32 dest_offset;
    struct vidtv_psi_table_header *h;
    u16 pid;
    u8 *continuity_counter;
    u32 dest_buf_sz;
    u32 *crc;
};
```

Members

dest_buf The buffer to write into.

dest_offset where to start writing in the dest_buffer.

h a pointer to the header.

pid TS packet ID.

continuity_counter Incremented on every new packet.

dest_buf_sz The size of the dest_buffer

crc a pointer to store the crc for this chunk

```
void vidtv_psi_sdt_service_assign(struct vidtv_psi_table_sdt *sdt, struct
                                vidtv_psi_table_sdt_service *service)
    Assigns the service loop to the SDT.
```

Parameters

struct vidtv_psi_table_sdt *sdt The SDT to assign to.
struct vidtv_psi_table_sdt_service *service The service loop (one or more services)

Description

This will free the previous service loop in the table. This will assign ownership of the service loop to the table, i.e. the table will free this service loop when a call to its destroy function is made.

```
void vidtv_psi_desc_assign(struct vidtv_psi_desc **to, struct vidtv_psi_desc *desc)
    Assigns a descriptor loop at some point
```

Parameters

struct vidtv_psi_desc **to Where to assign this descriptor loop to
struct vidtv_psi_desc *desc The descriptor loop that will be assigned.

Description

This will free the loop in 'to', if any.

```
void vidtv_pmt_desc_assign(struct vidtv_psi_table_pmt *pmt, struct vidtv_psi_desc **to,
                           struct vidtv_psi_desc *desc)
    Assigns a descriptor loop at some point in a PMT section.
```

Parameters

struct vidtv_psi_table_pmt *pmt The PMT section that will contain the descriptor loop
struct vidtv_psi_desc **to Where in the PMT to assign this descriptor loop to
struct vidtv_psi_desc *desc The descriptor loop that will be assigned.

Description

This will free the loop in 'to', if any. This will assign ownership of the loop to the table, i.e. the table will free this loop when a call to its destroy function is made.

```
void vidtv_sdt_desc_assign(struct vidtv_psi_table_sdt *sdt, struct vidtv_psi_desc **to,
                           struct vidtv_psi_desc *desc)
    Assigns a descriptor loop at some point in a SDT.
```

Parameters

struct vidtv_psi_table_sdt *sdt The SDT that will contain the descriptor loop
struct vidtv_psi_desc **to Where in the PMT to assign this descriptor loop to
struct vidtv_psi_desc *desc The descriptor loop that will be assigned.

Description

This will free the loop in 'to', if any. This will assign ownership of the loop to the table, i.e. the table will free this loop when a call to its destroy function is made.

void **vidtv_psi_pat_program_assign**(struct vidtv_psi_table_pat *pat, struct vidtv_psi_table_pat_program *p)
 Assigns the program loop to the PAT.

Parameters

struct vidtv_psi_table_pat *pat The PAT to assign to.

struct vidtv_psi_table_pat_program *p The program loop (one or more programs)

Description

This will free the previous program loop in the table. This will assign ownership of the program loop to the table, i.e. the table will free this program loop when a call to its destroy function is made.

void **vidtv_psi_pmt_stream_assign**(struct vidtv_psi_table_pmt *pmt, struct vidtv_psi_table_pmt_stream *s)
 Assigns the stream loop to the PAT.

Parameters

struct vidtv_psi_table_pmt *pmt The PMT to assign to.

struct vidtv_psi_table_pmt_stream *s The stream loop (one or more streams)

Description

This will free the previous stream loop in the table. This will assign ownership of the stream loop to the table, i.e. the table will free this stream loop when a call to its destroy function is made.

struct vidtv_psi_table_pmt** **vidtv_psi_pmt_create_sec_for_each_pat_entry**(struct vidtv_psi_table_pat *pat, u16 pcr_pid)

Create a PMT section for each program found in the PAT

Parameters

struct vidtv_psi_table_pat *pat The PAT to look for programs.

u16 pcr_pid packet ID for the PCR to be used for the program described in this PMT section

u16 vidtv_psi_pmt_get_pid(struct vidtv_psi_table_pmt *section, struct vidtv_psi_table_pat *pat)
 Get the TS PID for a PMT section.

Parameters

struct vidtv_psi_table_pmt *section The PMT section whose PID we want to retrieve.

struct vidtv_psi_table_pat *pat The PAT table to look into.

Return

the TS PID for 'section'

void **vidtv_psi_pat_table_update_sec_len**(struct vidtv_psi_table_pat *pat)
 Recompute and update the PAT section length.

Parameters

struct vidtv_psi_table_pat *pat The PAT whose length is to be updated.

Description

This will traverse the table and accumulate the length of its components, which is then used to replace the 'section_length' field.

If section_length > MAX_SECTION_LEN, the operation fails.

void **vidtv_psi_pmt_table_update_sec_len**(struct vidtv_psi_table_pmt *pmt)
Recompute and update the PMT section length.

Parameters

struct vidtv_psi_table_pmt *pmt The PMT whose length is to be updated.

Description

This will traverse the table and accumulate the length of its components, which is then used to replace the 'section_length' field.

If section_length > MAX_SECTION_LEN, the operation fails.

void **vidtv_psi_sdt_table_update_sec_len**(struct vidtv_psi_table_sdt *sdt)
Recompute and update the SDT section length.

Parameters

struct vidtv_psi_table_sdt *sdt The SDT whose length is to be updated.

Description

This will traverse the table and accumulate the length of its components, which is then used to replace the 'section_length' field.

If section_length > MAX_SECTION_LEN, the operation fails.

struct **vidtv_psi_pat_write_args**
Arguments for writing a PAT table

Definition

```
struct vidtv_psi_pat_write_args {
    char *buf;
    u32 offset;
    struct vidtv_psi_table_pat *pat;
    u32 buf_sz;
    u8 *continuity_counter;
};
```

Members

buf The destination buffer.

offset The offset into the destination buffer.

pat A pointer to the PAT.

buf_sz The size of the destination buffer.

continuity_counter A pointer to the CC. Incremented on every new packet.

u32 **vidtv_psi_pat_write_into**(struct *vidtv_psi_pat_write_args* *args)
Write PAT as MPEG-TS packets into a buffer.

Parameters

struct vidtv_psi_pat_write_args *args An instance of *struct vidtv_psi_pat_write_args*

Description

This function writes the MPEG TS packets for a PAT table into a buffer. Calling code will usually generate the PAT via a call to its init function and thus is responsible for freeing it.

Return

The number of bytes written into the buffer. This is NOT equal to the size of the PAT, since more space is needed for TS headers during TS encapsulation.

struct vidtv_psi_sdt_write_args
Arguments for writing a SDT table

Definition

```
struct vidtv_psi_sdt_write_args {
    char *buf;
    u32 offset;
    struct vidtv_psi_table_sdt *sdt;
    u32 buf_sz;
    u8 *continuity_counter;
};
```

Members

buf The destination buffer.

offset The offset into the destination buffer.

sdt A pointer to the SDT.

buf_sz The size of the destination buffer.

continuity_counter A pointer to the CC. Incremented on every new packet.

u32 **vidtv_psi_sdt_write_into**(struct *vidtv_psi_sdt_write_args* *args)
Write SDT as MPEG-TS packets into a buffer.

Parameters

struct vidtv_psi_sdt_write_args *args an instance of *struct vidtv_psi_sdt_write_args*

Description

This function writes the MPEG TS packets for a SDT table into a buffer. Calling code will usually generate the SDT via a call to its init function and thus is responsible for freeing it.

Return

The number of bytes written into the buffer. This is NOT equal to the size of the SDT, since more space is needed for TS headers during TS encapsulation.

struct vidtv_psi_pmt_write_args
Arguments for writing a PMT section

Definition

```
struct vidtv_psi_pmt_write_args {
    char *buf;
    u32 offset;
    struct vidtv_psi_table_pmt *pmt;
    u16 pid;
    u32 buf_sz;
    u8 *continuity_counter;
    u16 pcr_pid;
};
```

Members

buf The destination buffer.

offset The offset into the destination buffer.

pmt A pointer to the PMT.

pid Program ID

buf_sz The size of the destination buffer.

continuity_counter A pointer to the CC. Incremented on every new packet.

pcr_pid The TS PID used for the PSI packets. All channels will share the same PCR.

u32 **vidtv_psi_pmt_write_into**(struct *vidtv_psi_pmt_write_args* *args)
Write PMT as MPEG-TS packets into a buffer.

Parameters

struct vidtv_psi_pmt_write_args *args an instance of *struct vidtv_psi_pmt_write_args*

Description

This function writes the MPEG TS packets for a PMT section into a buffer. Calling code will usually generate the PMT section via a call to its init function and thus is responsible for freeing it.

Return

The number of bytes written into the buffer. This is NOT equal to the size of the PMT section, since more space is needed for TS headers during TS encapsulation.

```
struct vidtv_psi_table_pmt * vidtv_psi_find_pmt_sec(struct vidtv_psi_table_pmt **pmt_sections,
                                                    u16 nsections,
                                                    u16 program_num)
```

Finds the PMT section for 'program_num'

Parameters

struct vidtv_psi_table_pmt **pmt_sections The sections to look into.

u16 nsections The number of sections.

u16 program_num The 'program_num' from PAT pointing to a PMT section.

Return

A pointer to the PMT, if found, or NULL.

struct vidtv_psi_table_transport

A entry in the TS loop for the NIT and/or other tables. See ETSI 300 468 section 5.2.1

Definition

```
struct vidtv_psi_table_transport {
    __be16 transport_id;
    __be16 network_id;
    __be16 bitfield;
    struct vidtv_psi_desc *descriptor;
    struct vidtv_psi_table_transport *next;
};
```

Members

transport_id The TS ID being described

network_id The network_id that contains the TS ID

bitfield Contains the descriptor loop length

descriptor A descriptor loop

next Pointer to the next entry

struct vidtv_psi_table_nit

A Network Information Table (NIT). See ETSI 300 468 section 5.2.1

Definition

```
struct vidtv_psi_table_nit {
    struct vidtv_psi_table_header header;
    __be16 bitfield;
    struct vidtv_psi_desc *descriptor;
    __be16 bitfield2;
    struct vidtv_psi_table_transport *transport;
};
```

Members

header A PSI table header

bitfield Contains the network descriptor length

descriptor A descriptor loop describing the network

bitfield2 Contains the transport stream loop length

transport The transport stream loop

struct vidtv_psi_nit_write_args

Arguments for writing a NIT section

Definition

```
struct vidtv_psi_nit_write_args {
    char *buf;
    u32 offset;
    struct vidtv_psi_table_nit *nit;
    u32 buf_sz;
    u8 *continuity_counter;
};
```

Members

buf The destination buffer.

offset The offset into the destination buffer.

nit A pointer to the NIT

buf_sz The size of the destination buffer.

continuity_counter A pointer to the CC. Incremented on every new packet.

u32 **vidtv_psi_nit_write_into**(struct *vidtv_psi_nit_write_args* *args)
Write NIT as MPEG-TS packets into a buffer.

Parameters

struct vidtv_psi_nit_write_args *args an instance of *struct vidtv_psi_nit_write_args*

Description

This function writes the MPEG TS packets for a NIT table into a buffer. Calling code will usually generate the NIT via a call to its init function and thus is responsible for freeing it.

Return

The number of bytes written into the buffer. This is NOT equal to the size of the NIT, since more space is needed for TS headers during TS encapsulation.

struct **vidtv_psi_eit_write_args**
Arguments for writing an EIT section

Definition

```
struct vidtv_psi_eit_write_args {  
    char *buf;  
    u32 offset;  
    struct vidtv_psi_table_eit *eit;  
    u32 buf_sz;  
    u8 *continuity_counter;  
};
```

Members

buf The destination buffer.

offset The offset into the destination buffer.

eit A pointer to the EIT

buf_sz The size of the destination buffer.

continuity_counter A pointer to the CC. Incremented on every new packet.

u32 **vidtv_psi_eit_write_into**(struct *vidtv_psi_eit_write_args* *args)
Write EIT as MPEG-TS packets into a buffer.

Parameters

struct vidtv_psi_eit_write_args *args an instance of *struct vidtv_psi_eit_write_args*

Description

This function writes the MPEG TS packets for a EIT table into a buffer. Calling code will usually generate the EIT via a call to its init function and thus is responsible for freeing it.

Return

The number of bytes written into the buffer. This is NOT equal to the size of the EIT, since more space is needed for TS headers during TS encapsulation.

void **vidtv_psi_eit_table_update_sec_len**(struct vidtv_psi_table_eit *eit)
 Recompute and update the EIT section length.

Parameters

struct vidtv_psi_table_eit *eit The EIT whose length is to be updated.

Description

This will traverse the table and accumulate the length of its components, which is then used to replace the 'section_length' field.

If section_length > EIT_MAX_SECTION_LEN, the operation fails.

void **vidtv_psi_eit_event_assign**(struct vidtv_psi_table_eit *eit, struct
 vidtv_psi_table_eit_event *e)
 Assigns the event loop to the EIT.

Parameters

struct vidtv_psi_table_eit *eit The EIT to assign to.

struct vidtv_psi_table_eit_event *e The event loop

Description

This will free the previous event loop in the table. This will assign ownership of the stream loop to the table, i.e. the table will free this stream loop when a call to its destroy function is made.

struct **vidtv_s302m_ctx**
 s302m encoder context.

Definition

```
struct vidtv_s302m_ctx {
    struct vidtv_encoder *enc;
    u32 frame_index;
    u32 au_count;
    int last_duration;
    unsigned int note_offset;
    enum musical_notes last_tone;
};
```

Members

enc A pointer to the containing encoder structure.

frame_index The current frame in a block

au_count The total number of access units encoded up to now

last_duration Duration of the tone currently being played

note_offset Position at the music tone array

last_tone Tone currently being played

struct **vidtv_s302m_encoder_init_args**
Args for the s302m encoder.

Definition

```
struct vidtv_s302m_encoder_init_args {
    char *name;
    void *src_buf;
    u32 src_buf_sz;
    u16 es_pid;
    struct vidtv_encoder *sync;
    void (*last_sample_cb)(u32 sample_no);
    struct vidtv_encoder *head;
};
```

Members

name A name to identify this particular instance

src_buf The source buffer, encoder will default to a sine wave if this is NULL.

src_buf_sz The size of the source buffer.

es_pid The MPEG Elementary Stream PID to use.

sync Attempt to synchronize audio with this video encoder, if not NULL.

last_sample_cb A callback called when the encoder runs out of data.

head Add to this chain

struct **pcr_write_args**
Arguments for the pcr_write_into function.

Definition

```
struct pcr_write_args {
    void *dest_buf;
    u32 dest_offset;
    u16 pid;
    u32 buf_sz;
    u8 *continuity_counter;
    u64 pcr;
};
```

Members

dest_buf The buffer to write into.

dest_offset The byte offset into the buffer.

pid The TS PID for the PCR packets.

buf_sz The size of the buffer in bytes.

continuity_counter The TS continuity_counter.

pcr A sample from the system clock.

struct **null_packet_write_args**
Arguments for the null_write_into function

Definition

```

struct null_packet_write_args {
    void *dest_buf;
    u32 dest_offset;
    u32 buf_sz;
    u8 *continuity_counter;
};

```

Members

dest_buf The buffer to write into.

dest_offset The byte offset into the buffer.

buf_sz The size of the buffer in bytes.

continuity_counter The TS continuity_counter.

u32 **vidtv_ts_null_write_into**(struct *null_packet_write_args* args)
Write a TS null packet into a buffer.

Parameters

struct null_packet_write_args args the arguments to use when writing.

Description

This function will write a null packet into a buffer. This is usually used to pad TS streams.

Return

The number of bytes written into the buffer.

u32 **vidtv_ts_pcr_write_into**(struct *pcr_write_args* args)
Write a PCR packet into a buffer.

Parameters

struct pcr_write_args args the arguments to use when writing.

Description

This function will write a PCR packet into a buffer. This is used to synchronize the clocks between encoders and decoders.

Return

The number of bytes written into the buffer.

struct **vidtv_tuner_config**
Configuration used to init the tuner.

Definition

```

struct vidtv_tuner_config {
    struct dvb_frontend *fe;
    u32 mock_power_up_delay_msec;
    u32 mock_tune_delay_msec;
    u32 vidtv_valid_dvb_t_freqs[NUM_VALID_TUNER_FREQS];
    u32 vidtv_valid_dvb_c_freqs[NUM_VALID_TUNER_FREQS];
    u32 vidtv_valid_dvb_s_freqs[NUM_VALID_TUNER_FREQS];
};

```

```
u8 max_frequency_shift_hz;
};
```

Members

fe A pointer to the `dvb_frontend` structure allocated by `vidtv_demod`.

mock_power_up_delay_msec Simulate a power-up delay.

mock_tune_delay_msec Simulate a tune delay.

vidtv_valid_dvb_t_freqs The valid DVB-T frequencies to simulate.

vidtv_valid_dvb_c_freqs The valid DVB-C frequencies to simulate.

vidtv_valid_dvb_s_freqs The valid DVB-S frequencies to simulate.

max_frequency_shift_hz The maximum frequency shift in HZ allowed when tuning in a channel

Description

The configuration used to init the tuner module, usually filled by a bridge driver. For `vidtv`, this is filled by `vidtv_bridge` before the tuner module is probed.

u32 **vidtv_memcpy**(void *to, size_t to_offset, size_t to_size, const void *from, size_t len)
wrapper routine to be used by MPEG-TS generator, in order to avoid going past the output buffer.

Parameters

void *to Starting element to where a MPEG-TS packet will be copied.

size_t to_offset Starting position of the **to** buffer to be filled.

size_t to_size Size of the **to** buffer.

const void *from Starting element of the buffer to be copied.

size_t len Number of elements to be copy from **from** buffer into **to**+ **to_offset** buffer.

Note

Real digital TV demod drivers should not have memcpy wrappers. We use it here because emulating MPEG-TS generation at kernelspace requires some extra care.

Return

Returns the number of bytes written

u32 **vidtv_memset**(void *to, size_t to_offset, size_t to_size, const int c, size_t len)
wrapper routine to be used by MPEG-TS generator, in order to avoid going past the output buffer.

Parameters

void *to Starting element to set

size_t to_offset Starting position of the **to** buffer to be filled.

size_t to_size Size of the **to** buffer.

const int c The value to set the memory to.

size_t len Number of elements to be copy from **from** buffer into **to**+ **to_offset** buffer.

Note

Real digital TV demod drivers should not have memset wrappers. We use it here because emulating MPEG-TS generation at kernelspace requires some extra care.

Return

Returns the number of bytes written

struct **vidtv_tuner_hardware_state**

Simulate the tuner hardware status

Definition

```
struct vidtv_tuner_hardware_state {
    bool asleep;
    u32 lock_status;
    u32 if_frequency;
    u32 tuned_frequency;
    u32 bandwidth;
};
```

Members

asleep whether the tuner is asleep, i.e whether `_sleep()` or `_suspend()` was called.

lock_status Whether the tuner has managed to lock on the requested frequency.

if_frequency The tuner's intermediate frequency. Hardcoded for the purposes of simulation.

tuned_frequency The actual tuned frequency.

bandwidth The actual bandwidth.

Description

This structure is meant to simulate the status of the tuner hardware, as if we had a physical tuner hardware.

struct **vidtv_tuner_dev**

The tuner struct

Definition

```
struct vidtv_tuner_dev {
    struct dvb_frontend *fe;
    struct vidtv_tuner_hardware_state hw_state;
    struct vidtv_tuner_config config;
};
```

Members

fe A pointer to the `dvb_frontend` structure allocated by `vidtv_demod`

hw_state A struct to simulate the tuner's hardware state as if we had a physical tuner hardware.

config The configuration used to start the tuner module, usually filled by a bridge driver. For `vidtv`, this is filled by `vidtv_bridge` before the tuner module is probed.

2.9.2.4 Contributors

Note: This documentation is outdated. There are several other DVB contributors that aren't listed below.

Thanks go to the following people for patches and contributions:

- Michael Hunold <m.hunold@gmx.de>
 - for the initial saa7146 driver and its recent overhaul
- Christian Theiss
 - for his work on the initial Linux DVB driver
- Marcus Metzler <mocm@metzlerbros.de> and Ralph Metzler <rjkm@metzlerbros.de>
 - for their continuing work on the DVB driver
- Michael Holzt <kju@debian.org>
 - for his contributions to the dvb-net driver
- Diego Picciani <d.picciani@novacomp.it>
 - for CyberLogin for Linux which allows logging onto EON (in case you are wondering where CyberLogin is, EON changed its login procedure and CyberLogin is no longer used.)
- Martin Schaller <martin@smurf.franken.de>
 - for patching the cable card decoder driver
- Klaus Schmidinger <Klaus.Schmidinger@cadsoft.de>
 - for various fixes regarding tuning, OSD and CI stuff and his work on VDR
- Steve Brown <sbrown@cortland.com>
 - for his AFC kernel thread
- Christoph Martin <martin@uni-mainz.de>
 - for his LIRC infrared handler
- Andreas Oberritter <obi@linuxtv.org>, Dennis Noermann <dennis.noermann@noernet.de>, Felix Domke <tmbinc@elitedvb.net>, Florian Schirmer <jolt@tuxbox.org>, Ronny Strutz <3des@elitedvb.de>, Wolfram Joost <dbox2@frokaschwei.de> and all the other dbox2 people
 - for many bugfixes in the generic DVB Core, frontend drivers and their work on the dbox2 port of the DVB driver
- Oliver Endriss <o.endriss@gmx.de>
 - for many bugfixes
- Andrew de Quincey <adq_dvb@lidskialf.net>
 - for the tda1004x frontend driver, and various bugfixes
- Peter Schildmann <peter.schildmann@web.de>

- for the driver for the Technisat SkyStar2 PCI DVB card
- Vadim Catana <skystar@moldova.cc>, Roberto Ragusa <r.ragusa@libero.it> and Augusto Cardoso <augusto@carhil.net>
 - for all the work for the FlexCopII chipset by B2C2, Inc.
- Davor Emard <emard@softhome.net>
 - for his work on the budget drivers, the demux code, the module unloading problems, ...
- Hans-Frieder Vogt <hfvogt@arcor.de>
 - for his work on calculating and checking the crc's for the TechnoTrend/Hauppage DEC driver firmware
- Michael Dreher <michael@5dot1.de> and Andreas 'randy' Weinberger
 - for the support of the Fujitsu-Siemens Activy budget DVB-S
- Kenneth Aafløy <ke-aa@frisurf.no>
 - for adding support for Typhoon DVB-S budget card
- Ernst Peinlich <e.peinlich@inode.at>
 - for tuning/DiSEqC support for the DEC 3000-s
- Peter Beutner <p.beutner@gmx.net>
 - for the IR code for the ttusb-dec driver
- Wilson Michaels <wilsonmichaels@earthlink.net>
 - for the lgdt330x frontend driver, and various bugfixes
- Michael Krufky <mkrufky@linuxtv.org>
 - for maintaining v4l/dvb inter-tree dependencies
- Taylor Jacob <rtjacob@earthlink.net>
 - for the nxt2002 frontend driver
- Jean-Francois Thibert <jeanfrancois@sagetv.com>
 - for the nxt2004 frontend driver
- Kirk Lapray <kirk.lapray@gmail.com>
 - for the or51211 and or51132 frontend drivers, and for merging the nxt2002 and nxt2004 modules into a single nxt200x frontend driver.

(If you think you should be in this list, but you are not, drop a line to the DVB mailing list)

Copyright © 2009-2020 : LinuxTV Developers

This documentation is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or

FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

For more details see the file COPYING in the source distribution of Linux.

LINUX MEDIA INFRASTRUCTURE USERSPACE API

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

Please see:

Documentation/admin-guide/media/index.rst

- for usage information about media subsystem and supported drivers;

Documentation/driver-api/media/index.rst

- for driver development information and Kernel APIs used by media devices;

3.1 Introduction

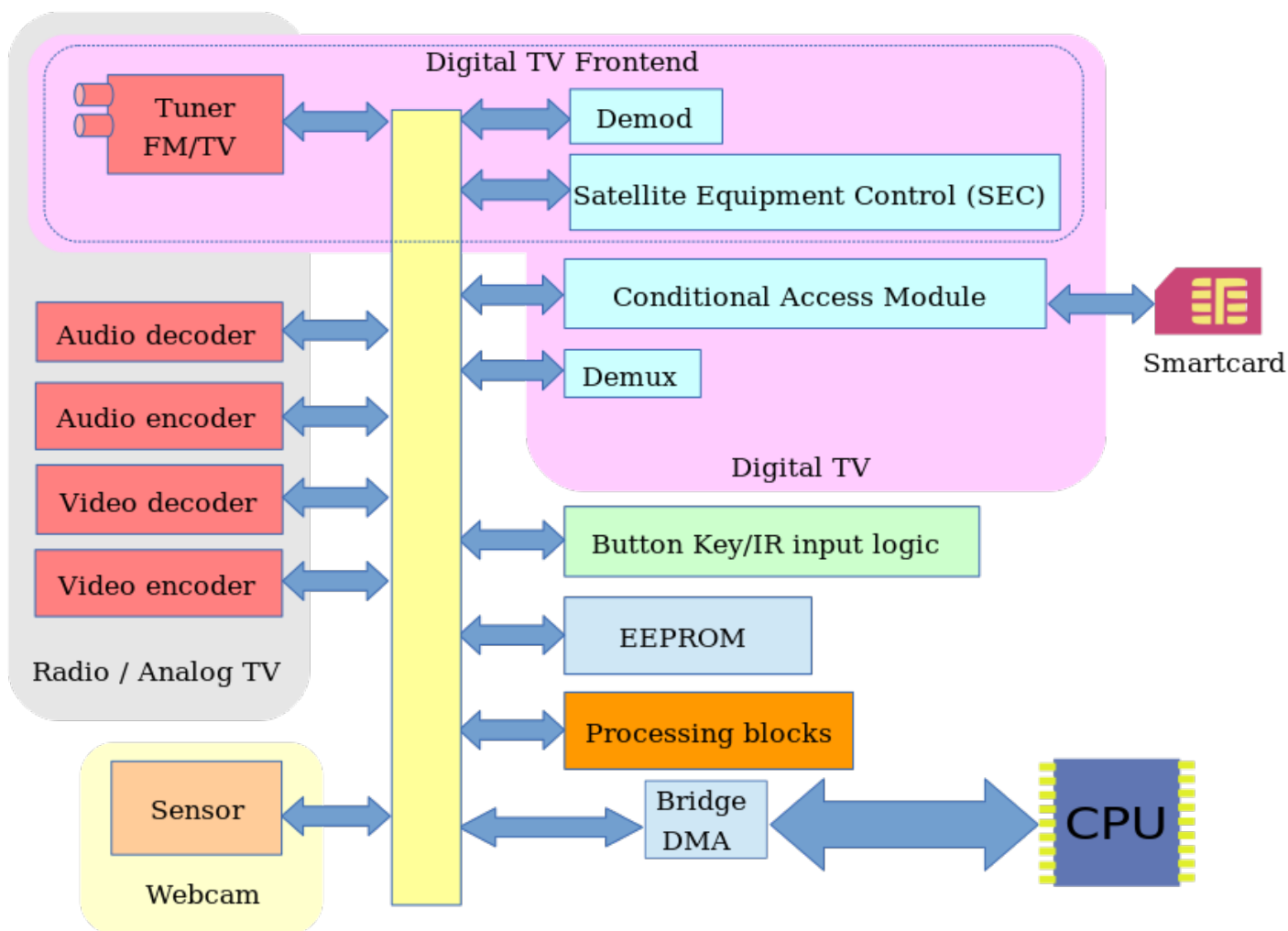
This document covers the Linux Kernel to Userspace API's used by video and radio streaming devices, including video cameras, analog and digital TV receiver cards, AM/FM receiver cards, Software Defined Radio (SDR), streaming capture and output devices, codec devices and remote controllers.

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

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

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

It should also be noted that a media device may also have audio components, like mixers, PCM capture, PCM playback, etc, which are controlled via ALSA API. For additional information and for the latest development code, see: <https://linuxtv.org>. For discussing improvements, reporting troubles, sending new drivers, etc, please mail to: [Linux Media Mailing List \(LMML\)](#).



PS.: picture is not complete: other blocks may be present

Fig. 1: Typical Media Device

3.2 Part I - Video for Linux API

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

Revision 4.5

3.2.1 Common API Elements

Programming a V4L2 device consists of these steps:

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

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

3.2.1.1 Opening and Closing Devices

Controlling a hardware peripheral via V4L2

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

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

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

When such V4L2 sub-devices are exposed, they allow controlling those other hardware components - usually connected via a serial bus (like I²C, SMBus or SPI). Depending on the bridge driver, those sub-devices can be controlled indirectly via the bridge driver or explicitly via the *Media Controller* and via the *V4L2 sub-devices*.

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

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

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

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

Note:

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

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

V4L2 Device Node Naming

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

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

The device nodes supported by the Video4Linux subsystem are:

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

Where X is a non-negative integer.

Note:

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

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

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

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

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

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

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

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

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

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

Related Devices

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

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

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

Today each V4L2 device node supports just one function.

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

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

Multiple Opens

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

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

Merely opening a V4L2 device does not grant exclusive access.⁴ Initiating data exchange however assigns the right to read or write the requested type of data, and to change related properties, to this file descriptor. Applications can request additional access privileges using the priority mechanism described in *Application Priority*.

Shared Data Streams

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

Functions

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

3.2.1.2 Querying Capabilities

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

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

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

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

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

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

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

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

3.2.1.3 Application Priority

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

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

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

3.2.1.4 Video Inputs and Outputs

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

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

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

Example: Information about the current video input

```
struct v4l2_input input;
int index;

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

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

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

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

Example: Switching to the first video input

```
int index;

index = 0;

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

3.2.1.5 Audio Inputs and Outputs

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

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

To learn about the number and attributes of the available inputs and outputs applications can enumerate them with the `ioctl VIDIOC_ENUMAUDIO` and `VIDIOC_ENUMAUDOUT` `ioctl`, re-

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

spectively. The struct `v4l2_audio` returned by the `ioctl VIDIOC_ENUMAUDIO` ioctl also contains signal status information applicable when the current audio input is queried.

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

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

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

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

Example: Information about the current audio input

```
struct v4l2_audio audio;

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

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

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

Example: Switching to the first audio input

```
struct v4l2_audio audio;

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

audio.index = 0;

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

3.2.1.6 Tuners and Modulators

Tuners

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

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

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

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

Modulators

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

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

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

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

Radio Frequency

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

3.2.1.7 Video Standards

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

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

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

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

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

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

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

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

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

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

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

Example: Information about the current video standard

```
v4l2_std_id std_id;
struct v4l2_standard standard;

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

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

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

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

    standard.index++;
}

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

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

Example: Listing the video standards supported by the current input

```
struct v4l2_input input;
struct v4l2_standard standard;

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

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

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

```

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

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

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

    standard.index++;
}

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

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

```

Example: Selecting a new video standard

```

struct v4l2_input input;
v4l2_std_id std_id;

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

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

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

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

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

std_id = V4L2_STD_PAL_BG;

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

```

3.2.1.8 Digital Video (DV) Timings

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

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

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

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

3.2.1.9 User Controls

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

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

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

Drivers may enumerate different controls after switching the current video input or output, tuner or modulator, or audio input or output. Different in the sense of other bounds, another

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

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

default and current value, step size or other menu items. A control with a certain *custom* ID can also change name and type.

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

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

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

All controls use machine endianness.

Control IDs

V4L2_CID_BASE First predefined ID, equal to `V4L2_CID_BRIGHTNESS`.

V4L2_CID_USER_BASE Synonym of `V4L2_CID_BASE`.

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

V4L2_CID_CONTRAST (integer) Picture contrast or luma gain.

V4L2_CID_SATURATION (integer) Picture color saturation or chroma gain.

V4L2_CID_HUE (integer) Hue or color balance.

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

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

V4L2_CID_AUDIO_BASS (integer) Audio bass adjustment.

V4L2_CID_AUDIO_TREBLE (integer) Audio treble adjustment.

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

V4L2_CID_AUDIO_LOUDNESS (boolean) Loudness mode (bass boost).

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

V4L2_CID_AUTO_WHITE_BALANCE (boolean) Automatic white balance (cameras).

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

V4L2_CID_RED_BALANCE (integer) Red chroma balance.

V4L2_CID_BLUE_BALANCE (integer) Blue chroma balance.

V4L2_CID_GAMMA (integer) Gamma adjust.

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

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

V4L2_CID_AUTOGAIN (boolean) Automatic gain/exposure control.

V4L2_CID_GAIN (integer) Gain control.

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

V4L2_CID_HFLIP (boolean) Mirror the picture horizontally.

V4L2_CID_VFLIP (boolean) Mirror the picture vertically.

V4L2_CID_POWER_LINE_FREQUENCY (enum) Enables a power line frequency filter to avoid flicker. Possible values for enum `v4l2_power_line_frequency` are:

<code>V4L2_CID_POWER_LINE_FREQUENCY_DISABLED</code>	0
<code>V4L2_CID_POWER_LINE_FREQUENCY_50HZ</code>	1
<code>V4L2_CID_POWER_LINE_FREQUENCY_60HZ</code>	2
<code>V4L2_CID_POWER_LINE_FREQUENCY_AUTO</code>	3

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

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

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

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

V4L2_CID_CHROMA_AGC (boolean) Chroma automatic gain control.

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

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

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

V4L2_COLORFX_NONE	Color effect is disabled.
V4L2_COLORFX_ANTIQUA	An aging (old photo) effect.
V4L2_COLORFX_ART_FREEZE	Frost color effect.
V4L2_COLORFX_AQUA	Water color, cool tone.
V4L2_COLORFX_BW	Black and white.
V4L2_COLORFX_EMOSS	Emboss, the highlights and shadows replace light/dark boundaries and low contrast areas are set to a gray background.
V4L2_COLORFX_GRASS_GREEN	Grass green.
V4L2_COLORFX_NEGATIVE	Negative.
V4L2_COLORFX_SEPIA	Sepia tone.
V4L2_COLORFX_SKETCH	Sketch.
V4L2_COLORFX_SKIN_WHITEN	Skin whiten.
V4L2_COLORFX_SKY_BLUE	Sky blue.
V4L2_COLORFX_SOLARIZATION	Solarization, the image is partially reversed in tone, only color values above or below a certain threshold are inverted.
V4L2_COLORFX_SILHOUETTE	Silhouette (outline).
V4L2_COLORFX_VIVID	Vivid colors.
V4L2_COLORFX_SET_CBCR	The Cb and Cr chroma components are replaced by fixed coefficients determined by V4L2_CID_COLORFX_CBCR control.
V4L2_COLORFX_SET_RGB	The RGB components are replaced by the fixed RGB components determined by V4L2_CID_COLORFX_RGB control.

V4L2_CID_COLORFX_RGB (integer) Determines the Red, Green, and Blue coefficients for V4L2_COLORFX_SET_RGB color effect. Bits [7:0] of the supplied 32 bit value are interpreted as Blue component, bits [15:8] as Green component, bits [23:16] as Red component, and bits [31:24] must be zero.

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

V4L2_CID_AUTOBRIGHTNESS (boolean) Enable Automatic Brightness.

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

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

V4L2_CID_ILLUMINATORS_1 V4L2_CID_ILLUMINATORS_2 (boolean) Switch on or off the illuminator 1 or 2 of the device (usually a microscope).

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

V4L2_CID_MIN_BUFFERS_FOR_OUTPUT (integer) This is a read-only control that can be read by

the application and used as a hint to determine the number of OUTPUT buffers to pass to REQBUFS. The value is the minimum number of OUTPUT buffers that is necessary for hardware to work.

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

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

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

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

Example: Enumerating all controls

```
struct v4l2_queryctrl queryctrl;
struct v4l2_querymenu querymenu;

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

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

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

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

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

        if (queryctrl.type == V4L2_CTRL_TYPE_MENU)
```

```

        enumerate_menu(queryctrl.id);
    }

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

```

Example: Enumerating all controls including compound controls

```

struct v4l2_query_ext_ctrl query_ext_ctrl;

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

query_ext_ctrl.id = V4L2_CTRL_FLAG_NEXT_CTRL | V4L2_CTRL_FLAG_NEXT_COMPOUND;
while (0 == ioctl(fd, VIDIOC_QUERY_EXT_CTRL, &query_ext_ctrl)) {
    if (!(query_ext_ctrl.flags & V4L2_CTRL_FLAG_DISABLED)) {
        printf("Control %s\\n", query_ext_ctrl.name);

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

    query_ext_ctrl.id |= V4L2_CTRL_FLAG_NEXT_CTRL | V4L2_CTRL_FLAG_NEXT_COMPOUND;
}
if (errno != EINVAL) {
    perror("VIDIOC_QUERY_EXT_CTRL");
    exit(EXIT_FAILURE);
}

```

Example: Enumerating all user controls (old style)

```

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

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

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

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

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

```

```

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

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

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

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

```

Example: Changing controls

```

struct v4l2_queryctrl queryctrl;
struct v4l2_control control;

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

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

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

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

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

```

```

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

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

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

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

```

3.2.1.10 Extended Controls API

Introduction

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

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

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

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

The Extended Control API

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

Each of the new ioctls expects a pointer to a struct `v4l2_ext_controls`. This structure contains a pointer to the control array, a count of the number of controls in that array and a control class. Control classes are used to group similar controls into a single class. For example, control class `V4L2_CTRL_CLASS_USER` contains all user controls (i. e. all controls that can also be set using the old *VIDIOC_S_CTRL* ioctl). Control class `V4L2_CTRL_CLASS_CODEC` contains controls relating to codecs.

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

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

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

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

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

Note:

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

All controls use machine endianness.

Enumerating Extended Controls

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

```
struct v4l2_queryctrl qctrl;

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

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

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

```
qctrl.id = V4L2_CTRL_CLASS_CODEC | V4L2_CTRL_FLAG_NEXT_CTRL;
while (0 == ioctl (fd, VIDIOC_QUERYCTRL, &qctrl)) {
    if (V4L2_CTRL_ID2CLASS(qctrl.id) != V4L2_CTRL_CLASS_CODEC)
        break;
    /* ... */
    qctrl.id |= V4L2_CTRL_FLAG_NEXT_CTRL;
}
```

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

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

Creating Control Panels

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

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

3.2.1.11 Camera Control Reference

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

Camera Control IDs

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

V4L2_CID_EXPOSURE_AUTO (enum)

enum v4l2_exposure_auto_type - Enables automatic adjustments of the exposure time and/or iris aperture. The effect of manual changes of the exposure time or iris aperture while these features are enabled is undefined, drivers should ignore such requests. Possible values are:

V4L2_EXPOSURE_AUTO	Automatic exposure time, automatic iris aperture.
V4L2_EXPOSURE_MANUAL	Manual exposure time, manual iris.
V4L2_EXPOSURE_SHUTTER_PRIORITY	Manual exposure time, auto iris.
V4L2_EXPOSURE_APERTURE_PRIORITY	Auto exposure time, manual iris.

V4L2_CID_EXPOSURE_ABSOLUTE (integer) Determines the exposure time of the camera sensor. The exposure time is limited by the frame interval. Drivers should interpret the values as 100 μ s units, where the value 1 stands for 1/10000th of a second, 10000 for 1 second and 100000 for 10 seconds.

V4L2_CID_EXPOSURE_AUTO_PRIORITY (boolean) When *V4L2_CID_EXPOSURE_AUTO* is set to *AUTO* or *APERTURE_PRIORITY*, this control determines if the device may dynamically vary the frame rate. By default this feature is disabled (0) and the frame rate must remain constant.

V4L2_CID_AUTO_EXPOSURE_BIAS (integer menu) Determines the automatic exposure compensation, it is effective only when *V4L2_CID_EXPOSURE_AUTO* control is set to *AUTO*, *SHUTTER_PRIORITY* or *APERTURE_PRIORITY*. It is expressed in terms of EV, drivers should interpret the values as 0.001 EV units, where the value 1000 stands for +1 EV.

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

V4L2_CID_EXPOSURE_METERING (enum)

enum v4l2_exposure_metering - Determines how the camera measures the amount of light available for the frame exposure. Possible values are:

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

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

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

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

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

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

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

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

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

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

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

ignore such requests.

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

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

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

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

V4L2_CID_AUTO_FOCUS_RANGE (enum)

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

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

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

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

V4L2_CID_ZOOM_CONTINUOUS (integer) Move the objective lens group at the specified speed until it reaches physical device limits or until an explicit request to stop the movement. A positive value moves the zoom lens group towards the telephoto direction. A value of zero stops the zoom lens group movement. A negative value moves the zoom lens group towards the wide-angle direction. The zoom speed unit is driver-specific.

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

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

V4L2_CID_PRIVACY (boolean) Prevent video from being acquired by the camera. When this control is set to TRUE (1), no image can be captured by the camera. Common means to enforce privacy are mechanical obturation of the sensor and firmware image processing,

but the device is not restricted to these methods. Devices that implement the privacy control must support read access and may support write access.

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

V4L2_CID_AUTO_N_PRESET_WHITE_BALANCE (enum)

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

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

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

V4L2_CID_IMAGE_STABILIZATION (boolean) Enables or disables image stabilization.

V4L2_CID_ISO_SENSITIVITY (integer menu) Determines ISO equivalent of an image sensor indicating the sensor's sensitivity to light. The numbers are expressed in arithmetic scale, as per *ISO 12232:2006* standard, where doubling the sensor sensitivity is represented

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

by doubling the numerical ISO value. Applications should interpret the values as standard ISO values multiplied by 1000, e.g. control value 800 stands for ISO 0.8. Drivers will usually support only a subset of standard ISO values. The effect of setting this control while the `V4L2_CID_ISO_SENSITIVITY_AUTO` control is set to a value other than `V4L2_CID_ISO_SENSITIVITY_MANUAL` is undefined, drivers should ignore such requests.

V4L2_CID_ISO_SENSITIVITY_AUTO (enum)

enum v4l2_iso_sensitivity_type - Enables or disables automatic ISO sensitivity adjustments.

<code>V4L2_CID_ISO_SENSITIVITY_MANUAL</code>	Manual ISO sensitivity.
<code>V4L2_CID_ISO_SENSITIVITY_AUTO</code>	Automatic ISO sensitivity adjustments.

V4L2_CID_SCENE_MODE (enum)

enum v4l2_scene_mode - This control allows to select scene programs as the camera automatic modes optimized for common shooting scenes. Within these modes the camera determines best exposure, aperture, focusing, light metering, white balance and equivalent sensitivity. The controls of those parameters are influenced by the scene mode control. An exact behavior in each mode is subject to the camera specification.

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

<code>V4L2_SCENE_MODE_NONE</code>	The scene mode feature is disabled.
<code>V4L2_SCENE_MODE_BACKLIGHT</code>	Backlight. Compensates for dark shadows when light is coming from behind a subject, also by automatically turning on the flash.
<code>V4L2_SCENE_MODE_BEACH_SNOW</code>	Beach and snow. This mode compensates for all-white or bright scenes, which tend to look gray and low contrast, when camera's automatic exposure is based on an average scene brightness. To compensate, this mode automatically slightly overexposes the frames. The white balance may also be adjusted to compensate for the fact that reflected snow looks bluish rather than white.
<code>V4L2_SCENE_MODE_CANDLELIGHT</code>	Candle light. The camera generally raises the ISO sensitivity and lowers the shutter speed. This mode compensates for relatively close subject in the scene. The flash is disabled in order to preserve the ambiance of the light.
<code>V4L2_SCENE_MODE_DAWN_DUSK</code>	Dawn and dusk. Preserves the colors seen in low natural light before dusk and after dawn. The camera may turn off the flash, and automatically focus at infinity. It will usually boost saturation and lower the shutter speed.
<code>V4L2_SCENE_MODE_FALL_COLORS</code>	Fall colors. Increases saturation and adjusts white balance for color enhancement. Pictures of autumn leaves get saturated reds and yellows.
<code>V4L2_SCENE_MODE_FIREWORKS</code>	Fireworks. Long exposure times are used to capture the expanding burst of light from a firework. The camera may invoke image stabilization.
<code>V4L2_SCENE_MODE_LANDSCAPE</code>	Landscape. The camera may choose a small aperture to provide deep depth of field and long exposure duration to help capture detail in dim light conditions. The focus is fixed at infinity. Suitable for distant and wide scenery.

Continued on next page

Table 1 - continued from previous page

V4L2_SCENE_MODE_NIGHT	Night, also known as Night Landscape. Designed for low light conditions, it preserves detail in the dark areas without blowing out bright objects. The camera generally sets itself to a medium-to-high ISO sensitivity, with a relatively long exposure time, and turns flash off. As such, there will be increased image noise and the possibility of blurred image.
V4L2_SCENE_MODE_PARTY_INDOOR	Party and indoor. Designed to capture indoor scenes that are lit by indoor background lighting as well as the flash. The camera usually increases ISO sensitivity, and adjusts exposure for the low light conditions.
V4L2_SCENE_MODE PORTRAIT	Portrait. The camera adjusts the aperture so that the depth of field is reduced, which helps to isolate the subject against a smooth background. Most cameras recognize the presence of faces in the scene and focus on them. The color hue is adjusted to enhance skin tones. The intensity of the flash is often reduced.
V4L2_SCENE_MODE_SPORTS	Sports. Significantly increases ISO and uses a fast shutter speed to freeze motion of rapidly-moving subjects. Increased image noise may be seen in this mode.
V4L2_SCENE_MODE_SUNSET	Sunset. Preserves deep hues seen in sunsets and sunrises. It bumps up the saturation.
V4L2_SCENE_MODE_TEXT	Text. It applies extra contrast and sharpness, it is typically a black-and-white mode optimized for readability. Automatic focus may be switched to close-up mode and this setting may also involve some lens-distortion correction.

V4L2_CID_3A_LOCK (bitmask) This control locks or unlocks the automatic focus, exposure and white balance. The automatic adjustments can be paused independently by setting the corresponding lock bit to 1. The camera then retains the settings until the lock bit is cleared. The following lock bits are defined:

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

<code>V4L2_LOCK_EXPOSURE</code>	Automatic exposure adjustments lock.
<code>V4L2_LOCK_WHITE_BALANCE</code>	Automatic white balance adjustments lock.
<code>V4L2_LOCK_FOCUS</code>	Automatic focus lock.

V4L2_CID_PAN_SPEED (integer) This control turns the camera horizontally at the specific speed. The unit is undefined. A positive value moves the camera to the right (clockwise when viewed from above), a negative value to the left. A value of zero stops the motion if one is in progress and has no effect otherwise.

V4L2_CID_TILT_SPEED (integer) This control turns the camera vertically at the specified speed. The unit is undefined. A positive value moves the camera up, a negative value down. A value of zero stops the motion if one is in progress and has no effect otherwise.

V4L2_CID_CAMERA_ORIENTATION (menu) This read-only control describes the camera orientation by reporting its mounting position on the device where the camera is installed. The control value is constant and not modifiable by software. This control is particularly meaningful for devices which have a well defined orientation, such as phones, laptops and portable devices since the control is expressed as a position relative to the device's intended usage orientation. For example, a camera installed on the user-facing side of a phone, a tablet or a laptop device is said to be have `V4L2_CAMERA_ORIENTATION_FRONT` ori-

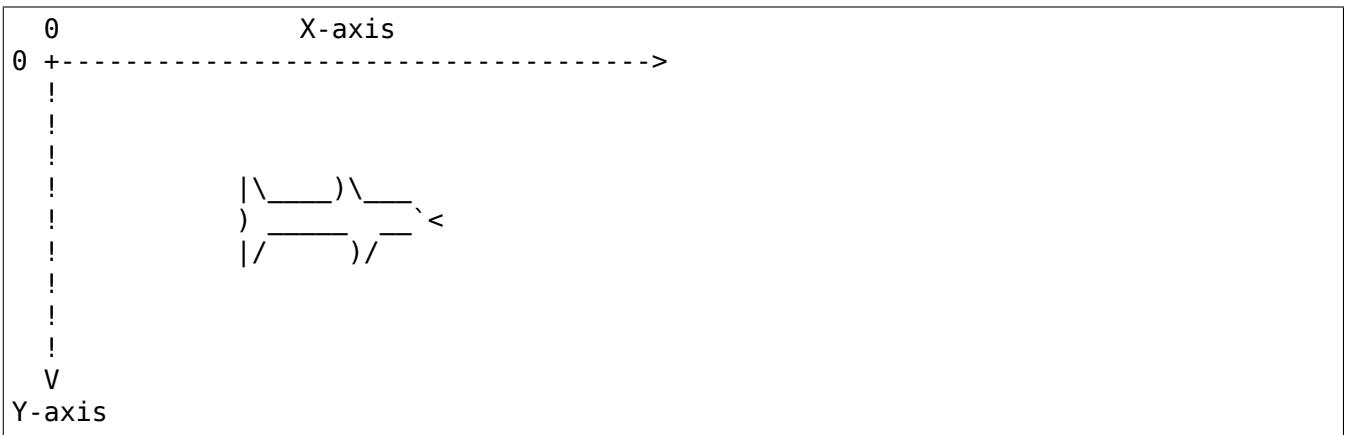
entation, while a camera installed on the opposite side of the front one is said to be have V4L2_CAMERA_ORIENTATION_BACK orientation. Camera sensors not directly attached to the device, or attached in a way that allows them to move freely, such as webcams and digital cameras, are said to have the V4L2_CAMERA_ORIENTATION_EXTERNAL orientation.

V4L2_CAMERA_ORIENTATION_FRONT	The camera is oriented towards the user facing side of the device.
V4L2_CAMERA_ORIENTATION_BACK	The camera is oriented towards the back facing side of the device.
V4L2_CAMERA_ORIENTATION_EXTERNAL	The camera is not directly attached to the device and is freely movable.

V4L2_CID_CAMERA_SENSOR_ROTATION (integer) This read-only control describes the rotation correction in degrees in the counter-clockwise direction to be applied to the captured images once captured to memory to compensate for the camera sensor mounting rotation.

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

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

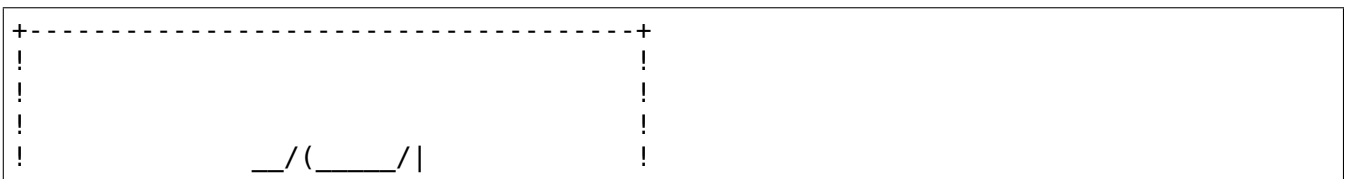


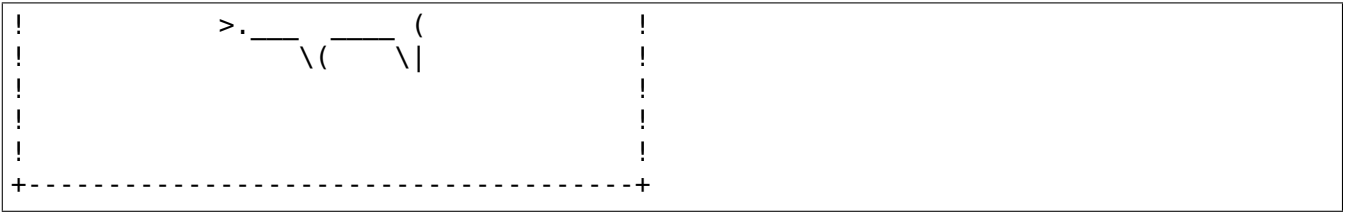
Example one - Webcam

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

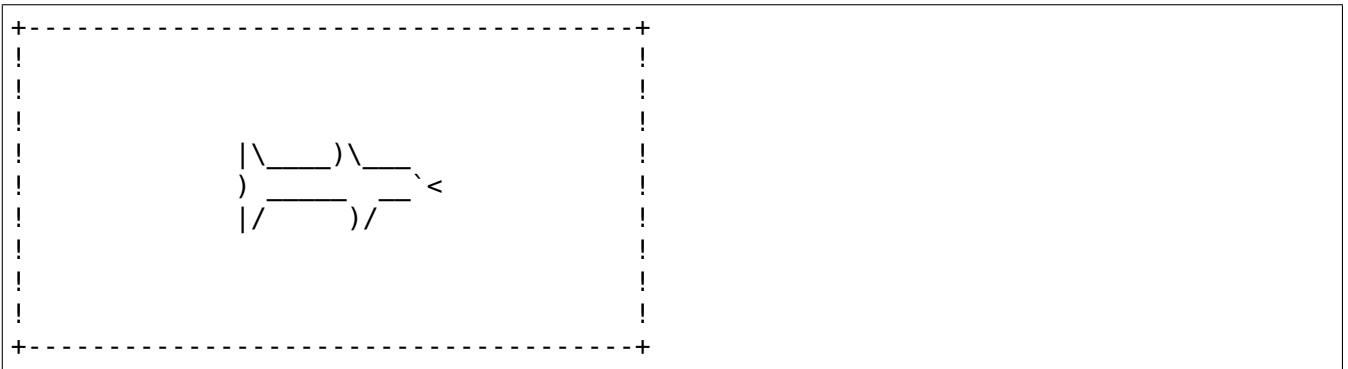
The camera is typically mounted upside-down to compensate the lens optical inversion effect. In this case the value of the V4L2_CID_CAMERA_SENSOR_ROTATION control is 0, no rotation is required to display images correctly to the user.

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





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

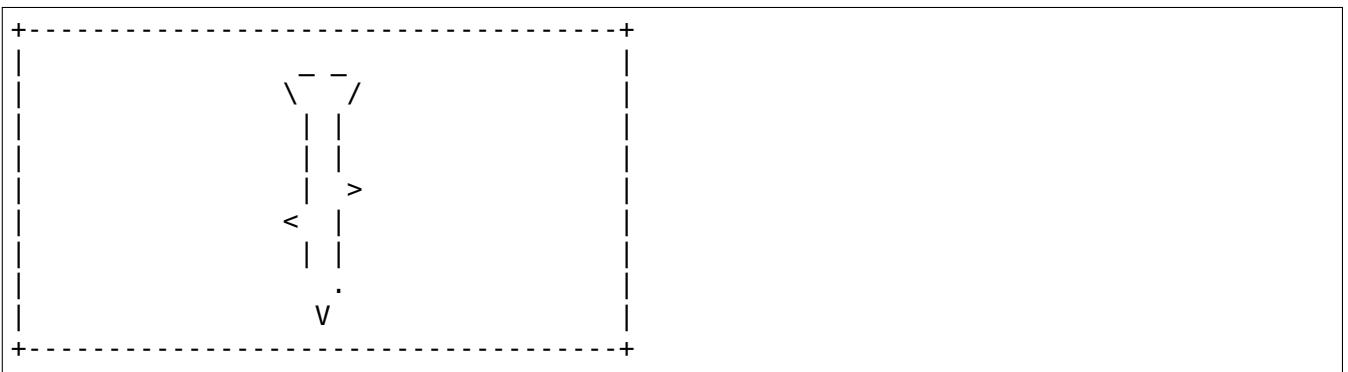


Example two - Phone camera

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

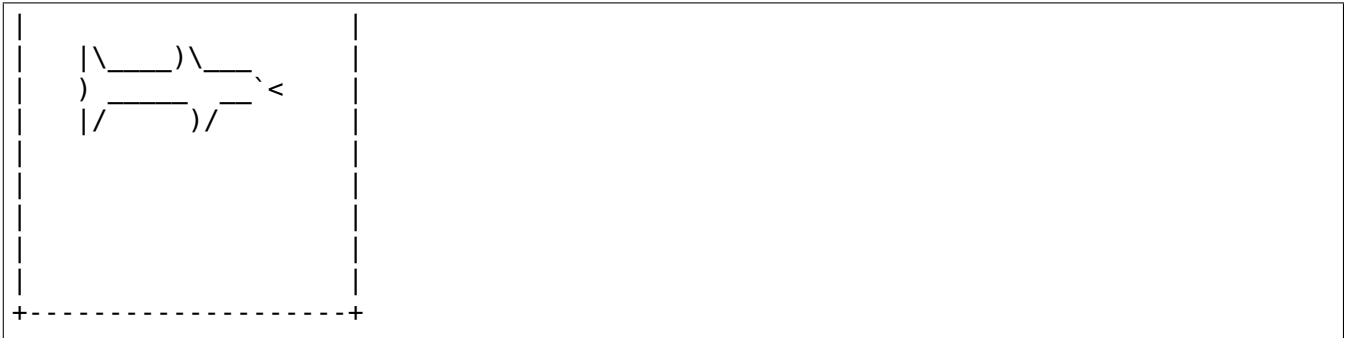
The camera sensor is typically mounted with its pixel array longer side aligned to the device longer side, upside-down mounted to compensate for the lens optical inversion effect.

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



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





3.2.1.12 Flash Control Reference

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

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

Supported use cases

Unsynchronised LED flash (software strobe)

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

Example of such device: Nokia N900.

Synchronised LED flash (hardware strobe)

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

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

LED flash as torch

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

Flash Control IDs

V4L2_CID_FLASH_CLASS (class) The FLASH class descriptor.

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

V4L2_FLASH_LED_MODE_NONE	Off.
V4L2_FLASH_LED_MODE_FLASH	Flash mode.
V4L2_FLASH_LED_MODE_TORCH	Torch mode. See V4L2_CID_FLASH_TORCH_INTENSITY.

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

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

V4L2_CID_FLASH_STROBE (button) Strobe flash. Valid when V4L2_CID_FLASH_LED_MODE is set to V4L2_FLASH_LED_MODE_FLASH and V4L2_CID_FLASH_STROBE_SOURCE is set to V4L2_FLASH_STROBE_SOURCE_SOFTWARE. Setting this control may not be possible in presence of some faults. See V4L2_CID_FLASH_FAULT.

V4L2_CID_FLASH_STROBE_STOP (button) Stop flash strobe immediately.

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

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

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

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

V4L2_CID_FLASH_INDICATOR_INTENSITY (integer) Intensity of the indicator LED. The indicator LED may be fully independent of the flash LED. The unit should be microamps (uA) if possible.

V4L2_CID_FLASH_FAULT (bitmask) Faults related to the flash. The faults tell about specific problems in the flash chip itself or the LEDs attached to it. Faults may prevent further use of some of the flash controls. In particular, V4L2_CID_FLASH_LED_MODE is set to V4L2_FLASH_LED_MODE_NONE if the fault affects the flash LED. Exactly which faults

have such an effect is chip dependent. Reading the faults resets the control and returns the chip to a usable state if possible.

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

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

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

3.2.1.13 Image Source Control Reference

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

Image Source Control IDs

V4L2_CID_IMAGE_SOURCE_CLASS (class) The IMAGE_SOURCE class descriptor.

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

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

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

V4L2_CID_TEST_PATTERN_RED (integer) Test pattern red colour component.

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

V4L2_CID_TEST_PATTERN_BLUE (integer) Test pattern blue colour component.

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

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

v4l2_area

Table 2: struct v4l2_area

__u32	width	Width of the area.
__u32	height	Height of the area.

V4L2_CID_NOTIFY_GAINS (integer array) The sensor is notified what gains will be applied to the different colour channels by subsequent processing (such as by an ISP). The sensor is merely informed of these values in case it performs processing that requires them, but it does not apply them itself to the output pixels.

Currently it is defined only for Bayer sensors, and is an array control taking 4 gain values, being the gains for each of the Bayer channels. The gains are always in the order B, Gb, Gr and R, irrespective of the exact Bayer order of the sensor itself.

The use of an array allows this control to be extended to sensors with, for example, non-Bayer CFAs (colour filter arrays).

The units for the gain values are linear, with the default value representing a gain of exactly 1.0. For example, if this default value is reported as being (say) 128, then a value of 192 would represent a gain of exactly 1.5.

3.2.1.14 Image Process Control Reference

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

Image Process Control IDs

V4L2_CID_IMAGE_PROC_CLASS (class) The IMAGE_PROC class descriptor.

V4L2_CID_LINK_FREQ (integer menu) The frequency of the data bus (e.g. parallel or CSI-2).

V4L2_CID_PIXEL_RATE (64-bit integer) Pixel sampling rate in the device's pixel array. This control is read-only and its unit is pixels / second.

Some devices use horizontal and vertical blanking to configure the frame rate. The frame rate can be calculated from the pixel rate, analogue crop rectangle as well as horizontal and vertical blanking. The pixel rate control may be present in a different sub-device than the blanking controls and the analogue crop rectangle configuration.

The configuration of the frame rate is performed by selecting the desired horizontal and vertical blanking. The unit of this control is Hz.

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

V4L2_CID_DEINTERLACING_MODE (menu) The video deinterlacing mode (such as Bob, Weave, ...). The menu items are driver specific and are documented in *Video4Linux (V4L) driver-specific documentation*.

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

3.2.1.15 Codec Control Reference

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

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

Generic Codec Controls

Codec Control IDs

V4L2_CID_CODEC_CLASS (class) The Codec class descriptor. Calling *ioctl VIDIOC_QUERYCTRL*, *VIDIOC_QUERY_EXT_CTRL* and *VIDIOC_QUERYMENU* for this control will return a description of this control class. This description can be used as the caption of a Tab page in a GUI, for example.

V4L2_CID_MPEG_STREAM_TYPE (enum)

enum v4l2_mpeg_stream_type - The MPEG-1, -2 or -4 output stream type. One cannot assume anything here. Each hardware MPEG encoder tends to support different subsets of

the available MPEG stream types. This control is specific to multiplexed MPEG streams. The currently defined stream types are:

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

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

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

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

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

V4L2_CID_MPEG_STREAM_PES_ID_AUDIO (integer) Audio ID for MPEG PES

V4L2_CID_MPEG_STREAM_PES_ID_VIDEO (integer) Video ID for MPEG PES

V4L2_CID_MPEG_STREAM_VBI_FMT (enum)

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

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

V4L2_CID_MPEG_AUDIO_SAMPLING_FREQ (enum)

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

V4L2_MPEG_AUDIO_SAMPLING_FREQ_44100	44.1 kHz
V4L2_MPEG_AUDIO_SAMPLING_FREQ_48000	48 kHz
V4L2_MPEG_AUDIO_SAMPLING_FREQ_32000	32 kHz

V4L2_CID_MPEG_AUDIO_ENCODING (enum)

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

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

V4L2_CID_MPEG_AUDIO_L1_BITRATE (enum)

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

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

V4L2_CID_MPEG_AUDIO_L2_BITRATE (enum)

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

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

V4L2_CID_MPEG_AUDIO_L3_BITRATE (enum)

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

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

V4L2_CID_MPEG_AUDIO_AAC_BITRATE (**integer**) AAC bitrate in bits per second.

V4L2_CID_MPEG_AUDIO_AC3_BITRATE (**enum**)

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

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

V4L2_CID_MPEG_AUDIO_MODE (**enum**)

enum v4l2_mpeg_audio_mode - MPEG Audio mode. Possible values are:

V4L2_MPEG_AUDIO_MODE_STEREO	Stereo
V4L2_MPEG_AUDIO_MODE_JOINT_STEREO	Joint Stereo
V4L2_MPEG_AUDIO_MODE_DUAL	Bilingual
V4L2_MPEG_AUDIO_MODE_MONO	Mono

V4L2_CID_MPEG_AUDIO_MODE_EXTENSION (**enum**)

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

V4L2_MPEG_AUDIO_MODE_EXTENSION_BOUND_4	Subbands 4-31 in intensity stereo
V4L2_MPEG_AUDIO_MODE_EXTENSION_BOUND_8	Subbands 8-31 in intensity stereo
V4L2_MPEG_AUDIO_MODE_EXTENSION_BOUND_12	Subbands 12-31 in intensity stereo
V4L2_MPEG_AUDIO_MODE_EXTENSION_BOUND_16	Subbands 16-31 in intensity stereo

V4L2_CID_MPEG_AUDIO_EMPHASIS (enum)

enum v4l2_mpeg_audio_emphasis - Audio Emphasis. Possible values are:

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

V4L2_CID_MPEG_AUDIO_CRC (enum)

enum v4l2_mpeg_audio_crc - CRC method. Possible values are:

V4L2_MPEG_AUDIO_CRC_NONE	None
V4L2_MPEG_AUDIO_CRC_CRC16	16 bit parity check

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

V4L2_CID_MPEG_AUDIO_DEC_PLAYBACK (enum)

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

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

V4L2_CID_MPEG_AUDIO_DEC_MULTILINGUAL_PLAYBACK (enum)

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

V4L2_CID_MPEG_VIDEO_ENCODING (enum)

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

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

V4L2_CID_MPEG_VIDEO_ASPECT (enum)

enum v4l2_mpeg_video_aspect - Video aspect. Possible values are:

V4L2_MPEG_VIDEO_ASPECT_1x1
V4L2_MPEG_VIDEO_ASPECT_4x3
V4L2_MPEG_VIDEO_ASPECT_16x9
V4L2_MPEG_VIDEO_ASPECT_221x100

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

V4L2_CID_MPEG_VIDEO_GOP_SIZE (integer) GOP size (default 12)

V4L2_CID_MPEG_VIDEO_GOP_CLOSURE (boolean) GOP closure (default 1)

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

V4L2_CID_MPEG_VIDEO_BITRATE_MODE (enum)

enum v4l2_mpeg_video_bitrate_mode - Video bitrate mode. Possible values are:

V4L2_MPEG_VIDEO_BITRATE_MODE_VBR	Variable bitrate
V4L2_MPEG_VIDEO_BITRATE_MODE_CBR	Constant bitrate
V4L2_MPEG_VIDEO_BITRATE_MODE_CQ	Constant quality

V4L2_CID_MPEG_VIDEO_BITRATE (integer) Average video bitrate in bits per second.

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

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

V4L2_CID_MPEG_VIDEO_FRAME_SKIP_MODE (enum)

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

V4L2_MPEG_VIDEO_FRAME_SKIP_MODE_DISABLED	Frame skip mode is disabled.
V4L2_MPEG_VIDEO_FRAME_SKIP_MODE_LEVEL_LIMIT	Frame skip mode enabled and buffer limit is set by the chosen level and is defined by the standard.
V4L2_MPEG_VIDEO_FRAME_SKIP_MODE_BUF_LIMIT	Frame skip mode enabled and buffer limit is set by the <i>VBV (MPEG1/2/4)</i> or <i>CPB (H264) buffer size</i> control.

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

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

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

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

V4L2_CID_MPEG_VIDEO_DECPTS (integer64) This read-only control returns the 33-bit video Presentation Time Stamp as defined in ITU T-REC-H.222.0 and ISO/IEC 13818-1 of the currently displayed frame. This is the same PTS as is used in *ioctl VIDIOC_DECODER_CMD*, *VIDIOC_TRY_DECODER_CMD*.

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

V4L2_CID_MPEG_VIDEO_DEC_CONCEAL_COLOR (integer64) This control sets the conceal color in YUV color space. It describes the client preference of the error conceal color in case of an error where the reference frame is missing. The decoder should fill the reference buffer with the preferred color and use it for future decoding. The control is using 16 bits per channel. Applicable to decoders.

	8bit format	10bit format	12bit format
Y luminance	Bit 0:7	Bit 0:9	Bit 0:11
Cb chrominance	Bit 16:23	Bit 16:25	Bit 16:27
Cr chrominance	Bit 32:39	Bit 32:41	Bit 32:43
Must be zero	Bit 48:63	Bit 48:63	Bit 48:63

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

V4L2_CID_MPEG_VIDEO_DEC_DISPLAY_DELAY_ENABLE (boolean) If the display delay is enabled then the decoder is forced to return a CAPTURE buffer (decoded frame) after processing a certain number of OUTPUT buffers. The delay can be set through *V4L2_CID_MPEG_VIDEO_DEC_DISPLAY_DELAY*. This feature can be used for example for generating thumbnails of videos. Applicable to the decoder.

V4L2_CID_MPEG_VIDEO_DEC_DISPLAY_DELAY (integer) Display delay value for decoder. The decoder is forced to return a decoded frame after the set ‘display delay’ number of frames. If this number is low it may result in frames returned out of display order, in addition the hardware may still be using the returned buffer as a reference picture for subsequent frames.

V4L2_CID_MPEG_VIDEO_AU_DELIMITER (boolean) If enabled then, AUD (Access Unit Delimiter) NALUs will be generated. That could be useful to find the start of a frame without having to fully parse each NALU. Applicable to the H264 and HEVC encoders.

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

V4L2_CID_MPEG_VIDEO_H264_VUI_SAR_IDC (enum)

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

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

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

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

V4L2_CID_MPEG_VIDEO_H264_LEVEL (enum)

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

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

V4L2_CID_MPEG_VIDEO_MPEG2_LEVEL (enum)

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

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

V4L2_CID_MPEG_VIDEO_MPEG4_LEVEL (enum)

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

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

V4L2_CID_MPEG_VIDEO_H264_PROFILE (enum)

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

V4L2_MPEG_VIDEO_H264_PROFILE_BASELINE	Baseline profile
V4L2_MPEG_VIDEO_H264_PROFILE_CONSTRAINED_BASELINE	Constrained Baseline profile
V4L2_MPEG_VIDEO_H264_PROFILE_MAIN	Main profile
V4L2_MPEG_VIDEO_H264_PROFILE_EXTENDED	Extended profile
V4L2_MPEG_VIDEO_H264_PROFILE_HIGH	High profile
V4L2_MPEG_VIDEO_H264_PROFILE_HIGH_10	High 10 profile
V4L2_MPEG_VIDEO_H264_PROFILE_HIGH_422	High 422 profile
V4L2_MPEG_VIDEO_H264_PROFILE_HIGH_444_PREDICTIVE	High 444 Predictive profile
V4L2_MPEG_VIDEO_H264_PROFILE_HIGH_10_INTRA	High 10 Intra profile
V4L2_MPEG_VIDEO_H264_PROFILE_HIGH_422_INTRA	High 422 Intra profile
V4L2_MPEG_VIDEO_H264_PROFILE_HIGH_444_INTRA	High 444 Intra profile
V4L2_MPEG_VIDEO_H264_PROFILE_CAVLC_444_INTRA	CAVLC 444 Intra profile
V4L2_MPEG_VIDEO_H264_PROFILE_SCALABLE_BASELINE	Scalable Baseline profile
V4L2_MPEG_VIDEO_H264_PROFILE_SCALABLE_HIGH	Scalable High profile
V4L2_MPEG_VIDEO_H264_PROFILE_SCALABLE_HIGH_INTRA	Scalable High Intra profile
V4L2_MPEG_VIDEO_H264_PROFILE_STEREO_HIGH	Stereo High profile
V4L2_MPEG_VIDEO_H264_PROFILE_MULTIVIEW_HIGH	Multiview High profile
V4L2_MPEG_VIDEO_H264_PROFILE_CONSTRAINED_HIGH	Constrained High profile

V4L2_CID_MPEG_VIDEO_MPEG2_PROFILE (enum)

enum v4l2_mpeg_video_mpeg2_profile - The profile information for MPEG2. Applicable to MPEG2 codecs. Possible values are:

V4L2_MPEG_VIDEO_MPEG2_PROFILE_SIMPLE	Simple profile (SP)
V4L2_MPEG_VIDEO_MPEG2_PROFILE_MAIN	Main profile (MP)
V4L2_MPEG_VIDEO_MPEG2_PROFILE_SNR_SCALABLE	SNR Scalable profile (SNR)
V4L2_MPEG_VIDEO_MPEG2_PROFILE_SPATIALLY_SCALABLE	Spatially Scalable profile (Spt)
V4L2_MPEG_VIDEO_MPEG2_PROFILE_HIGH	High profile (HP)
V4L2_MPEG_VIDEO_MPEG2_PROFILE_MULTIVIEW	Multi-view profile (MVP)

V4L2_CID_MPEG_VIDEO_MPEG4_PROFILE (enum)

enum v4l2_mpeg_video_mpeg4_profile - The profile information for MPEG4. Applicable to the MPEG4 encoder. Possible values are:

V4L2_MPEG_VIDEO_MPEG4_PROFILE_SIMPLE	Simple profile
V4L2_MPEG_VIDEO_MPEG4_PROFILE_ADVANCED_SIMPLE	Advanced Simple profile
V4L2_MPEG_VIDEO_MPEG4_PROFILE_CORE	Core profile
V4L2_MPEG_VIDEO_MPEG4_PROFILE_SIMPLE_SCALABLE	Simple Scalable profile
V4L2_MPEG_VIDEO_MPEG4_PROFILE_ADVANCED_CODING_EFFICIENCY	Advanced Coding Efficiency profile

V4L2_CID_MPEG_VIDEO_MAX_REF_PIC (integer) The maximum number of reference pictures used for encoding. Applicable to the encoder.

V4L2_CID_MPEG_VIDEO_MULTI_SLICE_MODE (enum)

enum v4l2_mpeg_video_multi_slice_mode - Determines how the encoder should handle division of frame into slices. Applicable to the encoder. Possible values are:

V4L2_MPEG_VIDEO_MULTI_SLICE_MODE_SINGLE	Single slice per frame.
V4L2_MPEG_VIDEO_MULTI_SLICE_MODE_MAX_MB	Multiple slices with set maximum number of macroblocks per slice.
V4L2_MPEG_VIDEO_MULTI_SLICE_MODE_MAX_BYTES	Multiple slice with set maximum size in bytes per slice.

V4L2_CID_MPEG_VIDEO_MULTI_SLICE_MAX_MB (integer) The maximum number of macroblocks in a slice. Used when V4L2_CID_MPEG_VIDEO_MULTI_SLICE_MODE is set to V4L2_MPEG_VIDEO_MULTI_SLICE_MODE_MAX_MB. Applicable to the encoder.

V4L2_CID_MPEG_VIDEO_MULTI_SLICE_MAX_BYTES (integer) The maximum size of a slice in bytes. Used when V4L2_CID_MPEG_VIDEO_MULTI_SLICE_MODE is set to V4L2_MPEG_VIDEO_MULTI_SLICE_MODE_MAX_BYTES. Applicable to the encoder.

V4L2_CID_MPEG_VIDEO_H264_LOOP_FILTER_MODE (enum)

enum v4l2_mpeg_video_h264_loop_filter_mode - Loop filter mode for H264 encoder. Possible values are:

V4L2_MPEG_VIDEO_H264_LOOP_FILTER_MODE_ENABLED	Loop filter is enabled.
V4L2_MPEG_VIDEO_H264_LOOP_FILTER_MODE_DISABLED	Loop filter is disabled.
V4L2_MPEG_VIDEO_H264_LOOP_FILTER_MODE_DISABLED_AT_SLICE_BOUNDARY	Loop filter is disabled at the slice boundary.

V4L2_CID_MPEG_VIDEO_H264_LOOP_FILTER_ALPHA (integer) Loop filter alpha coefficient, defined in the H264 standard. This value corresponds to the slice_alpha_c0_offset_div2 slice header field, and should be in the range of -6 to +6, inclusive. The actual alpha offset FilterOffsetA is twice this value. Applicable to the H264 encoder.

V4L2_CID_MPEG_VIDEO_H264_LOOP_FILTER_BETA (integer) Loop filter beta coefficient, defined in the H264 standard. This corresponds to the slice_beta_offset_div2 slice header field, and should be in the range of -6 to +6, inclusive. The actual beta offset FilterOffsetB is twice this value. Applicable to the H264 encoder.

V4L2_CID_MPEG_VIDEO_H264_ENTROPY_MODE (enum)

enum v4l2_mpeg_video_h264_entropy_mode - Entropy coding mode for H264 - CABAC/CAVLC. Applicable to the H264 encoder. Possible values are:

V4L2_MPEG_VIDEO_H264_ENTROPY_MODE_CAVLC	Use CAVLC entropy coding.
V4L2_MPEG_VIDEO_H264_ENTROPY_MODE_CABAC	Use CABAC entropy coding.

V4L2_CID_MPEG_VIDEO_H264_8X8_TRANSFORM (boolean) Enable 8X8 transform for H264. Applicable to the H264 encoder.

V4L2_CID_MPEG_VIDEO_H264_CONSTRAINED_INTRA_PREDICTION (boolean) Enable constrained intra prediction for H264. Applicable to the H264 encoder.

V4L2_CID_MPEG_VIDEO_H264_CHROMA_QP_INDEX_OFFSET (integer) Specify the offset that should be added to the luma quantization parameter to determine the chroma quantization parameter. Applicable to the H264 encoder.

V4L2_CID_MPEG_VIDEO_CYCLIC_INTRA_REFRESH_MB (integer) Cyclic intra macroblock refresh. This is the number of continuous macroblocks refreshed every frame. Each frame a successive set of macroblocks is refreshed until the cycle completes and

starts from the top of the frame. Setting this control to zero means that macroblocks will not be refreshed. Note that this control will not take effect when `V4L2_CID_MPEG_VIDEO_INTRA_REFRESH_PERIOD` control is set to non zero value. Applicable to H264, H263 and MPEG4 encoder.

V4L2_CID_MPEG_VIDEO_INTRA_REFRESH_PERIOD (integer) Intra macroblock refresh period. This sets the period to refresh the whole frame. In other words, this defines the number of frames for which the whole frame will be intra-refreshed. An example: setting period to 1 means that the whole frame will be refreshed, setting period to 2 means that the half of macroblocks will be intra-refreshed on frameX and the other half of macroblocks will be refreshed in frameX + 1 and so on. Setting the period to zero means no period is specified. Note that if the client sets this control to non zero value the `V4L2_CID_MPEG_VIDEO_CYCLIC_INTRA_REFRESH_MB` control shall be ignored. Applicable to H264 and HEVC encoders.

V4L2_CID_MPEG_VIDEO_FRAME_RC_ENABLE (boolean) Frame level rate control enable. If this control is disabled then the quantization parameter for each frame type is constant and set with appropriate controls (e.g. `V4L2_CID_MPEG_VIDEO_H263_I_FRAME_QP`). If frame rate control is enabled then quantization parameter is adjusted to meet the chosen bitrate. Minimum and maximum value for the quantization parameter can be set with appropriate controls (e.g. `V4L2_CID_MPEG_VIDEO_H263_MIN_QP`). Applicable to encoders.

V4L2_CID_MPEG_VIDEO_MB_RC_ENABLE (boolean) Macroblock level rate control enable. Applicable to the MPEG4 and H264 encoders.

V4L2_CID_MPEG_VIDEO_MPEG4_QPEL (boolean) Quarter pixel motion estimation for MPEG4. Applicable to the MPEG4 encoder.

V4L2_CID_MPEG_VIDEO_H263_I_FRAME_QP (integer) Quantization parameter for an I frame for H263. Valid range: from 1 to 31.

V4L2_CID_MPEG_VIDEO_H263_MIN_QP (integer) Minimum quantization parameter for H263. Valid range: from 1 to 31.

V4L2_CID_MPEG_VIDEO_H263_MAX_QP (integer) Maximum quantization parameter for H263. Valid range: from 1 to 31.

V4L2_CID_MPEG_VIDEO_H263_P_FRAME_QP (integer) Quantization parameter for an P frame for H263. Valid range: from 1 to 31.

V4L2_CID_MPEG_VIDEO_H263_B_FRAME_QP (integer) Quantization parameter for an B frame for H263. Valid range: from 1 to 31.

V4L2_CID_MPEG_VIDEO_H264_I_FRAME_QP (integer) Quantization parameter for an I frame for H264. Valid range: from 0 to 51.

V4L2_CID_MPEG_VIDEO_H264_MIN_QP (integer) Minimum quantization parameter for H264. Valid range: from 0 to 51.

V4L2_CID_MPEG_VIDEO_H264_MAX_QP (integer) Maximum quantization parameter for H264. Valid range: from 0 to 51.

V4L2_CID_MPEG_VIDEO_H264_P_FRAME_QP (integer) Quantization parameter for an P frame for H264. Valid range: from 0 to 51.

V4L2_CID_MPEG_VIDEO_H264_B_FRAME_QP (integer) Quantization parameter for an B frame for H264. Valid range: from 0 to 51.

V4L2_CID_MPEG_VIDEO_H264_I_FRAME_MIN_QP (integer) Minimum quantization parameter for the H264 I frame to limit I frame quality to a range. Valid range: from 0 to 51. If `V4L2_CID_MPEG_VIDEO_H264_MIN_QP` is also set, the quantization parameter should be chosen to meet both requirements.

V4L2_CID_MPEG_VIDEO_H264_I_FRAME_MAX_QP (integer) Maximum quantization parameter for the H264 I frame to limit I frame quality to a range. Valid range: from 0 to 51. If `V4L2_CID_MPEG_VIDEO_H264_MAX_QP` is also set, the quantization parameter should be chosen to meet both requirements.

V4L2_CID_MPEG_VIDEO_H264_P_FRAME_MIN_QP (integer) Minimum quantization parameter for the H264 P frame to limit P frame quality to a range. Valid range: from 0 to 51. If `V4L2_CID_MPEG_VIDEO_H264_MIN_QP` is also set, the quantization parameter should be chosen to meet both requirements.

V4L2_CID_MPEG_VIDEO_H264_P_FRAME_MAX_QP (integer) Maximum quantization parameter for the H264 P frame to limit P frame quality to a range. Valid range: from 0 to 51. If `V4L2_CID_MPEG_VIDEO_H264_MAX_QP` is also set, the quantization parameter should be chosen to meet both requirements.

V4L2_CID_MPEG_VIDEO_H264_B_FRAME_MIN_QP (integer) Minimum quantization parameter for the H264 B frame to limit B frame quality to a range. Valid range: from 0 to 51. If `V4L2_CID_MPEG_VIDEO_H264_MIN_QP` is also set, the quantization parameter should be chosen to meet both requirements.

V4L2_CID_MPEG_VIDEO_H264_B_FRAME_MAX_QP (integer) Maximum quantization parameter for the H264 B frame to limit B frame quality to a range. Valid range: from 0 to 51. If `V4L2_CID_MPEG_VIDEO_H264_MAX_QP` is also set, the quantization parameter should be chosen to meet both requirements.

V4L2_CID_MPEG_VIDEO_MPEG4_I_FRAME_QP (integer) Quantization parameter for an I frame for MPEG4. Valid range: from 1 to 31.

V4L2_CID_MPEG_VIDEO_MPEG4_MIN_QP (integer) Minimum quantization parameter for MPEG4. Valid range: from 1 to 31.

V4L2_CID_MPEG_VIDEO_MPEG4_MAX_QP (integer) Maximum quantization parameter for MPEG4. Valid range: from 1 to 31.

V4L2_CID_MPEG_VIDEO_MPEG4_P_FRAME_QP (integer) Quantization parameter for an P frame for MPEG4. Valid range: from 1 to 31.

V4L2_CID_MPEG_VIDEO_MPEG4_B_FRAME_QP (integer) Quantization parameter for an B frame for MPEG4. Valid range: from 1 to 31.

V4L2_CID_MPEG_VIDEO_VBV_SIZE (integer) The Video Buffer Verifier size in kilobytes, it is used as a limitation of frame skip. The VBV is defined in the standard as a mean to verify that the produced stream will be successfully decoded. The standard describes it as “Part of a hypothetical decoder that is conceptually connected to the output of the encoder. Its purpose is to provide a constraint on the variability of the data rate that an encoder or editing process may produce.”. Applicable to the MPEG1, MPEG2, MPEG4 encoders.

V4L2_CID_MPEG_VIDEO_VBV_DELAY (integer) Sets the initial delay in milliseconds for VBV buffer control.

V4L2_CID_MPEG_VIDEO_MV_H_SEARCH_RANGE (integer) Horizontal search range defines maximum horizontal search area in pixels to search and match for the present Macroblock

(MB) in the reference picture. This V4L2 control macro is used to set horizontal search range for motion estimation module in video encoder.

V4L2_CID_MPEG_VIDEO_MV_V_SEARCH_RANGE (integer) Vertical search range defines maximum vertical search area in pixels to search and match for the present Macroblock (MB) in the reference picture. This V4L2 control macro is used to set vertical search range for motion estimation module in video encoder.

V4L2_CID_MPEG_VIDEO_FORCE_KEY_FRAME (button) Force a key frame for the next queued buffer. Applicable to encoders. This is a general, codec-agnostic keyframe control.

V4L2_CID_MPEG_VIDEO_H264_CPB_SIZE (integer) The Coded Picture Buffer size in kilobytes, it is used as a limitation of frame skip. The CPB is defined in the H264 standard as a mean to verify that the produced stream will be successfully decoded. Applicable to the H264 encoder.

V4L2_CID_MPEG_VIDEO_H264_I_PERIOD (integer) Period between I-frames in the open GOP for H264. In case of an open GOP this is the period between two I-frames. The period between IDR (Instantaneous Decoding Refresh) frames is taken from the GOP_SIZE control. An IDR frame, which stands for Instantaneous Decoding Refresh is an I-frame after which no prior frames are referenced. This means that a stream can be restarted from an IDR frame without the need to store or decode any previous frames. Applicable to the H264 encoder.

V4L2_CID_MPEG_VIDEO_HEADER_MODE (enum)

enum v4l2_mpeg_video_header_mode - Determines whether the header is returned as the first buffer or is it returned together with the first frame. Applicable to encoders. Possible values are:

V4L2_MPEG_VIDEO_HEADER_MODE_SEPARATE	The stream header is returned separately in the first buffer.
V4L2_MPEG_VIDEO_HEADER_MODE_JOINED_WITH_1ST_FRAME	The stream header is returned together with the first encoded frame.

V4L2_CID_MPEG_VIDEO_REPEAT_SEQ_HEADER (boolean) Repeat the video sequence headers. Repeating these headers makes random access to the video stream easier. Applicable to the MPEG1, 2 and 4 encoder.

V4L2_CID_MPEG_VIDEO_DECODER_MPEG4_DEBLOCK_FILTER (boolean) Enabled the deblocking post processing filter for MPEG4 decoder. Applicable to the MPEG4 decoder.

V4L2_CID_MPEG_VIDEO_MPEG4_VOP_TIME_RES (integer) vop_time_increment_resolution value for MPEG4. Applicable to the MPEG4 encoder.

V4L2_CID_MPEG_VIDEO_MPEG4_VOP_TIME_INC (integer) vop_time_increment value for MPEG4. Applicable to the MPEG4 encoder.

V4L2_CID_MPEG_VIDEO_H264_SEI_FRAME_PACKING (boolean) Enable generation of frame packing supplemental enhancement information in the encoded bitstream. The frame packing SEI message contains the arrangement of L and R planes for 3D viewing. Applicable to the H264 encoder.

V4L2_CID_MPEG_VIDEO_H264_SEI_FP_CURRENT_FRAME_0 (boolean) Sets current frame as frame0 in frame packing SEI. Applicable to the H264 encoder.

V4L2_CID_MPEG_VIDEO_H264_SEI_FP_ARRANGEMENT_TYPE (enum)

enum v4l2_mpeg_video_h264_sei_fp_arrangement_type - Frame packing arrangement type for H264 SEI. Applicable to the H264 encoder. Possible values are:

V4L2_MPEG_VIDEO_H264_SEI_FP_ARRANGEMENT_TYPE_CHEKERBOARD	Pixels are alternatively from L and R.
V4L2_MPEG_VIDEO_H264_SEI_FP_ARRANGEMENT_TYPE_COLUMN	L and R are interlaced by column.
V4L2_MPEG_VIDEO_H264_SEI_FP_ARRANGEMENT_TYPE_ROW	L and R are interlaced by row.
V4L2_MPEG_VIDEO_H264_SEI_FP_ARRANGEMENT_TYPE_SIDE_BY_SIDE	L is on the left, R on the right.
V4L2_MPEG_VIDEO_H264_SEI_FP_ARRANGEMENT_TYPE_TOP_BOTTOM	L is on top, R on bottom.
V4L2_MPEG_VIDEO_H264_SEI_FP_ARRANGEMENT_TYPE_TEMPORAL	One view per frame.

V4L2_CID_MPEG_VIDEO_H264_FMO (boolean) Enables flexible macroblock ordering in the encoded bitstream. It is a technique used for restructuring the ordering of macroblocks in pictures. Applicable to the H264 encoder.

V4L2_CID_MPEG_VIDEO_H264_FMO_MAP_TYPE (enum)

enum v4l2_mpeg_video_h264_fmo_map_type - When using FMO, the map type divides the image in different scan patterns of macroblocks. Applicable to the H264 encoder. Possible values are:

V4L2_MPEG_VIDEO_H264_FMO_MAP_TYPE_INTERLEAVED_SLICES	Slices are interleaved one after other with macroblocks in run length order.
V4L2_MPEG_VIDEO_H264_FMO_MAP_TYPE_SCATTERED_SLICES	Scatters the macroblocks based on a mathematical function known to both encoder and decoder.
V4L2_MPEG_VIDEO_H264_FMO_MAP_TYPE_FOREGROUND_WITH_LEFT_OVER	Macroblocks arranged in rectangular areas or regions of interest.
V4L2_MPEG_VIDEO_H264_FMO_MAP_TYPE_BOX_OUT	Slice groups grow in a cyclic way from centre to outwards.
V4L2_MPEG_VIDEO_H264_FMO_MAP_TYPE_RASTER_SCAN	Slice groups grow in raster scan pattern from left to right.
V4L2_MPEG_VIDEO_H264_FMO_MAP_TYPE_WIPE_SCAN	Slice groups grow in wipe scan pattern from top to bottom.
V4L2_MPEG_VIDEO_H264_FMO_MAP_TYPE_EXPLICIT	User defined map type.

V4L2_CID_MPEG_VIDEO_H264_FMO_SLICE_GROUP (integer) Number of slice groups in FMO. Applicable to the H264 encoder.

V4L2_CID_MPEG_VIDEO_H264_FMO_CHANGE_DIRECTION (enum)

enum v4l2_mpeg_video_h264_fmo_change_dir - Specifies a direction of the slice group change for raster and wipe maps. Applicable to the H264 encoder. Possible values are:

V4L2_MPEG_VIDEO_H264_FMO_CHANGE_DIR_RIGHT	Raster scan or wipe right.
V4L2_MPEG_VIDEO_H264_FMO_CHANGE_DIR_LEFT	Reverse raster scan or wipe left.

V4L2_CID_MPEG_VIDEO_H264_FMO_CHANGE_RATE (integer) Specifies the size of the first slice

group for raster and wipe map. Applicable to the H264 encoder.

V4L2_CID_MPEG_VIDEO_H264_FMO_RUN_LENGTH (integer) Specifies the number of consecutive macroblocks for the interleaved map. Applicable to the H264 encoder.

V4L2_CID_MPEG_VIDEO_H264_ASO (boolean) Enables arbitrary slice ordering in encoded bit-stream. Applicable to the H264 encoder.

V4L2_CID_MPEG_VIDEO_H264_ASO_SLICE_ORDER (integer) Specifies the slice order in ASO. Applicable to the H264 encoder. The supplied 32-bit integer is interpreted as follows (bit 0 = least significant bit):

Bit 0:15	Slice ID
Bit 16:32	Slice position or order

V4L2_CID_MPEG_VIDEO_H264_HIERARCHICAL_CODING (boolean) Enables H264 hierarchical coding. Applicable to the H264 encoder.

V4L2_CID_MPEG_VIDEO_H264_HIERARCHICAL_CODING_TYPE (enum)

enum v4l2_mpeg_video_h264_hierarchical_coding_type - Specifies the hierarchical coding type. Applicable to the H264 encoder. Possible values are:

V4L2_MPEG_VIDEO_H264_HIERARCHICAL_CODING_B	Hierarchical B coding.
V4L2_MPEG_VIDEO_H264_HIERARCHICAL_CODING_P	Hierarchical P coding.

V4L2_CID_MPEG_VIDEO_H264_HIERARCHICAL_CODING_LAYER (integer) Specifies the number of hierarchical coding layers. Applicable to the H264 encoder.

V4L2_CID_MPEG_VIDEO_H264_HIERARCHICAL_CODING_LAYER_QP (integer) Specifies a user defined QP for each layer. Applicable to the H264 encoder. The supplied 32-bit integer is interpreted as follows (bit 0 = least significant bit):

Bit 0:15	QP value
Bit 16:32	Layer number

V4L2_CID_MPEG_VIDEO_H264_HIER_CODING_L0_BR (integer) Indicates bit rate (bps) for hierarchical coding layer 0 for H264 encoder.

V4L2_CID_MPEG_VIDEO_H264_HIER_CODING_L1_BR (integer) Indicates bit rate (bps) for hierarchical coding layer 1 for H264 encoder.

V4L2_CID_MPEG_VIDEO_H264_HIER_CODING_L2_BR (integer) Indicates bit rate (bps) for hierarchical coding layer 2 for H264 encoder.

V4L2_CID_MPEG_VIDEO_H264_HIER_CODING_L3_BR (integer) Indicates bit rate (bps) for hierarchical coding layer 3 for H264 encoder.

V4L2_CID_MPEG_VIDEO_H264_HIER_CODING_L4_BR (integer) Indicates bit rate (bps) for hierarchical coding layer 4 for H264 encoder.

V4L2_CID_MPEG_VIDEO_H264_HIER_CODING_L5_BR (integer) Indicates bit rate (bps) for hierarchical coding layer 5 for H264 encoder.

V4L2_CID_MPEG_VIDEO_H264_HIER_CODING_L6_BR (integer) Indicates bit rate (bps) for hierarchical coding layer 6 for H264 encoder.

V4L2_CID_FWHT_I_FRAME_QP (integer) Quantization parameter for an I frame for FWHT. Valid range: from 1 to 31.

V4L2_CID_FWHT_P_FRAME_QP (integer) Quantization parameter for a P frame for FWHT. Valid range: from 1 to 31.

MFC 5.1 MPEG Controls

The following MPEG class controls deal with MPEG decoding and encoding settings that are specific to the Multi Format Codec 5.1 device present in the S5P family of SoCs by Samsung.

MFC 5.1 Control IDs

V4L2_CID_MPEG_MFC51_VIDEO_DECODER_H264_DISPLAY_DELAY_ENABLE (boolean) If the display delay is enabled then the decoder is forced to return a CAPTURE buffer (decoded frame) after processing a certain number of OUTPUT buffers. The delay can be set through **V4L2_CID_MPEG_MFC51_VIDEO_DECODER_H264_DISPLAY_DELAY**. This feature can be used for example for generating thumbnails of videos. Applicable to the H264 decoder.

Note: This control is deprecated. Use the standard **V4L2_CID_MPEG_VIDEO_DEC_DISPLAY_DELAY_ENABLE** control instead.

V4L2_CID_MPEG_MFC51_VIDEO_DECODER_H264_DISPLAY_DELAY (integer) Display delay value for H264 decoder. The decoder is forced to return a decoded frame after the set 'display delay' number of frames. If this number is low it may result in frames returned out of display order, in addition the hardware may still be using the returned buffer as a reference picture for subsequent frames.

Note: This control is deprecated. Use the standard **V4L2_CID_MPEG_VIDEO_DEC_DISPLAY_DELAY** control instead.

V4L2_CID_MPEG_MFC51_VIDEO_H264_NUM_REF_PIC_FOR_P (integer) The number of reference pictures used for encoding a P picture. Applicable to the H264 encoder.

V4L2_CID_MPEG_MFC51_VIDEO_PADDING (boolean) Padding enable in the encoder - use a color instead of repeating border pixels. Applicable to encoders.

V4L2_CID_MPEG_MFC51_VIDEO_PADDING_YUV (integer) Padding color in the encoder. Applicable to encoders. The supplied 32-bit integer is interpreted as follows (bit 0 = least significant bit):

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

V4L2_CID_MPEG_MFC51_VIDEO_RC_REACTION_COEFF (integer) Reaction coefficient for MFC rate control. Applicable to encoders.

Note:

1. Valid only when the frame level RC is enabled.
2. For tight CBR, this field must be small (ex. 2 ~ 10). For VBR, this field must be large (ex. 100 ~ 1000).
3. It is not recommended to use the greater number than $\text{FRAME_RATE} * (10^9 / \text{BIT_RATE})$.

V4L2_CID_MPEG_MFC51_VIDEO_H264_ADAPTIVE_RC_DARK (boolean) Adaptive rate control for dark region. Valid only when H.264 and macroblock level RC is enabled (V4L2_CID_MPEG_VIDEO_MB_RC_ENABLE). Applicable to the H264 encoder.

V4L2_CID_MPEG_MFC51_VIDEO_H264_ADAPTIVE_RC_SMOOTH (boolean) Adaptive rate control for smooth region. Valid only when H.264 and macroblock level RC is enabled (V4L2_CID_MPEG_VIDEO_MB_RC_ENABLE). Applicable to the H264 encoder.

V4L2_CID_MPEG_MFC51_VIDEO_H264_ADAPTIVE_RC_STATIC (boolean) Adaptive rate control for static region. Valid only when H.264 and macroblock level RC is enabled (V4L2_CID_MPEG_VIDEO_MB_RC_ENABLE). Applicable to the H264 encoder.

V4L2_CID_MPEG_MFC51_VIDEO_H264_ADAPTIVE_RC_ACTIVITY (boolean) Adaptive rate control for activity region. Valid only when H.264 and macroblock level RC is enabled (V4L2_CID_MPEG_VIDEO_MB_RC_ENABLE). Applicable to the H264 encoder.

V4L2_CID_MPEG_MFC51_VIDEO_FRAME_SKIP_MODE (enum)

Note: This control is deprecated. Use the standard V4L2_CID_MPEG_VIDEO_FRAME_SKIP_MODE control instead.

enum v4l2_mpeg_mfc51_video_frame_skip_mode - Indicates in what conditions the encoder should skip frames. If encoding a frame would cause the encoded stream to be larger then a chosen data limit then the frame will be skipped. Possible values are:

V4L2_MPEG_MFC51_VIDEO_FRAME_SKIP_MODE_DISABLED	Frame skip mode is disabled.
V4L2_MPEG_MFC51_VIDEO_FRAME_SKIP_MODE_LEVEL_LIMIT	Frame skip mode enabled and buffer limit is set by the chosen level and is defined by the standard.
V4L2_MPEG_MFC51_VIDEO_FRAME_SKIP_MODE_BUF_LIMIT	Frame skip mode enabled and buffer limit is set by the VBV (MPEG1/2/4) or CPB (H264) buffer size control.

V4L2_CID_MPEG_MFC51_VIDEO_RC_FIXED_TARGET_BIT (integer) Enable rate-control with fixed target bit. If this setting is enabled, then the rate control logic of the encoder will calculate the average bitrate for a GOP and keep it below or equal the set bitrate target. Otherwise the rate control logic calculates the overall average bitrate for the stream and keeps it below or equal to the set bitrate. In the first case the average bitrate for the whole stream will be smaller then the set bitrate. This is caused because the average is calculated for smaller number of frames, on the other hand enabling this setting will ensure that the stream will meet tight bandwidth constraints. Applicable to encoders.

V4L2_CID_MPEG_MFC51_VIDEO_FORCE_FRAME_TYPE (enum)

enum v4l2_mpeg_mfc51_video_force_frame_type - Force a frame type for the next queued buffer. Applicable to encoders. Possible values are:

V4L2_MPEG_MFC51_FORCE_FRAME_TYPE_DISABLED	Forcing a specific frame type disabled.
V4L2_MPEG_MFC51_FORCE_FRAME_TYPE_I_FRAME	Force an I-frame.
V4L2_MPEG_MFC51_FORCE_FRAME_TYPE_NOT_CODED	Force a non-coded frame.

CX2341x MPEG Controls

The following MPEG class controls deal with MPEG encoding settings that are specific to the Conexant CX23415 and CX23416 MPEG encoding chips.

CX2341x Control IDs

V4L2_CID_MPEG_CX2341X_VIDEO_SPATIAL_FILTER_MODE (enum)

enum v4l2_mpeg_cx2341x_video_spatial_filter_mode - Sets the Spatial Filter mode (default MANUAL). Possible values are:

V4L2_MPEG_CX2341X_VIDEO_SPATIAL_FILTER_MODE_MANUAL	Choose the filter manually
V4L2_MPEG_CX2341X_VIDEO_SPATIAL_FILTER_MODE_AUTO	Choose the filter automatically

V4L2_CID_MPEG_CX2341X_VIDEO_SPATIAL_FILTER (integer (0-15)) The setting for the Spatial Filter. 0 = off, 15 = maximum. (Default is 0.)

V4L2_CID_MPEG_CX2341X_VIDEO_LUMA_SPATIAL_FILTER_TYPE (enum)

enum v4l2_mpeg_cx2341x_video_luma_spatial_filter_type - Select the algorithm to use for the Luma Spatial Filter (default 1D_HOR). Possible values:

V4L2_MPEG_CX2341X_VIDEO_LUMA_SPATIAL_FILTER_TYPE_OFF	No filter
V4L2_MPEG_CX2341X_VIDEO_LUMA_SPATIAL_FILTER_TYPE_1D_HOR	One-dimensional horizontal
V4L2_MPEG_CX2341X_VIDEO_LUMA_SPATIAL_FILTER_TYPE_1D_VERT	One-dimensional vertical
V4L2_MPEG_CX2341X_VIDEO_LUMA_SPATIAL_FILTER_TYPE_2D_HV_SEPARABLE	Two-dimensional separable
V4L2_MPEG_CX2341X_VIDEO_LUMA_SPATIAL_FILTER_TYPE_2D_SYM_NON_SEPARABLE	Two-dimensional symmetrical non-separable

V4L2_CID_MPEG_CX2341X_VIDEO_CHROMA_SPATIAL_FILTER_TYPE (enum)

enum v4l2_mpeg_cx2341x_video_chroma_spatial_filter_type - Select the algorithm for the Chroma Spatial Filter (default 1D_HOR). Possible values are:

V4L2_MPEG_CX2341X_VIDEO_CHROMA_SPATIAL_FILTER_TYPE_OFF	No filter
V4L2_MPEG_CX2341X_VIDEO_CHROMA_SPATIAL_FILTER_TYPE_1D_HOR	One-dimensional horizontal

V4L2_CID_MPEG_CX2341X_VIDEO_TEMPORAL_FILTER_MODE (enum)

enum v4l2_mpeg_cx2341x_video_temporal_filter_mode - Sets the Temporal Filter mode (default MANUAL). Possible values are:

V4L2_MPEG_CX2341X_VIDEO_TEMPORAL_FILTER_MODE_MANUAL	Choose the filter manually
V4L2_MPEG_CX2341X_VIDEO_TEMPORAL_FILTER_MODE_AUTO	Choose the filter automatically

V4L2_CID_MPEG_CX2341X_VIDEO_TEMPORAL_FILTER (integer (0-31)) The setting for the Temporal Filter. 0 = off, 31 = maximum. (Default is 8 for full-scale capturing and 0 for scaled capturing.)

V4L2_CID_MPEG_CX2341X_VIDEO_MEDIAN_FILTER_TYPE (enum)

enum v4l2_mpeg_cx2341x_video_median_filter_type - Median Filter Type (default OFF). Possible values are:

V4L2_MPEG_CX2341X_VIDEO_MEDIAN_FILTER_TYPE_OFF	No filter
V4L2_MPEG_CX2341X_VIDEO_MEDIAN_FILTER_TYPE_HOR	Horizontal filter
V4L2_MPEG_CX2341X_VIDEO_MEDIAN_FILTER_TYPE_VERT	Vertical filter
V4L2_MPEG_CX2341X_VIDEO_MEDIAN_FILTER_TYPE_HOR_VERT	Horizontal and vertical filter
V4L2_MPEG_CX2341X_VIDEO_MEDIAN_FILTER_TYPE_DIAG	Diagonal filter

V4L2_CID_MPEG_CX2341X_VIDEO_LUMA_MEDIAN_FILTER_BOTTOM (integer (0-255)) Threshold above which the luminance median filter is enabled (default 0)

V4L2_CID_MPEG_CX2341X_VIDEO_LUMA_MEDIAN_FILTER_TOP (integer (0-255)) Threshold below which the luminance median filter is enabled (default 255)

V4L2_CID_MPEG_CX2341X_VIDEO_CHROMA_MEDIAN_FILTER_BOTTOM (integer (0-255)) Threshold above which the chroma median filter is enabled (default 0)

V4L2_CID_MPEG_CX2341X_VIDEO_CHROMA_MEDIAN_FILTER_TOP (integer (0-255)) Threshold below which the chroma median filter is enabled (default 255)

V4L2_CID_MPEG_CX2341X_STREAM_INSERT_NAV_PACKETS (boolean) The CX2341X MPEG encoder can insert one empty MPEG-2 PES packet into the stream between every four video frames. The packet size is 2048 bytes, including the packet_start_code_prefix and stream_id fields. The stream_id is 0xBF (private stream 2). The payload consists of 0x00 bytes, to be filled in by the application. 0 = do not insert, 1 = insert packets.

VPX Control Reference

The VPX controls include controls for encoding parameters of VPx video codec.

VPX Control IDs

V4L2_CID_MPEG_VIDEO_VPX_NUM_PARTITIONS (enum)

enum v4l2_vp8_num_partitions - The number of token partitions to use in VP8 encoder. Possible values are:

V4L2_CID_MPEG_VIDEO_VPX_1_PARTITION	1 coefficient partition
V4L2_CID_MPEG_VIDEO_VPX_2_PARTITIONS	2 coefficient partitions
V4L2_CID_MPEG_VIDEO_VPX_4_PARTITIONS	4 coefficient partitions
V4L2_CID_MPEG_VIDEO_VPX_8_PARTITIONS	8 coefficient partitions

V4L2_CID_MPEG_VIDEO_VPX_IMD_DISABLE_4X4 (boolean) Setting this prevents intra 4x4 mode in the intra mode decision.

V4L2_CID_MPEG_VIDEO_VPX_NUM_REF_FRAMES (enum)

enum v4l2_vp8_num_ref_frames - The number of reference pictures for encoding P frames. Possible values are:

V4L2_CID_MPEG_VIDEO_VPX_1_REF_FRAME	Last encoded frame will be searched
V4L2_CID_MPEG_VIDEO_VPX_2_REF_FRAME	Two frames will be searched among the last encoded frame, the golden frame and the alternate reference (altref) frame. The encoder implementation will decide which two are chosen.
V4L2_CID_MPEG_VIDEO_VPX_3_REF_FRAME	The last encoded frame, the golden frame and the altref frame will be searched.

V4L2_CID_MPEG_VIDEO_VPX_FILTER_LEVEL (integer) Indicates the loop filter level. The adjustment of the loop filter level is done via a delta value against a baseline loop filter value.

V4L2_CID_MPEG_VIDEO_VPX_FILTER_SHARPNESS (integer) This parameter affects the loop filter. Anything above zero weakens the deblocking effect on the loop filter.

V4L2_CID_MPEG_VIDEO_VPX_GOLDEN_FRAME_REF_PERIOD (integer) Sets the refresh period for the golden frame. The period is defined in number of frames. For a value of 'n', every nth frame starting from the first key frame will be taken as a golden frame. For eg. for encoding sequence of 0, 1, 2, 3, 4, 5, 6, 7 where the golden frame refresh period is set as 4, the frames 0, 4, 8 etc will be taken as the golden frames as frame 0 is always a key frame.

V4L2_CID_MPEG_VIDEO_VPX_GOLDEN_FRAME_SEL (enum)

enum v4l2_vp8_golden_frame_sel - Selects the golden frame for encoding. Possible values are:

V4L2_CID_MPEG_VIDEO_VPX_GOLDEN_FRAME_USE_PREV	Use the (n-2)th frame as a golden frame, current frame index being 'n'.
V4L2_CID_MPEG_VIDEO_VPX_GOLDEN_FRAME_USE_REF_PERIOD	Use the previous specific frame indicated by V4L2_CID_MPEG_VIDEO_VPX_GOLDEN_FRAME_REF_PERIOD as a golden frame.

V4L2_CID_MPEG_VIDEO_VPX_MIN_QP (integer) Minimum quantization parameter for VP8.

V4L2_CID_MPEG_VIDEO_VPX_MAX_QP (integer) Maximum quantization parameter for VP8.

V4L2_CID_MPEG_VIDEO_VPX_I_FRAME_QP (integer) Quantization parameter for an I frame for VP8.

V4L2_CID_MPEG_VIDEO_VPX_P_FRAME_QP (integer) Quantization parameter for a P frame for VP8.

V4L2_CID_MPEG_VIDEO_VP8_PROFILE (enum)

enum v4l2_mpeg_video_vp8_profile - This control allows selecting the profile for VP8 encoder. This is also used to enumerate supported profiles by VP8 encoder or decoder. Possible values are:

V4L2_MPEG_VIDEO_VP8_PROFILE_0	Profile 0
V4L2_MPEG_VIDEO_VP8_PROFILE_1	Profile 1
V4L2_MPEG_VIDEO_VP8_PROFILE_2	Profile 2
V4L2_MPEG_VIDEO_VP8_PROFILE_3	Profile 3

V4L2_CID_MPEG_VIDEO_VP9_PROFILE (enum)

enum v4l2_mpeg_video_vp9_profile - This control allows selecting the profile for VP9 encoder. This is also used to enumerate supported profiles by VP9 encoder or decoder. Possible values are:

V4L2_MPEG_VIDEO_VP9_PROFILE_0	Profile 0
V4L2_MPEG_VIDEO_VP9_PROFILE_1	Profile 1
V4L2_MPEG_VIDEO_VP9_PROFILE_2	Profile 2
V4L2_MPEG_VIDEO_VP9_PROFILE_3	Profile 3

V4L2_CID_MPEG_VIDEO_VP9_LEVEL (enum)

enum v4l2_mpeg_video_vp9_level - This control allows selecting the level for VP9 encoder. This is also used to enumerate supported levels by VP9 encoder or decoder. More information can be found at [webmproject](#). Possible values are:

V4L2_MPEG_VIDEO_VP9_LEVEL_1_0	Level 1
V4L2_MPEG_VIDEO_VP9_LEVEL_1_1	Level 1.1
V4L2_MPEG_VIDEO_VP9_LEVEL_2_0	Level 2
V4L2_MPEG_VIDEO_VP9_LEVEL_2_1	Level 2.1
V4L2_MPEG_VIDEO_VP9_LEVEL_3_0	Level 3
V4L2_MPEG_VIDEO_VP9_LEVEL_3_1	Level 3.1
V4L2_MPEG_VIDEO_VP9_LEVEL_4_0	Level 4
V4L2_MPEG_VIDEO_VP9_LEVEL_4_1	Level 4.1
V4L2_MPEG_VIDEO_VP9_LEVEL_5_0	Level 5
V4L2_MPEG_VIDEO_VP9_LEVEL_5_1	Level 5.1
V4L2_MPEG_VIDEO_VP9_LEVEL_5_2	Level 5.2
V4L2_MPEG_VIDEO_VP9_LEVEL_6_0	Level 6
V4L2_MPEG_VIDEO_VP9_LEVEL_6_1	Level 6.1
V4L2_MPEG_VIDEO_VP9_LEVEL_6_2	Level 6.2

High Efficiency Video Coding (HEVC/H.265) Control Reference

The HEVC/H.265 controls include controls for encoding parameters of HEVC/H.265 video codec.

HEVC/H.265 Control IDs

V4L2_CID_MPEG_VIDEO_HEVC_MIN_QP (integer) Minimum quantization parameter for HEVC. Valid range: from 0 to 51 for 8 bit and from 0 to 63 for 10 bit.

V4L2_CID_MPEG_VIDEO_HEVC_MAX_QP (integer) Maximum quantization parameter for HEVC. Valid range: from 0 to 51 for 8 bit and from 0 to 63 for 10 bit.

V4L2_CID_MPEG_VIDEO_HEVC_I_FRAME_QP (integer) Quantization parameter for an I frame for HEVC. Valid range: [V4L2_CID_MPEG_VIDEO_HEVC_MIN_QP, V4L2_CID_MPEG_VIDEO_HEVC_MAX_QP].

V4L2_CID_MPEG_VIDEO_HEVC_P_FRAME_QP (integer) Quantization parameter for a P frame for HEVC. Valid range: [V4L2_CID_MPEG_VIDEO_HEVC_MIN_QP, V4L2_CID_MPEG_VIDEO_HEVC_MAX_QP].

V4L2_CID_MPEG_VIDEO_HEVC_B_FRAME_QP (integer) Quantization parameter for a B frame for HEVC. Valid range: [V4L2_CID_MPEG_VIDEO_HEVC_MIN_QP, V4L2_CID_MPEG_VIDEO_HEVC_MAX_QP].

V4L2_CID_MPEG_VIDEO_HEVC_I_FRAME_MIN_QP (integer) Minimum quantization parameter for the HEVC I frame to limit I frame quality to a range. Valid range: from 0 to 51 for 8 bit and from 0 to 63 for 10 bit. If V4L2_CID_MPEG_VIDEO_HEVC_MIN_QP is also set, the quantization parameter should be chosen to meet both requirements.

V4L2_CID_MPEG_VIDEO_HEVC_I_FRAME_MAX_QP (integer) Maximum quantization parameter for the HEVC I frame to limit I frame quality to a range. Valid range: from 0 to 51 for 8 bit and from 0 to 63 for 10 bit. If V4L2_CID_MPEG_VIDEO_HEVC_MAX_QP is also set, the quantization parameter should be chosen to meet both requirements.

V4L2_CID_MPEG_VIDEO_HEVC_P_FRAME_MIN_QP (integer) Minimum quantization parameter for the HEVC P frame to limit P frame quality to a range. Valid range: from 0 to 51 for 8 bit and from 0 to 63 for 10 bit. If V4L2_CID_MPEG_VIDEO_HEVC_MIN_QP is also set, the quantization parameter should be chosen to meet both requirements.

V4L2_CID_MPEG_VIDEO_HEVC_P_FRAME_MAX_QP (integer) Maximum quantization parameter for the HEVC P frame to limit P frame quality to a range. Valid range: from 0 to 51 for 8 bit and from 0 to 63 for 10 bit. If V4L2_CID_MPEG_VIDEO_HEVC_MAX_QP is also set, the quantization parameter should be chosen to meet both requirements.

V4L2_CID_MPEG_VIDEO_HEVC_B_FRAME_MIN_QP (integer) Minimum quantization parameter for the HEVC B frame to limit B frame quality to a range. Valid range: from 0 to 51 for 8 bit and from 0 to 63 for 10 bit. If V4L2_CID_MPEG_VIDEO_HEVC_MIN_QP is also set, the quantization parameter should be chosen to meet both requirements.

V4L2_CID_MPEG_VIDEO_HEVC_B_FRAME_MAX_QP (integer) Maximum quantization parameter for the HEVC B frame to limit B frame quality to a range. Valid range: from 0 to 51 for 8 bit and from 0 to 63 for 10 bit. If V4L2_CID_MPEG_VIDEO_HEVC_MAX_QP is also set, the quantization parameter should be chosen to meet both requirements.

V4L2_CID_MPEG_VIDEO_HEVC_HIER_QP (boolean) HIERARCHICAL_QP allows the host to specify the quantization parameter values for each temporal layer through HIERARCHICAL_QP_LAYER. This is valid only if HIERARCHICAL_CODING_LAYER is greater than 1. Setting the control value to 1 enables setting of the QP values for the layers.

V4L2_CID_MPEG_VIDEO_HEVC_HIER_CODING_TYPE (enum)

enum v4l2_mpeg_video_hevc_hier_coding_type - Selects the hierarchical coding type for encoding. Possible values are:

V4L2_MPEG_VIDEO_HEVC_HIERARCHICAL_CODING_B	Use the B frame for hierarchical coding.
V4L2_MPEG_VIDEO_HEVC_HIERARCHICAL_CODING_P	Use the P frame for hierarchical coding.

V4L2_CID_MPEG_VIDEO_HEVC_HIER_CODING_LAYER (integer) Selects the hierarchical coding layer. In normal encoding (non-hierarchical coding), it should be zero. Possible values are [0, 6]. 0 indicates HIERARCHICAL CODING LAYER 0, 1 indicates HIERARCHICAL CODING LAYER 1 and so on.

V4L2_CID_MPEG_VIDEO_HEVC_HIER_CODING_L0_QP (integer) Indicates quantization parameter for hierarchical coding layer 0. Valid range: [V4L2_CID_MPEG_VIDEO_HEVC_MIN_QP, V4L2_CID_MPEG_VIDEO_HEVC_MAX_QP].

V4L2_CID_MPEG_VIDEO_HEVC_HIER_CODING_L1_QP (integer) Indicates quantization parameter for hierarchical coding layer 1. Valid range: [V4L2_CID_MPEG_VIDEO_HEVC_MIN_QP, V4L2_CID_MPEG_VIDEO_HEVC_MAX_QP].

V4L2_CID_MPEG_VIDEO_HEVC_HIER_CODING_L2_QP (integer) Indicates quantization parameter for hierarchical coding layer 2. Valid range: [V4L2_CID_MPEG_VIDEO_HEVC_MIN_QP, V4L2_CID_MPEG_VIDEO_HEVC_MAX_QP].

V4L2_CID_MPEG_VIDEO_HEVC_HIER_CODING_L3_QP (integer) Indicates quantization parameter for hierarchical coding layer 3. Valid range: [V4L2_CID_MPEG_VIDEO_HEVC_MIN_QP, V4L2_CID_MPEG_VIDEO_HEVC_MAX_QP].

V4L2_CID_MPEG_VIDEO_HEVC_HIER_CODING_L4_QP (integer) Indicates quantization parameter for hierarchical coding layer 4. Valid range: [V4L2_CID_MPEG_VIDEO_HEVC_MIN_QP, V4L2_CID_MPEG_VIDEO_HEVC_MAX_QP].

V4L2_CID_MPEG_VIDEO_HEVC_HIER_CODING_L5_QP (integer) Indicates quantization parameter for hierarchical coding layer 5. Valid range: [V4L2_CID_MPEG_VIDEO_HEVC_MIN_QP, V4L2_CID_MPEG_VIDEO_HEVC_MAX_QP].

V4L2_CID_MPEG_VIDEO_HEVC_HIER_CODING_L6_QP (integer) Indicates quantization parameter for hierarchical coding layer 6. Valid range: [V4L2_CID_MPEG_VIDEO_HEVC_MIN_QP, V4L2_CID_MPEG_VIDEO_HEVC_MAX_QP].

V4L2_CID_MPEG_VIDEO_HEVC_PROFILE (enum)

enum v4l2_mpeg_video_hevc_profile - Select the desired profile for HEVC encoder.

V4L2_MPEG_VIDEO_HEVC_PROFILE_MAIN	Main profile.
V4L2_MPEG_VIDEO_HEVC_PROFILE_MAIN_STILL_PICTURE	Main still picture profile.
V4L2_MPEG_VIDEO_HEVC_PROFILE_MAIN_10	Main 10 profile.

V4L2_CID_MPEG_VIDEO_HEVC_LEVEL (enum)

enum v4l2_mpeg_video_hevc_level - Selects the desired level for HEVC encoder.

V4L2_MPEG_VIDEO_HEVC_LEVEL_1	Level 1.0
V4L2_MPEG_VIDEO_HEVC_LEVEL_2	Level 2.0
V4L2_MPEG_VIDEO_HEVC_LEVEL_2_1	Level 2.1
V4L2_MPEG_VIDEO_HEVC_LEVEL_3	Level 3.0
V4L2_MPEG_VIDEO_HEVC_LEVEL_3_1	Level 3.1
V4L2_MPEG_VIDEO_HEVC_LEVEL_4	Level 4.0
V4L2_MPEG_VIDEO_HEVC_LEVEL_4_1	Level 4.1
V4L2_MPEG_VIDEO_HEVC_LEVEL_5	Level 5.0
V4L2_MPEG_VIDEO_HEVC_LEVEL_5_1	Level 5.1
V4L2_MPEG_VIDEO_HEVC_LEVEL_5_2	Level 5.2
V4L2_MPEG_VIDEO_HEVC_LEVEL_6	Level 6.0
V4L2_MPEG_VIDEO_HEVC_LEVEL_6_1	Level 6.1
V4L2_MPEG_VIDEO_HEVC_LEVEL_6_2	Level 6.2

V4L2_CID_MPEG_VIDEO_HEVC_FRAME_RATE_RESOLUTION (integer) Indicates the number of evenly spaced subintervals, called ticks, within one second. This is a 16 bit unsigned integer and has a maximum value up to 0xffff and a minimum value of 1.

V4L2_CID_MPEG_VIDEO_HEVC_TIER (enum)

enum v4l2_mpeg_video_hevc_tier - TIER_FLAG specifies tiers information of the HEVC encoded picture. Tier were made to deal with applications that differ in terms of maximum bit rate. Setting the flag to 0 selects HEVC tier as Main tier and setting this flag to 1 indicates High tier. High tier is for applications requiring high bit rates.

V4L2_MPEG_VIDEO_HEVC_TIER_MAIN	Main tier.
V4L2_MPEG_VIDEO_HEVC_TIER_HIGH	High tier.

V4L2_CID_MPEG_VIDEO_HEVC_MAX_PARTITION_DEPTH (integer) Selects HEVC maximum coding unit depth.

V4L2_CID_MPEG_VIDEO_HEVC_LOOP_FILTER_MODE (enum)

enum v4l2_mpeg_video_hevc_loop_filter_mode - Loop filter mode for HEVC encoder. Possible values are:

V4L2_MPEG_VIDEO_HEVC_LOOP_FILTER_MODE_DISABLED	Loop filter is disabled.
V4L2_MPEG_VIDEO_HEVC_LOOP_FILTER_MODE_ENABLED	Loop filter is enabled.
V4L2_MPEG_VIDEO_HEVC_LOOP_FILTER_MODE_DISABLED_AT_SLICE_BOUNDARY	Loop filter is disabled at the slice boundary.

V4L2_CID_MPEG_VIDEO_HEVC_LF_BETA_OFFSET_DIV2 (integer) Selects HEVC loop filter beta offset. The valid range is [-6, +6].

V4L2_CID_MPEG_VIDEO_HEVC_LF_TC_OFFSET_DIV2 (integer) Selects HEVC loop filter tc offset. The valid range is [-6, +6].

V4L2_CID_MPEG_VIDEO_HEVC_REFRESH_TYPE (enum)

enum v4l2_mpeg_video_hevc_hier_refresh_type - Selects refresh type for HEVC encoder. Host has to specify the period into V4L2_CID_MPEG_VIDEO_HEVC_REFRESH_PERIOD.

V4L2_MPEG_VIDEO_HEVC_REFRESH_NONE	Use the B frame for hierarchical coding.
V4L2_MPEG_VIDEO_HEVC_REFRESH_CRA	Use CRA (Clean Random Access Unit) picture encoding.
V4L2_MPEG_VIDEO_HEVC_REFRESH_IDR	Use IDR (Instantaneous Decoding Refresh) picture encoding.

V4L2_CID_MPEG_VIDEO_HEVC_REFRESH_PERIOD (integer) Selects the refresh period for HEVC encoder. This specifies the number of I pictures between two CRA/IDR pictures. This is valid only if REFRESH_TYPE is not 0.

V4L2_CID_MPEG_VIDEO_HEVC_LOSSLESS_CU (boolean) Indicates HEVC lossless encoding. Setting it to 0 disables lossless encoding. Setting it to 1 enables lossless encoding.

V4L2_CID_MPEG_VIDEO_HEVC_CONST_INTRA_PRED (boolean) Indicates constant intra prediction for HEVC encoder. Specifies the constrained intra prediction in which intra largest coding unit (LCU) prediction is performed by using residual data and decoded samples of neighboring intra LCU only. Setting the value to 1 enables constant intra prediction and setting the value to 0 disables constant intra prediction.

V4L2_CID_MPEG_VIDEO_HEVC_WAVEFRONT (boolean) Indicates wavefront parallel processing for HEVC encoder. Setting it to 0 disables the feature and setting it to 1 enables the wavefront parallel processing.

V4L2_CID_MPEG_VIDEO_HEVC_GENERAL_PB (boolean) Setting the value to 1 enables combination of P and B frame for HEVC encoder.

V4L2_CID_MPEG_VIDEO_HEVC_TEMPORAL_ID (boolean) Indicates temporal identifier for HEVC encoder which is enabled by setting the value to 1.

V4L2_CID_MPEG_VIDEO_HEVC_STRONG_SMOOTHING (boolean) Indicates bi-linear interpolation is conditionally used in the intra prediction filtering process in the CVS when set to 1. Indicates bi-linear interpolation is not used in the CVS when set to 0.

V4L2_CID_MPEG_VIDEO_HEVC_MAX_NUM_MERGE_MV_MINUS1 (integer) Indicates maximum number of merge candidate motion vectors. Values are from 0 to 4.

V4L2_CID_MPEG_VIDEO_HEVC_TMV_PREDICTION (boolean) Indicates temporal motion vector prediction for HEVC encoder. Setting it to 1 enables the prediction. Setting it to 0 disables the prediction.

V4L2_CID_MPEG_VIDEO_HEVC_WITHOUT_STARTCODE (boolean) Specifies if HEVC generates a stream with a size of the length field instead of start code pattern. The size of the length field is configurable through the **V4L2_CID_MPEG_VIDEO_HEVC_SIZE_OF_LENGTH_FIELD** control. Setting the value to 0 disables encoding without startcode pattern. Setting the value to 1 will enables encoding without startcode pattern.

V4L2_CID_MPEG_VIDEO_HEVC_SIZE_OF_LENGTH_FIELD (enum)

enum v4l2_mpeg_video_hevc_size_of_length_field - Indicates the size of length field. This is valid when encoding **WITHOUT_STARTCODE_ENABLE** is enabled.

V4L2_MPEG_VIDEO_HEVC_SIZE_0	Generate start code pattern (Normal).
V4L2_MPEG_VIDEO_HEVC_SIZE_1	Generate size of length field instead of start code pattern and length is 1.
V4L2_MPEG_VIDEO_HEVC_SIZE_2	Generate size of length field instead of start code pattern and length is 2.
V4L2_MPEG_VIDEO_HEVC_SIZE_4	Generate size of length field instead of start code pattern and length is 4.

V4L2_CID_MPEG_VIDEO_HEVC_HIER_CODING_L0_BR (integer) Indicates bit rate for hierarchical coding layer 0 for HEVC encoder.

V4L2_CID_MPEG_VIDEO_HEVC_HIER_CODING_L1_BR (integer) Indicates bit rate for hierarchical coding layer 1 for HEVC encoder.

V4L2_CID_MPEG_VIDEO_HEVC_HIER_CODING_L2_BR (integer) Indicates bit rate for hierarchical coding layer 2 for HEVC encoder.

V4L2_CID_MPEG_VIDEO_HEVC_HIER_CODING_L3_BR (integer) Indicates bit rate for hierarchical coding layer 3 for HEVC encoder.

V4L2_CID_MPEG_VIDEO_HEVC_HIER_CODING_L4_BR (integer) Indicates bit rate for hierarchical coding layer 4 for HEVC encoder.

V4L2_CID_MPEG_VIDEO_HEVC_HIER_CODING_L5_BR (integer) Indicates bit rate for hierarchical coding layer 5 for HEVC encoder.

V4L2_CID_MPEG_VIDEO_HEVC_HIER_CODING_L6_BR (integer) Indicates bit rate for hierarchical coding layer 6 for HEVC encoder.

V4L2_CID_MPEG_VIDEO_REF_NUMBER_FOR_PFRAMES (integer) Selects number of P reference pictures required for HEVC encoder. P-Frame can use 1 or 2 frames for reference.

V4L2_CID_MPEG_VIDEO_PREPEND_SPSPPS_TO_IDR (integer) Indicates whether to generate SPS and PPS at every IDR. Setting it to 0 disables generating SPS and PPS at every IDR. Setting it to one enables generating SPS and PPS at every IDR.

V4L2_CID_MPEG_VIDEO_HEVC_SPS (struct) Specifies the Sequence Parameter Set fields (as extracted from the bitstream) for the associated HEVC slice data. These bitstream pa-

parameters are defined according to *ITU H.265/HEVC*. They are described in section 7.4.3.2 “Sequence parameter set RBSP semantics” of the specification.

v4l2_ctrl_hevc_sps

Table 3: struct v4l2_ctrl_hevc_sps

__u16	pic_width_in_luma_samples	
__u16	pic_height_in_luma_samples	
__u8	bit_depth_luma_minus8	
__u8	bit_depth_chroma_minus8	
__u8	log2_max_pic_order_cnt_lsb_minus4	
__u8	sps_max_dec_pic_buffering_minus1	
__u8	sps_max_num_reorder_pics	
__u8	sps_max_latency_increase_plus1	
__u8	log2_min_luma_coding_block_size_minus3	
__u8	log2_diff_max_min_luma_coding_block_size	
__u8	log2_min_luma_transform_block_size_minus2	
__u8	log2_diff_max_min_luma_transform_block_size	
__u8	max_transform_hierarchy_depth_inter	
__u8	max_transform_hierarchy_depth_intra	
__u8	pcm_sample_bit_depth_luma_minus1	
__u8	pcm_sample_bit_depth_chroma_minus1	
__u8	log2_min_pcm_luma_coding_block_size_minus3	
__u8	log2_diff_max_min_pcm_luma_coding_block_size	
__u8	num_short_term_ref_pic_sets	
__u8	num_long_term_ref_pics_sps	
__u8	chroma_format_idc	
__u8	sps_max_sub_layers_minus1	
__u64	flags	See <i>Sequence Parameter Set Flags</i>

Sequence Parameter Set Flags

V4L2_HEVC_SPS_FLAG_SEPARATE_COLOUR_PLANE	0x00000001	
V4L2_HEVC_SPS_FLAG_SCALING_LIST_ENABLED	0x00000002	
V4L2_HEVC_SPS_FLAG_AMP_ENABLED	0x00000004	
V4L2_HEVC_SPS_FLAG_SAMPLE_ADAPTIVE_OFFSET	0x00000008	
V4L2_HEVC_SPS_FLAG_PCM_ENABLED	0x00000010	
V4L2_HEVC_SPS_FLAG_PCM_LOOP_FILTER_DISABLED	0x00000020	
V4L2_HEVC_SPS_FLAG_LONG_TERM_REF_PICS_PRESENT	0x00000040	
V4L2_HEVC_SPS_FLAG_SPS_TEMPORAL_MVP_ENABLED	0x00000080	
V4L2_HEVC_SPS_FLAG_STRONG_INTRA_SMOOTHING_ENABLED	0x00000100	

V4L2_CID_MPEG_VIDEO_HEVC_PPS (struct) Specifies the Picture Parameter Set fields (as extracted from the bitstream) for the associated HEVC slice data. These bitstream parameters are defined according to *ITU H.265/HEVC*. They are described in section 7.4.3.3 “Picture parameter set RBSP semantics” of the specification.

v4l2_ctrl_hevc_pps

Table 5: struct v4l2_ctrl_hevc_pps

__u8	num_extra_slice_header_bits	
__u8	num_ref_idx_l0_default_active_minus1	Specifies the inferred value of num_ref_idx_l0_active_minus1

Continued on next page

Table 5 - continued from previous page

__u8	num_ref_idx_l1_default_active_minus1	Specifies the inferred value of num_ref_idx_l1_active_minus1
__s8	init_qp_minus26	
__u8	diff_cu_qp_delta_depth	
__s8	pps_cb_qp_offset	
__s8	pps_cr_qp_offset	
__u8	num_tile_columns_minus1	
__u8	num_tile_rows_minus1	
__u8	column_width_minus1[20]	
__u8	row_height_minus1[22]	
__s8	pps_beta_offset_div2	
__s8	pps_tc_offset_div2	
__u8	log2_parallel_merge_level_minus2	
__u8	padding[4]	Applications and drivers must set this to zero.
__u64	flags	See <i>Picture Parameter Set Flags</i>

Picture Parameter Set Flags

V4L2_HEVC_PPS_FLAG_DEPENDENT_SLICE_SEGMENT_ENABLED	0x00000000	
V4L2_HEVC_PPS_FLAG_OUTPUT_FLAG_PRESENT	0x00000002	
V4L2_HEVC_PPS_FLAG_SIGN_DATA_HIDING_ENABLED	0x00000004	
V4L2_HEVC_PPS_FLAG_CABAC_INITTOR_PRESENT	0x00000008	
V4L2_HEVC_PPS_FLAG_CONSTRAINED_TRANSFORM	0x00000010	
V4L2_HEVC_PPS_FLAG_CU_QP_DELTA_ENABLED	0x00000020	
V4L2_HEVC_PPS_FLAG_PPS_SLICE_OFFSETS_PRESENT	0x00000040	
V4L2_HEVC_PPS_FLAG_WEIGHTED_PREDICTION_100	0x00000100	
V4L2_HEVC_PPS_FLAG_WEIGHTED_PREDICTION_200	0x00000200	
V4L2_HEVC_PPS_FLAG_TRANSQUANT_BYPASS_ENABLED	0x00000400	
V4L2_HEVC_PPS_FLAG_TILES_ENABLED	0x00000800	
V4L2_HEVC_PPS_FLAG_ENTROPY_CODING_SYNC_ENABLED	0x00001000	
V4L2_HEVC_PPS_FLAG_LOOP_FILTER_AOBSOLETE	0x00002000	
V4L2_HEVC_PPS_FLAG_PPS_LOOP_FILTER_AOBSOLETE	0x00004000	
V4L2_HEVC_PPS_FLAG_DEBLOCKING_OVERRIDE_ENABLED	0x00008000	
V4L2_HEVC_PPS_FLAG_PPS_DISABLE_DEBLOCKING_FILTER	0x00010000	
V4L2_HEVC_PPS_FLAG_LISTS_MODIFICATION_PRESENT	0x00020000	
V4L2_HEVC_PPS_FLAG_SLICE_SEGMENT_HEADER_EXTENSION_PRESENT	0x00040000	
V4L2_HEVC_PPS_FLAG_DEBLOCKING_FILTER_PRESENT	0x00080000	The presence of deblocking filter control syntax elements in the PPS
V4L2_HEVC_PPS_FLAG_UNIFORM_SPACING	0x00100000	Specifies that tile column boundaries and likewise tile row boundaries are distributed uniformly across the picture

V4L2_CID_MPEG_VIDEO_HEVC_SLICE_PARAMS (struct) Specifies various slice-specific parameters, especially from the NAL unit header, general slice segment header and weighted prediction parameter parts of the bitstream. These bitstream parameters are defined according to *ITU H.265/HEVC*. They are described in section 7.4.7 “General slice segment header semantics” of the specification.

v4l2_ctrl_hevc_slice_params

Table 6: struct v4l2_ctrl_hevc_slice_params

__u32	bit_size	Size (in bits) of the current slice data.
__u32	data_bit_offset	Offset (in bits) to the video data in the current slice data.
__u8	nal_unit_type	
__u8	nuh_temporal_id_plus1	
__u8	slice_type	(V4L2_HEVC_SLICE_TYPE_I, V4L2_HEVC_SLICE_TYPE_P or V4L2_HEVC_SLICE_TYPE_B).
__u8	colour_plane_id	
__u16	slice_pic_order_cnt	
__u8	num_ref_idx_l0_active_minus1	
__u8	num_ref_idx_l1_active_minus1	
__u8	collocated_ref_idx	
__u8	five_minus_max_num_merge_cand	
__s8	slice_qp_delta	
__s8	slice_cb_qp_offset	
__s8	slice_cr_qp_offset	
__s8	slice_act_y_qp_offset	
__s8	slice_act_cb_qp_offset	
__s8	slice_act_cr_qp_offset	
__s8	slice_beta_offset_div2	
__s8	slice_tc_offset_div2	
__u8	pic_struct	
__u32	slice_segment_addr	
__u8	ref_idx_l0[V4L2_HEVC_DPB_ENTRIES_NUM_MAX]	The list of L0 reference elements as indices in the DPB.
__u8	ref_idx_l1[V4L2_HEVC_DPB_ENTRIES_NUM_MAX]	The list of L1 reference elements as indices in the DPB.
__u8	padding	Applications and drivers must set this to zero.
struct v4l2_hevc_pred_weight_table	pred_weight_table	The prediction weight coefficients for inter-picture prediction.
__u64	flags	See <i>Slice Parameters Flags</i>

Slice Parameters Flags

V4L2_HEVC_SLICE_PARAMS_FLAG_SLICE_SAO_LUMA	0x00000001	
V4L2_HEVC_SLICE_PARAMS_FLAG_SLICE_SAO_CHROMA	0x00000002	
V4L2_HEVC_SLICE_PARAMS_FLAG_SLICE_TEMPORAL_MVP_ENABLED	0x00000004	
V4L2_HEVC_SLICE_PARAMS_FLAG_MVD_L1_ZERO	0x00000008	
V4L2_HEVC_SLICE_PARAMS_FLAG_CABAC_INIT	0x00000010	
V4L2_HEVC_SLICE_PARAMS_FLAG_COLLOCATED_FROM_L0	0x00000020	
V4L2_HEVC_SLICE_PARAMS_FLAG_USE_INTEGER_MV	0x00000040	
V4L2_HEVC_SLICE_PARAMS_FLAG_SLICE_DEBLOCKING_FILTER_DISABLED	0x00000080	
V4L2_HEVC_SLICE_PARAMS_FLAG_SLICE_LOOP_FILTER_ACROSS_SLICES_ENABLED	0x00000100	
V4L2_HEVC_SLICE_PARAMS_FLAG_DEPENDENT_SLICE_SEGMENT	0x00000200	

V4L2_CID_MPEG_VIDEO_HEVC_SCALING_MATRIX (struct) Specifies the HEVC scaling matrix parameters used for the scaling process for transform coefficients. These matrix and parameters are defined according to *ITU H.265/HEVC*. They are described in section 7.4.5 “Scaling list data semantics” of the specification.

v4l2_ctrl_hevc_scaling_matrix

Table 7: struct v4l2_ctrl_hevc_scaling_matrix

__u8	scaling_list_4x4[6][16]	Scaling list is used for the scaling process for transform coefficients. The values on each scaling list are expected in raster scan order.
__u8	scaling_list_8x8[6][64]	Scaling list is used for the scaling process for transform coefficients. The values on each scaling list are expected in raster scan order.

Continued on next page

Table 7 - continued from previous page

__u8	scaling_list_16x16[6][64]	Scaling list is used for the scaling process for transform coefficients. The values on each scaling list are expected in raster scan order.
__u8	scaling_list_32x32[2][64]	Scaling list is used for the scaling process for transform coefficients. The values on each scaling list are expected in raster scan order.
__u8	scaling_list_dc_coef_16x16[6]	Scaling list is used for the scaling process for transform coefficients. The values on each scaling list are expected in raster scan order.
__u8	scaling_list_dc_coef_32x32[2]	Scaling list is used for the scaling process for transform coefficients. The values on each scaling list are expected in raster scan order.

v4l2_hevc_dpb_entry

Table 8: struct v4l2_hevc_dpb_entry

__u64	timestamp	Timestamp of the V4L2 capture buffer to use as reference, used with B-coded and P-coded frames. The timestamp refers to the timestamp field in struct v4l2_buffer. Use the v4l2_timeval_to_ns() function to convert the struct timeval in struct v4l2_buffer to a __u64.
__u8	rps	The reference set for the reference frame (V4L2_HEVC_DPB_ENTRY_RPS_ST_CURR_BEFORE, V4L2_HEVC_DPB_ENTRY_RPS_ST_CURR_AFTER or V4L2_HEVC_DPB_ENTRY_RPS_LT_CURR)
__u8	field_pic	Whether the reference is a field picture or a frame.
__u16	pic_order_cnt[2]	The picture order count of the reference. Only the first element of the array is used for frame pictures, while the first element identifies the top field and the second the bottom field in field-coded pictures.
__u8	padding[2]	Applications and drivers must set this to zero.

v4l2_hevc_pred_weight_table

Table 9: struct v4l2_hevc_pred_weight_table

__u8	luma_log2_weight_denom	
__s8	delta_chroma_log2_weight_denom	
__s8	delta_luma_weight_l0[V4L2_HEVC_DPB_ENTRIES_NUM_MAX]	
__s8	luma_offset_l0[V4L2_HEVC_DPB_ENTRIES_NUM_MAX]	
__s8	delta_chroma_weight_l0[V4L2_HEVC_DPB_ENTRIES_NUM_MAX][2]	
__s8	chroma_offset_l0[V4L2_HEVC_DPB_ENTRIES_NUM_MAX][2]	
__s8	delta_luma_weight_l1[V4L2_HEVC_DPB_ENTRIES_NUM_MAX]	
__s8	luma_offset_l1[V4L2_HEVC_DPB_ENTRIES_NUM_MAX]	
__s8	delta_chroma_weight_l1[V4L2_HEVC_DPB_ENTRIES_NUM_MAX][2]	
__s8	chroma_offset_l1[V4L2_HEVC_DPB_ENTRIES_NUM_MAX][2]	
__u8	padding[6]	Applications and drivers must set this to zero.

V4L2_CID_MPEG_VIDEO_HEVC_DECODE_MODE (enum) Specifies the decoding mode to use. Currently exposes slice-based and frame-based decoding but new modes might be added later on. This control is used as a modifier for V4L2_PIX_FMT_HEVC_SLICE pixel format. Applications that support V4L2_PIX_FMT_HEVC_SLICE are required to set this control in order to specify the decoding mode that is expected for the buffer. Drivers may expose a single or multiple decoding modes, depending on what they can support.

Note: This menu control is not yet part of the public kernel API and it is expected to

change.

v4l2_mpeg_video_hevc_decode_mode

V4L2_MPEG_VIDEO_HEVC_DECODE_MODE_SLICE_BASED	0	Decoding is done at the slice granularity. The OUTPUT buffer must contain a single slice.
V4L2_MPEG_VIDEO_HEVC_DECODE_MODE_FRAME_BASED	1	Decoding is done at the frame granularity. The OUTPUT buffer must contain all slices needed to decode the frame. The OUTPUT buffer must also contain both fields.

V4L2_CID_MPEG_VIDEO_HEVC_START_CODE (enum) Specifies the HEVC slice start code expected for each slice. This control is used as a modifier for V4L2_PIX_FMT_HEVC_SLICE pixel format. Applications that support V4L2_PIX_FMT_HEVC_SLICE are required to set this control in order to specify the start code that is expected for the buffer. Drivers may expose a single or multiple start codes, depending on what they can support.

Note: This menu control is not yet part of the public kernel API and it is expected to change.

v4l2_mpeg_video_hevc_start_code

V4L2_MPEG_VIDEO_HEVC_START_CODE_NONE	0	Selecting this value specifies that HEVC slices are passed to the driver without any start code. The bit-stream data should be according to <i>ITU H.265/HEVC</i> 7.3.1.1 General NAL unit syntax, hence contains emulation prevention bytes when required.
V4L2_MPEG_VIDEO_HEVC_START_CODE_ANNEX_B	1	Selecting this value specifies that HEVC slices are expected to be prefixed by Annex B start codes. According to <i>ITU H.265/HEVC</i> valid start codes can be 3-bytes 0x000001 or 4-bytes 0x00000001.

V4L2_CID_MPEG_VIDEO_BASELAYER_PRIORITY_ID (integer) Specifies a priority identifier for the NAL unit, which will be applied to the base layer. By default this value is set to 0 for the base layer, and the next layer will have the priority ID assigned as 1, 2, 3 and so on. The video encoder can't decide the priority id to be applied to a layer, so this has to come from client. This is applicable to H264 and valid Range is from 0 to 63. Source Rec. ITU-T H.264 (06/2019); G.7.4.1.1, G.8.8.1.

V4L2_CID_MPEG_VIDEO_LTR_COUNT (integer) Specifies the maximum number of Long Term Reference (LTR) frames at any given time that the encoder can keep. This is applicable to the H264 and HEVC encoders.

V4L2_CID_MPEG_VIDEO_FRAME_LTR_INDEX (integer) After setting this control the frame that will be queued next will be marked as a Long Term Reference (LTR) frame and given this

LTR index which ranges from 0 to LTR_COUNT-1. This is applicable to the H264 and HEVC encoders. Source Rec. ITU-T H.264 (06/2019); Table 7.9

V4L2_CID_MPEG_VIDEO_USE_LTR_FRAMES (bitmask) Specifies the Long Term Reference (LTR) frame(s) to be used for encoding the next frame queued after setting this control. This provides a bitmask which consists of bits [0, LTR_COUNT-1]. This is applicable to the H264 and HEVC encoders.

V4L2_CID_MPEG_VIDEO_HEVC_DECODE_PARAMS (struct) Specifies various decode parameters, especially the references picture order count (POC) for all the lists (short, long, before, current, after) and the number of entries for each of them. These parameters are defined according to *ITU H.265/HEVC*. They are described in section 8.3 “Slice decoding process” of the specification.

v4l2_ctrl_hevc_decode_params

Table 10: struct v4l2_hevc_decode_params

__s32	pic_order_cnt_val	PicOrderC
__u8	num_active_dpb_entries	The num
struct v4l2_hevc_dpb_entry	dpb[V4L2_HEVC_DPB_ENTRIES_NUM_MAX]	The dec
__u8	num_poc_st_curr_before	The num
__u8	num_poc_st_curr_after	The num
__u8	num_poc_lt_curr	The num
__u8	poc_st_curr_before[V4L2_HEVC_DPB_ENTRIES_NUM_MAX]	PocStCu
__u8	poc_st_curr_after[V4L2_HEVC_DPB_ENTRIES_NUM_MAX]	PocStCu
__u8	poc_lt_curr[V4L2_HEVC_DPB_ENTRIES_NUM_MAX]	PocLtCu
__u64	flags	See <i>Dec</i>

Decode Parameters Flags

V4L2_HEVC_DECODE_PARAM_FLAG_IRAP_PIC	0x00000001	
V4L2_HEVC_DECODE_PARAM_FLAG_IDR_PIC	0x00000002	
V4L2_HEVC_DECODE_PARAM_FLAG_NO_OUTPUT_OF_PRIOR	0x00000004	

3.2.1.16 Stateless Codec Control Reference

The Stateless Codec control class is intended to support stateless decoder and encoders (i.e. hardware accelerators).

These drivers are typically supported by the *Memory-to-memory Stateless Video Decoder Interface*, and deal with parsed pixel formats such as V4L2_PIX_FMT_H264_SLICE.

Stateless Codec Control ID

V4L2_CID_CODEC_STATELESS_CLASS (**class**) The Stateless Codec class descriptor.

V4L2_CID_STATELESS_H264_SPS (**struct**) Specifies the sequence parameter set (as extracted from the bitstream) for the associated H264 slice data. This includes the necessary parameters for configuring a stateless hardware decoding pipeline for H264. The bitstream parameters are defined according to *ITU-T Rec. H.264 Specification (04/2017 Edition)*, section 7.4.2.1.1 “Sequence Parameter Set Data Semantics”. For further documentation, refer to the above specification, unless there is an explicit comment stating otherwise.

v4l2_ctrl_h264_sps

Table 12: struct v4l2_ctrl_h264_sps

__u8	profile_idc	
__u8	constraint_set_flags	See <i>Sequence Parameter Set Constraints Set Flags</i>
__u8	level_idc	
__u8	seq_parameter_set_id	
__u8	chroma_format_idc	
__u8	bit_depth_luma_minus8	
__u8	bit_depth_chroma_minus8	
__u8	log2_max_frame_num_minus4	
__u8	pic_order_cnt_type	
__u8	log2_max_pic_order_cnt_lsb_minus4	
__u8	max_num_ref_frames	
__u8	num_ref_frames_in_pic_order_cnt_cycle	
__s32	offset_for_ref_frame[255]	
__s32	offset_for_non_ref_pic	
__s32	offset_for_top_to_bottom_field	
__u16	pic_width_in_mbs_minus1	
__u16	pic_height_in_map_units_minus1	
__u32	flags	See <i>Sequence Parameter Set Flags</i>

Sequence Parameter Set Constraints Set Flags

V4L2_H264_SPS_CONSTRAINT_SET0_FLAG	0x00000001	
V4L2_H264_SPS_CONSTRAINT_SET1_FLAG	0x00000002	
V4L2_H264_SPS_CONSTRAINT_SET2_FLAG	0x00000004	
V4L2_H264_SPS_CONSTRAINT_SET3_FLAG	0x00000008	
V4L2_H264_SPS_CONSTRAINT_SET4_FLAG	0x00000010	
V4L2_H264_SPS_CONSTRAINT_SET5_FLAG	0x00000020	

Sequence Parameter Set Flags

V4L2_H264_SPS_FLAG_SEPARATE_COLOUR_PLANE	0x00000001	
V4L2_H264_SPS_FLAG_QPPRIME_Y_ZERO_TRANSFORM_BYPASS	0x00000002	
V4L2_H264_SPS_FLAG_DELTA_PIC_ORDER_ALWAYS_ZERO	0x00000004	
V4L2_H264_SPS_FLAG_GAPS_IN_FRAME_NUM_VALUE_ALLOWED	0x00000008	
V4L2_H264_SPS_FLAG_FRAME_MBS_ONLY	0x00000010	
V4L2_H264_SPS_FLAG_MB_ADAPTIVE_FRAME_FIELD	0x00000020	

Continued on next page

Table 14 - continued from previous page

V4L2_H264_SPS_FLAG_DIRECT_8X8_INFERENCE	0x00000040
---	------------

V4L2_CID_STATELESS_H264_PPS (struct) Specifies the picture parameter set (as extracted from the bitstream) for the associated H264 slice data. This includes the necessary parameters for configuring a stateless hardware decoding pipeline for H264. The bitstream parameters are defined according to *ITU-T Rec. H.264 Specification (04/2017 Edition)*, section 7.4.2.2 “Picture Parameter Set RBSP Semantics”. For further documentation, refer to the above specification, unless there is an explicit comment stating otherwise.

v4l2_ctrl_h264_pps

Table 15: struct v4l2_ctrl_h264_pps

__u8	pic_parameter_set_id	
__u8	seq_parameter_set_id	
__u8	num_slice_groups_minus1	
__u8	num_ref_idx_l0_default_active_minus1	
__u8	num_ref_idx_l1_default_active_minus1	
__u8	weighted_bipred_idc	
__s8	pic_init_qp_minus26	
__s8	pic_init_qs_minus26	
__s8	chroma_qp_index_offset	
__s8	second_chroma_qp_index_offset	
__u16	flags	See <i>Picture Parameter Set Flags</i>

Picture Parameter Set Flags

V4L2_H264_PPS_FLAG_ENTROPY_CODING_MODE	0x0001	
V4L2_H264_PPS_FLAG_BOTTOM_FIELD_PIC_ORDER_IN_FRAME_PRESENT	0x0002	
V4L2_H264_PPS_FLAG_WEIGHTED_PRED	0x0004	
V4L2_H264_PPS_FLAG_DEBLOCKING_FILTER_CONTROL_PRESENT	0x0008	
V4L2_H264_PPS_FLAG_CONSTRAINED_INTRA_PRED	0x0010	
V4L2_H264_PPS_FLAG_REDUNDANT_PIC_CNT_PRESENT	0x0020	
V4L2_H264_PPS_FLAG_TRANSFORM_8X8_MODE	0x0040	
V4L2_H264_PPS_FLAG_SCALING_MATRIX_PRESENT	0x0080	V4L2_CID_STATELESS_H264_SCALING_MATRIX must be used for this picture.

V4L2_CID_STATELESS_H264_SCALING_MATRIX (struct) Specifies the scaling matrix (as extracted from the bitstream) for the associated H264 slice data. The bitstream parameters are defined according to *ITU-T Rec. H.264 Specification (04/2017 Edition)*, section 7.4.2.1.1.1 “Scaling List Semantics”. For further documentation, refer to the above specification, unless there is an explicit comment stating otherwise.

v4l2_ctrl_h264_scaling_matrix

Table 16: struct v4l2_ctrl_h264_scaling_matrix

__u8	scaling_list_4x4[6][16]	Scaling matrix after applying the inverse scanning process. Expected list order is Intra Y, Intra Cb, Intra Cr, Inter Y, Inter Cb, Inter Cr. The values on each scaling list are expected in raster scan order.
__u8	scaling_list_8x8[6][64]	Scaling matrix after applying the inverse scanning process. Expected list order is Intra Y, Inter Y, Intra Cb, Inter Cb, Intra Cr, Inter Cr. The values on each scaling list are expected in raster scan order.

V4L2_CID_STATELESS_H264_SLICE_PARAMS (struct) Specifies the slice parameters (as extracted from the bitstream) for the associated H264 slice data. This includes the necessary parameters for con-

figuring a stateless hardware decoding pipeline for H264. The bitstream parameters are defined according to *ITU-T Rec. H.264 Specification (04/2017 Edition)*, section 7.4.3 “Slice Header Semantics”. For further documentation, refer to the above specification, unless there is an explicit comment stating otherwise.

v4l2_ctrl_h264_slice_params

Table 17: struct v4l2_ctrl_h264_slice_params

__u32	header_bit_size	Offset in bits to slice_data() from the beginning of this slice.
__u32	first_mb_in_slice	
__u8	slice_type	
__u8	colour_plane_id	
__u8	redundant_pic_cnt	
__u8	cabac_init_idc	
__s8	slice_qp_delta	
__s8	slice_qs_delta	
__u8	disable_deblocking_filter_idc	
__s8	slice_alpha_c0_offset_div2	
__s8	slice_beta_offset_div2	
__u8	num_ref_idx_l0_active_minus1	If num_ref_idx_active_override_flag is not set, this field must be set to the value of num_ref_idx_l0_default_active_minus1
__u8	num_ref_idx_l1_active_minus1	If num_ref_idx_active_override_flag is not set, this field must be set to the value of num_ref_idx_l1_default_active_minus1
__u8	reserved	Applications and drivers must set this to zero.
struct v4l2_h264_reference	ref_pic_list0[32]	Reference picture list after applying the per-slice modifications
struct v4l2_h264_reference	ref_pic_list1[32]	Reference picture list after applying the per-slice modifications
__u32	flags	See Slice Parameter Flags

Slice Parameter Set Flags

V4L2_H264_SLICE_FLAG_DIRECT_SPATIAL_MV_PRED	0x00000001	
V4L2_H264_SLICE_FLAG_SP_FOR_SWITCH	0x00000002	

V4L2_CID_STATELESS_H264_PRED_WEIGHTS (struct) Prediction weight table defined according to *ITU-T Rec. H.264 Specification (04/2017 Edition)*, section 7.4.3.2 “Prediction Weight Table Semantics”. The prediction weight table must be passed by applications under the conditions explained in section 7.3.3 “Slice header syntax”.

v4l2_ctrl_h264_pred_weights

Table 19: struct v4l2_ctrl_h264_pred_weights

__u16	luma_log2_weight_denom	
__u16	chroma_log2_weight_denom	
struct v4l2_h264_weight_factors	weight_factors[2]	The weight factors at index 0 are the weight factors for the reference list 0, the one at index 1 for the reference list 1.

v4l2_h264_weight_factors

Table 20: struct v4l2_h264_weight_factors

__s16	luma_weight[32]	
__s16	luma_offset[32]	
__s16	chroma_weight[32][2]	
__s16	chroma_offset[32][2]	

Picture Reference

v4l2_h264_reference

Table 21: struct v4l2_h264_reference

__u8	fields	Specifies how the picture is referenced. See <i>Reference Fields</i>
__u8	index	Index into the <i>v4l2_ctrl_h264_decode_params.dpb</i> array.

Reference Fields

V4L2_H264_TOP_FIELD_REF	0x1	The top field in field pair is used for short-term reference.
V4L2_H264_BOTTOM_FIELD_REF	0x2	The bottom field in field pair is used for short-term reference.
V4L2_H264_FRAME_REF	0x3	The frame (or the top/bottom fields, if it's a field pair) is used for short-term reference.

V4L2_CID_STATELESS_H264_DECODE_PARAMS (struct) Specifies the decode parameters (as extracted from the bitstream) for the associated H264 slice data. This includes the necessary parameters for configuring a stateless hardware decoding pipeline for H264. The bitstream parameters are defined according to *ITU-T Rec. H.264 Specification (04/2017 Edition)*. For further documentation, refer to the above specification, unless there is an explicit comment stating otherwise.

v4l2_ctrl_h264_decode_params

Table 22: struct v4l2_ctrl_h264_decode_params

struct <i>v4l2_h264_dpb_entry</i>	dpb[16]	
__u16	nal_ref_idc	NAL reference ID value coming from the NAL Unit header
__u16	frame_num	
__s32	top_field_order_cnt	Picture Order Count for the coded top field
__s32	bottom_field_order_cnt	Picture Order Count for the coded bottom field
__u16	idr_pic_id	
__u16	pic_order_cnt_lsb	
__s32	delta_pic_order_cnt_bottom	
__s32	delta_pic_order_cnt0	
__s32	delta_pic_order_cnt1	
__u32	dec_ref_pic_marking_bit_size	Size in bits of the dec_ref_pic_marking() syntax element.
__u32	pic_order_cnt_bit_size	Combined size in bits of the picture order count related syntax elements: pic_order_cnt_lsb, delta_pic_order_cnt_bottom, delta_pic_order_cnt0, and delta_pic_order_cnt1.
__u32	slice_group_change_cycle	
__u32	reserved	Applications and drivers must set this to zero.
__u32	flags	See <i>Decode Parameters Flags</i>

Decode Parameters Flags

V4L2_H264_DECODE_PARAM_FLAG_IDR_PIC	0x00000001	That picture is an IDR picture
V4L2_H264_DECODE_PARAM_FLAG_FIELD_PIC	0x00000002	
V4L2_H264_DECODE_PARAM_FLAG_BOTTOM_FIELD	0x00000004	

v4l2_h264_dpb_entry

Table 23: struct v4l2_h264_dpb_entry

__u64	reference_ts	Timestamp of the V4L2 capture buffer to use as reference, used with B-coded and P-coded frames. The timestamp refers to the timestamp field in struct v4l2_buffer. Use the v4l2_timeval_to_ns() function to convert the struct timeval in struct v4l2_buffer to a __u64.
__u32	pic_num	
__u16	frame_num	
__u8	fields	Specifies how the DPB entry is referenced. See <i>Reference Fields</i>
__u8	reserved[5]	Applications and drivers must set this to zero.
__s32	top_field_order_cnt	
__s32	bottom_field_order_cnt	
__u32	flags	See <i>DPB Entry Flags</i>

DPB Entries Flags

V4L2_H264_DPB_ENTRY_FLAG_VALID	0x00000001	The DPB entry is valid (non-empty) and should be considered.
V4L2_H264_DPB_ENTRY_FLAG_ACTIVE	0x00000002	The DPB entry is used for reference.
V4L2_H264_DPB_ENTRY_FLAG_LONG_TERM	0x00000004	The DPB entry is used for long-term reference.
V4L2_H264_DPB_ENTRY_FLAG_FIELD	0x00000008	The DPB entry is a single field or a complementary field pair.

V4L2_CID_STATELESS_H264_DECODE_MODE (enum) Specifies the decoding mode to use. Currently exposes slice-based and frame-based decoding but new modes might be added later on. This control is used as a modifier for V4L2_PIX_FMT_H264_SLICE pixel format. Applications that support V4L2_PIX_FMT_H264_SLICE are required to set this control in order to specify the decoding mode that is expected for the buffer. Drivers may expose a single or multiple decoding modes, depending on what they can support.

v4l2_stateless_h264_decode_mode

V4L2_STATELESS_H264_DECODE_MODE_SLICE_BASED	0	Decoding is done at the slice granularity. The OUTPUT buffer must contain a single slice. When this mode is selected, the V4L2_CID_STATELESS_H264_SLICE_PARAMS control shall be set. When multiple slices compose a frame, use of V4L2_BUF_CAP_SUPPORTS_M2M_HOLD_CAPTURE_BUF flag is required.
V4L2_STATELESS_H264_DECODE_MODE_FRAME_BASED	1	Decoding is done at the frame granularity. The OUTPUT buffer must contain all slices needed to decode the frame. The OUTPUT buffer must also contain both fields. This mode will be supported by devices that parse the slice(s) header(s) in hardware. When this mode is selected, the V4L2_CID_STATELESS_H264_SLICE_PARAMS control shall not be set.

V4L2_CID_STATELESS_H264_START_CODE (enum) Specifies the H264 slice start code expected for each slice. This control is used as a modifier for V4L2_PIX_FMT_H264_SLICE pixel format. Applications that support V4L2_PIX_FMT_H264_SLICE are required to set this control in order to specify the start code that is expected for the buffer. Drivers may expose a single or multiple start codes, depending on what they can support.

v4l2_stateless_h264_start_code

V4L2_STATELESS_H264_START_CODE_NONE	0	Selecting this value specifies that H264 slices are passed to the driver without any start code. The bitstream data should be according to <i>ITU-T Rec. H.264 Specification (04/2017 Edition)</i> 7.3.1 NAL unit syntax, hence contains emulation prevention bytes when required.
V4L2_STATELESS_H264_START_CODE_ANNEX_B	1	Selecting this value specifies that H264 slices are expected to be prefixed by Annex B start codes. According to <i>ITU-T Rec. H.264 Specification (04/2017 Edition)</i> valid start codes can be 3-bytes 0x000001 or 4-bytes 0x00000001.

V4L2_CID_STATELESS_FWHT_PARAMS (struct) Specifies the FWHT (Fast Walsh Hadamard Transform) parameters (as extracted from the bitstream) for the associated FWHT data. This includes the necessary parameters for configuring a stateless hardware decoding pipeline for FWHT. This codec is specific to the vicodec test driver.

v4l2_ctrl_fwht_params

Table 24: struct v4l2_ctrl_fwht_params

__u64	backward_ref_ts	Timestamp of the V4L2 capture buffer to use as backward reference, used with P-coded frames. The timestamp refers to the timestamp field in struct v4l2_buffer. Use the v4l2_timeval_to_ns() function to convert the struct timeval in struct v4l2_buffer to a __u64.
__u32	version	The version of the codec. Set to V4L2_FWHT_VERSION.
__u32	width	The width of the frame.
__u32	height	The height of the frame.
__u32	flags	The flags of the frame, see <i>FWHT Flags</i> .
__u32	colorspace	The colorspace of the frame, from enum v4l2_colorspace.
__u32	xfer_func	The transfer function, from enum v4l2_xfer_func.
__u32	ycbcr_enc	The Y'CbCr encoding, from enum v4l2_ycbcr_encoding.
__u32	quantization	The quantization range, from enum v4l2_quantization.

FWHT Flags

V4L2_FWHT_FL_IS_INTERLACED	0x00000001	Set if this is an interlaced format.
V4L2_FWHT_FL_IS_BOTTOM_FIRST	0x00000002	Set if this is a bottom-first (NTSC) interlaced format.
V4L2_FWHT_FL_IS_ALTERNATE	0x00000004	Set if each 'frame' contains just one field.
V4L2_FWHT_FL_IS_BOTTOM_FIELD	0x00000008	If V4L2_FWHT_FL_IS_ALTERNATE was set, then this is set if this 'frame' is the bottom field, else it is the top field.
V4L2_FWHT_FL_LUMA_IS_UNCOMPRESSED	0x00000010	Set if the Y' (luma) plane is uncompressed.
V4L2_FWHT_FL_CB_IS_UNCOMPRESSED	0x00000020	Set if the Cb plane is uncompressed.
V4L2_FWHT_FL_CR_IS_UNCOMPRESSED	0x00000040	Set if the Cr plane is uncompressed.
V4L2_FWHT_FL_CHROMA_FULL_HEIGHT	0x00000080	Set if the chroma plane has the same height as the luma plane, else the chroma plane is half the height of the luma plane.
V4L2_FWHT_FL_CHROMA_FULL_WIDTH	0x00000100	Set if the chroma plane has the same width as the luma plane, else the chroma plane is half the width of the luma plane.
V4L2_FWHT_FL_ALPHA_IS_UNCOMPRESSED	0x00000200	Set if the alpha plane is uncompressed.
V4L2_FWHT_FL_I_FRAME	0x00000400	Set if this is an I-frame.
V4L2_FWHT_FL_COMPONENTS_NUM_MSK	0x00070000	The number of color components minus one.
V4L2_FWHT_FL_PIXENC_MSK	0x00180000	The mask for the pixel encoding.
V4L2_FWHT_FL_PIXENC_YUV	0x00080000	Set if the pixel encoding is YUV.
V4L2_FWHT_FL_PIXENC_RGB	0x00100000	Set if the pixel encoding is RGB.
V4L2_FWHT_FL_PIXENC_HSV	0x00180000	Set if the pixel encoding is HSV.

V4L2_CID_STATELESS_VP8_FRAME (struct) Specifies the frame parameters for the associated VP8 parsed frame data. This includes the necessary parameters for configuring a stateless hardware decoding pipeline for VP8. The bitstream parameters are defined according to *VP8*.

v4l2_ctrl_vp8_frame

Table 25: struct v4l2_ctrl_vp8_frame

struct v4l2_vp8_segment	segment	Structure with segment-based adjustments metadata.
struct v4l2_vp8_loop_filter	lf	Structure with loop filter level adjustments metadata.

Continued on next page

Table 25 - continued from previous page

<code>struct v4l2_vp8_quantization</code>	<code>quant</code>	Structure with VP8 dequantization indices metadata.
<code>struct v4l2_vp8_entropy</code>	<code>entropy</code>	Structure with VP8 entropy coder probabilities metadata.
<code>struct v4l2_vp8_entropy_coder_state</code>	<code>coder_state</code>	Structure with VP8 entropy coder state.
<code>__u16</code>	<code>width</code>	The width of the frame. Must be set for all frames.
<code>__u16</code>	<code>height</code>	The height of the frame. Must be set for all frames.
<code>__u8</code>	<code>horizontal_scale</code>	Horizontal scaling factor.
<code>__u8</code>	<code>vertical_scaling_factor</code>	Vertical scale.
<code>__u8</code>	<code>version</code>	Bitstream version.
<code>__u8</code>	<code>prob_skip_false</code>	Indicates the probability that the macroblock is not skipped.
<code>__u8</code>	<code>prob_intra</code>	Indicates the probability that a macroblock is intra-predicted.
<code>__u8</code>	<code>prob_last</code>	Indicates the probability that the last reference frame is used for inter-prediction
<code>__u8</code>	<code>prob_gf</code>	Indicates the probability that the golden reference frame is used for inter-prediction
<code>__u8</code>	<code>num_dct_parts</code>	Number of DCT coefficients partitions. Must be one of: 1, 2, 4, or 8.
<code>__u32</code>	<code>first_part_size</code>	Size of the first partition, i.e. the control partition.
<code>__u32</code>	<code>first_part_header_bits</code>	Size in bits of the first partition header portion.
<code>__u32</code>	<code>dct_part_sizes[8]</code>	DCT coefficients sizes.
<code>__u64</code>	<code>last_frame_ts</code>	Timestamp for the V4L2 capture buffer to use as last reference frame, used with inter-coded frames. The timestamp refers to the timestamp field in struct <code>v4l2_buffer</code> . Use the <code>v4l2_timeval_to_ns()</code> function to convert the struct <code>timeval</code> in struct <code>v4l2_buffer</code> to a <code>__u64</code> .
<code>__u64</code>	<code>golden_frame_ts</code>	Timestamp for the V4L2 capture buffer to use as last reference frame, used with inter-coded frames. The timestamp refers to the timestamp field in struct <code>v4l2_buffer</code> . Use the <code>v4l2_timeval_to_ns()</code> function to convert the struct <code>timeval</code> in struct <code>v4l2_buffer</code> to a <code>__u64</code> .

Continued on next page

Table 25 - continued from previous page

__u64	alt_frame_ts	Timestamp for the V4L2 capture buffer to use as alternate reference frame, used with inter-coded frames. The timestamp refers to the timestamp field in struct v4l2_buffer. Use the v4l2_timeval_to_ns() function to convert the struct timeval in struct v4l2_buffer to a __u64.
__u64	flags	See <i>Frame Flags</i>

Frame Flags

V4L2_VP8_FRAME_FLAG_KEY_FRAME	0x01	Indicates if the frame is a key frame.
V4L2_VP8_FRAME_FLAG_EXPERIMENTAL	0x02	Experimental bitstream.
V4L2_VP8_FRAME_FLAG_SHOW_FRAME	0x04	Show frame flag, indicates if the frame is for display.
V4L2_VP8_FRAME_FLAG_MB_NO_SKIP_COEFF	0x08	Enable/disable skipping of macroblocks with no non-zero coefficients.
V4L2_VP8_FRAME_FLAG_SIGN_BIAS_GOLDEN	0x10	Sign of motion vectors when the golden frame is referenced.
V4L2_VP8_FRAME_FLAG_SIGN_BIAS_ALT	0x20	Sign of motion vectors when the alt frame is referenced.

v4l2_vp8_entropy_coder_state

Table 27: struct v4l2_vp8_entropy_coder_state

__u8	range	coder state value for "Range"
__u8	value	coder state value for "Value"
__u8	bit_count	number of bits left.
__u8	padding	Applications and drivers must set this to zero.

v4l2_vp8_segment

Table 28: struct v4l2_vp8_segment

__s8	quant_update[4]	Signed quantizer value update.
__s8	lf_update[4]	Signed loop filter level value update.
__u8	segment_probs[3]	Segment probabilities.
__u8	padding	Applications and drivers must set this to zero.
__u32	flags	See <i>Segment Flags</i>

Segment Flags

V4L2_VP8_SEGMENT_FLAG_ENABLED	0x01	Enable/disable segment-based adjustments.
V4L2_VP8_SEGMENT_FLAG_UPDATE_MAP	0x02	Indicates if the macroblock segmentation map is updated in this frame.
V4L2_VP8_SEGMENT_FLAG_UPDATE_FEATURE_DATA	0x04	Indicates if the segment feature data is updated in this frame.
V4L2_VP8_SEGMENT_FLAG_DELTA_VALUE_MODE	0x08	If is set, the segment feature data mode is delta-value. If cleared, it's absolute-value.

v4l2_vp8_loop_filter

Table 29: struct v4l2_vp8_loop_filter

__s8	ref_frm_delta[4]	Reference adjustment (signed) delta value.
__s8	mb_mode_delta[4]	Macroblock prediction mode adjustment (signed) delta value.
__u8	sharpness_level	Sharpness level
__u8	level	Filter level
__u16	padding	Applications and drivers must set this to zero.
__u32	flags	See <i>Loop Filter Flags</i>

Loop Filter Flags

V4L2_VP8_LF_ADJ_ENABLE	0x01	Enable/disable macroblock-level loop filter adjustment.
V4L2_VP8_LF_DELTA_UPDATE	0x02	Indicates if the delta values used in an adjustment are updated.
V4L2_VP8_LF_FILTER_TYPE_SIMPLE	0x04	If set, indicates the filter type is simple. If cleared, the filter type is normal.

v4l2_vp8_quantization

Table 30: struct v4l2_vp8_quantization

__u8	y_ac_qi	Luma AC coefficient table index.
__s8	y_dc_delta	Luma DC delta vaue.
__s8	y2_dc_delta	Y2 block DC delta value.
__s8	y2_ac_delta	Y2 block AC delta value.
__s8	uv_dc_delta	Chroma DC delta value.
__s8	uv_ac_delta	Chroma AC delta value.
__u16	padding	Applications and drivers must set this to zero.

v4l2_vp8_entropy

Table 31: struct v4l2_vp8_entropy

__u8	coeff_probs[4][8][3][11]	Coefficient update probabilities.
__u8	y_mode_probs[4]	Luma mode update probabilities.
__u8	uv_mode_probs[3]	Chroma mode update probabilities.
__u8	mv_probs[2][19]	MV decoding update probabilities.
__u8	padding[3]	Applications and drivers must set this to zero.

V4L2_CID_STATELESS_MPEG2_SEQUENCE (struct) Specifies the sequence parameters (as extracted from the bitstream) for the associated MPEG-2 slice data. This includes fields matching the syntax elements from the sequence header and sequence extension parts of the bitstream as specified by *ISO 13818-2*.

v4l2_ctrl_mpeg2_sequence

Table 32: struct v4l2_ctrl_mpeg2_sequence

__u16	horizontal_size	The width of the displayable part of the frame's luminance component.
__u16	vertical_size	The height of the displayable part of the frame's luminance component.
__u32	vbv_buffer_size	Used to calculate the required size of the video buffering verifier, defined (in bits) as: $16 * 1024 * \text{vbv_buffer_size}$.
__u16	profile_and_level_indication	The current profile and level indication as extracted from the bitstream.
__u8	chroma_format	The chrominance sub-sampling format (1: 4:2:0, 2: 4:2:2, 3: 4:4:4).
__u8	flags	See <i>MPEG-2 Sequence Flags</i> .

MPEG-2 Sequence Flags

V4L2_MPEG2_SEQ_FLAG_PROGRESSIVE	0x01	Indication that all the frames for the sequence are progressive
---------------------------------	------	---

V4L2_CID_STATELESS_MPEG2_PICTURE (struct) Specifies the picture parameters (as extracted from the bitstream) for the associated MPEG-2 slice data. This includes fields matching the syntax elements from the picture header and picture coding extension parts of the bitstream as specified by *ISO 13818-2*.

v4l2_ctrl_mpeg2_picture

Table 34: struct v4l2_ctrl_mpeg2_picture

__u64	backward_ref_ts	Timestamp of the V4L2 capture buffer to use as backward reference, used with B-coded and P-coded frames. The timestamp refers to the timestamp field in struct v4l2_buffer. Use the v4l2_timeval_to_ns() function to convert the struct timeval in struct v4l2_buffer to a __u64.
__u64	forward_ref_ts	Timestamp for the V4L2 capture buffer to use as forward reference, used with B-coded frames. The timestamp refers to the timestamp field in struct v4l2_buffer. Use the v4l2_timeval_to_ns() function to convert the struct timeval in struct v4l2_buffer to a __u64.
__u32	flags	See <i>MPEG-2 Picture Flags</i> .
__u8	f_code[2][2]	Motion vector codes.
__u8	picture_coding_type	Picture coding type for the frame covered by the current slice (V4L2_MPEG2_PIC_CODING_TYPE_I, V4L2_MPEG2_PIC_CODING_TYPE_P or V4L2_MPEG2_PIC_CODING_TYPE_B).
__u8	picture_structure	Picture structure (1: interlaced top field, 2: interlaced bottom field, 3: progressive frame).
__u8	intra_dc_precision	Precision of Discrete Cosine transform (0: 8 bits precision, 1: 9 bits precision, 2: 10 bits precision, 3: 11 bits precision).
__u8	reserved[5]	Applications and drivers must set this to zero.

MPEG-2 Picture Flags

V4L2_MPEG2_PIC_FLAG_TOP_FIELD_FIRST	0x00000001	If set and it's an interlaced stream, top field is outp
V4L2_MPEG2_PIC_FLAG_FRAME_PRED_DCT	0x00000002	If set only frame-DCT and frame prediction are use
V4L2_MPEG2_PIC_FLAG_CONCEALMENT_MV	0x00000004	If set motion vectors are coded for intra macroblo
V4L2_MPEG2_PIC_FLAG_Q_SCALE_TYPE	0x00000008	This flag affects the inverse quantization process.
V4L2_MPEG2_PIC_FLAG_INTRA_VLC	0x00000010	This flag affects the decoding of transform coeffici
V4L2_MPEG2_PIC_FLAG_ALT_SCAN	0x00000020	This flag affects the decoding of transform coeffici
V4L2_MPEG2_PIC_FLAG_REPEAT_FIRST	0x00000040	This flag affects the decoding process of progressi
V4L2_MPEG2_PIC_FLAG_PROGRESSIVE	0x00000080	Indicates whether the current frame is progressive

V4L2_CID_STATELESS_MPEG2_QUANTISATION (struct) Specifies quantisation matrices, in zigzag scanning order, for the associated MPEG-2 slice data. This control is initialized by the kernel to the matrices default values. If a bitstream transmits a user-defined quantisation matrices load, applications are expected to use this control. Applications are also expected to set the control loading the default values, if the quantisation matrices need to be reset, for instance on a sequence header. This process is specified by section 6.3.7. "Quant matrix extension" of the specification.

v4l2_ctrl_mpeg2_quantisation

Table 36: struct v4l2_ctrl_mpeg2_quantisation

__u8	intra_quantiser_matrix[64]	The quantisation matrix coefficients for intra-coded frames, in zigzag scanning order. It is relevant for both luma and chroma components, although it can be superseded by the chroma-specific matrix for non-4:2:0 YUV formats.
__u8	non_intra_quantiser_matrix[64]	The quantisation matrix coefficients for non-intra-coded frames, in zigzag scanning order. It is relevant for both luma and chroma components, although it can be superseded by the chroma-specific matrix for non-4:2:0 YUV formats.
__u8	chroma_intra_quantiser_matrix[64]	The quantisation matrix coefficients for the chrominance component of intra-coded frames, in zigzag scanning order. Only relevant for non-4:2:0 YUV formats.
__u8	chroma_non_intra_quantiser_matrix[64]	The quantisation matrix coefficients for the chrominance component of non-intra-coded frames, in zigzag scanning order. Only relevant for non-4:2:0 YUV formats.

V4L2_CID_STATELESS_VP9_COMPRESSED_HDR (struct) Stores VP9 probabilities updates as parsed from the current compressed frame header. A value of zero in an array element means no update of the relevant probability. Motion vector-related updates contain a new value or zero. All other updates contain values translated with `inv_map_table[]` (see 6.3.5 in *VP9*).

v4l2_ctrl_vp9_compressed_hdr

Table 37: struct v4l2_ctrl_vp9_compressed_hdr

__u8	tx_mode	Specifies the TX mode. See <i>TX Mode</i> for more details.
__u8	tx8[2][1]	TX 8x8 probabilities delta.

Continued on next page

Table 37 - continued from previous page

__u8	tx16[2][2]	TX 16x16 probabilities delta.
__u8	tx32[2][3]	TX 32x32 probabilities delta.
__u8	coef[4][2][2][6][6][3]	Coefficient probabilities delta.
__u8	skip[3]	Skip probabilities delta.
__u8	inter_mode[7][3]	Inter prediction mode probabilities delta.
__u8	interp_filter[4][2]	Interpolation filter probabilities delta.
__u8	is_inter[4]	Is inter-block probabilities delta.
__u8	comp_mode[5]	Compound prediction mode probabilities delta.
__u8	single_ref[5][2]	Single reference probabilities delta.
__u8	comp_ref[5]	Compound reference probabilities delta.
__u8	y_mode[4][9]	Y prediction mode probabilities delta.
__u8	uv_mode[10][9]	UV prediction mode probabilities delta.
__u8	partition[16][3]	Partition probabilities delta.
__u8	mv.joint[3]	Motion vector joint probabilities delta.
__u8	mv.sign[2]	Motion vector sign probabilities delta.
__u8	mv.classes[2][10]	Motion vector class probabilities delta.
__u8	mv.class0_bit[2]	Motion vector class0 bit probabilities delta.
__u8	mv.bits[2][10]	Motion vector bits probabilities delta.
__u8	mv.class0_fr[2][2][3]	Motion vector class0 fractional bit probabilities delta.
__u8	mv.fr[2][3]	Motion vector fractional bit probabilities delta.
__u8	mv.class0_hp[2]	Motion vector class0 high precision fractional bit probabilities delta.
__u8	mv.hp[2]	Motion vector high precision fractional bit probabilities delta.

TX Mode

V4L2_VP9_TX_MODE_ONLY_4X4	0	Transform size is 4x4.
V4L2_VP9_TX_MODE_ALLOW_8X8	1	Transform size can be up to 8x8.
V4L2_VP9_TX_MODE_ALLOW_16X16	2	Transform size can be up to 16x16.
V4L2_VP9_TX_MODE_ALLOW_32X32	3	transform size can be up to 32x32.
V4L2_VP9_TX_MODE_SELECT	4	Bitstream contains the transform size for each block.

See section ‘7.3.1 Tx mode semantics’ of the *VP9* specification for more details.

V4L2_CID_STATELESS_VP9_FRAME (struct) Specifies the frame parameters for the associated VP9 frame decode request. This includes the necessary parameters for configuring a stateless hardware decoding pipeline for VP9. The bitstream parameters are defined according to *VP9*.

v4l2_ctrl_vp9_frame

Table 38: struct v4l2_ctrl_vp9_frame

struct <i>v4l2_vp9_loop_filter</i>	lf	Loop filter parameters. See struct <i>v4l2_vp9_loop_filter</i> for more details.
struct <i>v4l2_vp9_quantization</i>	quant	Quantization parameters. See <i>v4l2_vp9_quantization</i> for more details.

Continued on next page

Table 38 - continued from previous page

struct <i>v4l2_vp9_segmentation</i>	seg	Segmentation parameters. See <i>v4l2_vp9_segmentation</i> for more details.
__u32	flags	Combination of V4L2_VP9_FRAME_FLAG_* flags. See <i>Frame Flags</i> .
__u16	compressed_header_size	Compressed header size in bytes.
__u16	uncompressed_header_size	Uncompressed header size in bytes.
__u16	frame_width_minus_1	Add 1 to get the frame width expressed in pixels. See section 7.2.3 in <i>VP9</i> .
__u16	frame_height_minus_1	Add 1 to get the frame height expressed in pixels. See section 7.2.3 in <i>VP9</i> .
__u16	render_width_minus_1	Add 1 to get the expected render width expressed in pixels. This is not used during the decoding process but might be used by HW scalers to prepare a frame that's ready for scanout. See section 7.2.4 in <i>VP9</i> .
__u16	render_height_minus_1	Add 1 to get the expected render height expressed in pixels. This is not used during the decoding process but might be used by HW scalers to prepare a frame that's ready for scanout. See section 7.2.4 in <i>VP9</i> .
__u64	last_frame_ts	"last" reference buffer timestamp. The timestamp refers to the timestamp field in struct <i>v4l2_buffer</i> . Use the <i>v4l2_timeval_to_ns()</i> function to convert the struct <i>timeval</i> in struct <i>v4l2_buffer</i> to a <i>_u64</i> .
__u64	golden_frame_ts	"golden" reference buffer timestamp. The timestamp refers to the timestamp field in struct <i>v4l2_buffer</i> . Use the <i>v4l2_timeval_to_ns()</i> function to convert the struct <i>timeval</i> in struct <i>v4l2_buffer</i> to a <i>_u64</i> .
__u64	alt_frame_ts	"alt" reference buffer timestamp. The timestamp refers to the timestamp field in struct <i>v4l2_buffer</i> . Use the <i>v4l2_timeval_to_ns()</i> function to convert the struct <i>timeval</i> in struct <i>v4l2_buffer</i> to a <i>_u64</i> .
__u8	ref_frame_sign_bias	a bitfield specifying whether the sign bias is set for a given reference frame. See <i>Reference Frame Sign Bias</i> for more details.
__u8	reset_frame_context	specifies whether the frame context should be reset to default values. See <i>Reset Frame Context</i> for more details.
__u8	frame_context_idx	Frame context that should be used/updated.
__u8	profile	VP9 profile. Can be 0, 1, 2 or 3.
__u8	bit_depth	Component depth in bits. Can be 8, 10 or 12. Note that not all profiles support 10 and/or 12 bits depths.

Continued on next page

Table 38 - continued from previous page

__u8	interpolation_filter	Specifies the filter selection used for performing inter prediction. See <i>Interpolation Filter</i> for more details.
__u8	tile_cols_log2	Specifies the base 2 logarithm of the width of each tile (where the width is measured in units of 8x8 blocks). Shall be less than or equal to 6.
__u8	tile_rows_log2	Specifies the base 2 logarithm of the height of each tile (where the height is measured in units of 8x8 blocks).
__u8	reference_mode	Specifies the type of inter prediction to be used. See <i>Reference Mode</i> for more details.
__u8	reserved[7]	Applications and drivers must set this to zero.

Frame Flags

V4L2_VP9_FRAME_FLAG_KEY_FRAME	0x001	The frame is a key frame.
V4L2_VP9_FRAME_FLAG_SHOW_FRAME	0x002	The frame should be displayed.
V4L2_VP9_FRAME_FLAG_ERROR_RESILIENT	0x004	The decoding should be error resilient.
V4L2_VP9_FRAME_FLAG_INTRA_ONLY	0x008	The frame does not reference other frames.
V4L2_VP9_FRAME_FLAG_ALLOW_HIGH_PREC_MV	0x010	The frame can use high precision motion vectors.
V4L2_VP9_FRAME_FLAG_REFRESH_FRAME_CTX	0x020	Frame context should be updated after decoding.
V4L2_VP9_FRAME_FLAG_PARALLEL_DEC_MODE	0x040	Parallel decoding is used.
V4L2_VP9_FRAME_FLAG_X_SUBSAMPLING	0x080	Vertical subsampling is enabled.
V4L2_VP9_FRAME_FLAG_Y_SUBSAMPLING	0x100	Horizontal subsampling is enabled.
V4L2_VP9_FRAME_FLAG_COLOR_RANGE_FULL_SWING	0x200	The full UV range is used.

Reference Frame Sign Bias

V4L2_VP9_SIGN_BIAS_LAST	0x1	Sign bias is set for the last reference frame.
V4L2_VP9_SIGN_BIAS_GOLDEN	0x2	Sign bias is set for the golden reference frame.
V4L2_VP9_SIGN_BIAS_ALT	0x2	Sign bias is set for the alt reference frame.

Reset Frame Context

V4L2_VP9_RESET_FRAME_CTX_NONE	0	Do not reset any frame context.
V4L2_VP9_RESET_FRAME_CTX_SPEC	1	Reset the frame context pointed to by <i>v4l2_ctrl_vp9_frame.frame_context_idx</i> .
V4L2_VP9_RESET_FRAME_CTX_ALL	2	Reset all frame contexts.

See section ‘7.2 Uncompressed header semantics’ of the *VP9* specification for more details.

Interpolation Filter

V4L2_VP9_INTERP_FILTER_EIGHTTAP	0	Eight tap filter.
V4L2_VP9_INTERP_FILTER_EIGHTTAP_SMOOTH	1	Eight tap smooth filter.
V4L2_VP9_INTERP_FILTER_EIGHTTAP_SHARP	2	Eight tap sharp filter.
V4L2_VP9_INTERP_FILTER_BILINEAR	3	Bilinear filter.
V4L2_VP9_INTERP_FILTER_SWITCHABLE	4	Filter selection is signaled at the block level.

See section ‘7.2.7 Interpolation filter semantics’ of the *VP9* specification for more details.

Reference Mode

V4L2_VP9_REFERENCE_MODE_SINGLE_REFERENCE	0	Indicates that all the inter blocks use only a single reference frame to generate motion compensated prediction.
V4L2_VP9_REFERENCE_MODE_COMPOUND_REFERENCE	1	Requires all the inter blocks to use compound mode. Single reference frame prediction is not allowed.
V4L2_VP9_REFERENCE_MODE_SELECT	2	Allows each individual inter block to select between single and compound prediction modes.

See section ‘7.3.6 Frame reference mode semantics’ of the *VP9* specification for more details.

v4l2_vp9_segmentation

Encodes the quantization parameters. See section ‘7.2.10 Segmentation params syntax’ of the *VP9* specification for more details.

Table 39: struct v4l2_vp9_segmentation

__u8	feature_data[8][4]	Data attached to each feature. Data entry is only valid if the feature is enabled. The array shall be indexed with segment number as the first dimension (0..7) and one of V4L2_VP9_SEG_* as the second dimension. See <i>Segment Feature IDs</i> .
__u8	feature_enabled[8]	Bitmask defining which features are enabled in each segment. The value for each segment is a combination of V4L2_VP9_SEGMENT_FEATURE_ENABLED(id) values where id is one of V4L2_VP9_SEG_*. See <i>Segment Feature IDs</i> .
__u8	tree_probs[7]	Specifies the probability values to be used when decoding a Segment-ID. See ‘5.15. Segmentation map’ section of <i>VP9</i> for more details.
__u8	pred_probs[3]	Specifies the probability values to be used when decoding a Predicted-Segment-ID. See ‘6.4.14. Get segment id syntax’ section of <i>VP9</i> for more details.
__u8	flags	Combination of V4L2_VP9_SEGMENTATION_FLAG_* flags. See <i>Segmentation Flags</i> .
__u8	reserved[5]	Applications and drivers must set this to zero.

Segment feature IDs

V4L2_VP9_SEG_LVL_ALT_Q	0	Quantizer segment feature.
V4L2_VP9_SEG_LVL_ALT_L	1	Loop filter segment feature.
V4L2_VP9_SEG_LVL_REF_FRAME	2	Reference frame segment feature.
V4L2_VP9_SEG_LVL_SKIP	3	Skip segment feature.
V4L2_VP9_SEG_LVL_MAX	4	Number of segment features.

Segmentation Flags

V4L2_VP9_SEGMENTATION_FLAG_ENABLED	0x01	Indicates that this frame makes use of the segmentation tool.
V4L2_VP9_SEGMENTATION_FLAG_UPDATE_MAP	0x02	Indicates that the segmentation map should be updated during the decoding of this frame.
V4L2_VP9_SEGMENTATION_FLAG_TEMPORAL_UPDATE	0x04	Indicates that the updates to the segmentation map are coded relative to the existing segmentation map.
V4L2_VP9_SEGMENTATION_FLAG_UPDATE_DATA	0x08	Indicates that new parameters are about to be specified for each segment.
V4L2_VP9_SEGMENTATION_FLAG_ABS_OR_DELTA_UPDATE	0x10	Indicates that the segmentation parameters represent the actual values to be used.

v4l2_vp9_quantization

Encodes the quantization parameters. See section ‘7.2.9 Quantization params syntax’ of the VP9 specification for more details.

Table 40: struct v4l2_vp9_quantization

__u8	base_q_idx	Indicates the base frame qindex.
__s8	delta_q_y_dc	Indicates the Y DC quantizer relative to base_q_idx.
__s8	delta_q_uv_dc	Indicates the UV DC quantizer relative to base_q_idx.
__s8	delta_q_uv_ac	Indicates the UV AC quantizer relative to base_q_idx.
__u8	reserved[4]	Applications and drivers must set this to zero.

v4l2_vp9_loop_filter

This structure contains all loop filter related parameters. See sections ‘7.2.8 Loop filter semantics’ of the *VP9* specification for more details.

Table 41: struct v4l2_vp9_loop_filter

__s8	ref_deltas[4]	Contains the adjustment needed for the filter level based on the chosen reference frame.
------	---------------	--

Continued on next page

Table 41 - continued from previous page

__s8	mode_deltas[2]	Contains the adjustment needed for the filter level based on the chosen mode.
__u8	level	Indicates the loop filter strength.
__u8	sharpness	Indicates the sharpness level.
__u8	flags	Combination of V4L2_VP9_LOOP_FILTER_FLAG_* flags. See Loop Filter Flags .
__u8	reserved[7]	Applications and drivers must set this to zero.

Loop Filter Flags

V4L2_VP9_LOOP_FILTER_FLAG_DELTA_ENABLED	0x1	When set, the filter level depends on the mode and reference frame used to predict a block.
V4L2_VP9_LOOP_FILTER_FLAG_DELTA_UPDATE	0x2	When set, the bitstream contains additional syntax elements that specify which mode and reference frame deltas are to be updated.

3.2.1.17 JPEG Control Reference

The JPEG class includes controls for common features of JPEG encoders and decoders. Currently it includes features for codecs implementing progressive baseline DCT compression process with Huffman entropy coding.

JPEG Control IDs

V4L2_CID_JPEG_CLASS (class) The JPEG class descriptor. Calling *ioctl*s `VIDIOC_QUERYCTRL`, `VIDIOC_QUERY_EXT_CTRL` and `VIDIOC_QUERYMENU` for this control will return a description of this control class.

V4L2_CID_JPEG_CHROMA_SUBSAMPLING (menu) The chroma subsampling factors describe how each component of an input image is sampled, in respect to maximum sample rate in each spatial dimension. See *ITU-T.81*, clause A.1.1. for more details. The `V4L2_CID_JPEG_CHROMA_SUBSAMPLING` control determines how Cb and Cr components are downsampled after converting an input image from RGB to Y'CbCr color space.

V4L2_JPEG_CHROMA_SUBSAMPLING_444	No chroma subsampling, each pixel has Y, Cr and Cb values.
V4L2_JPEG_CHROMA_SUBSAMPLING_422	Horizontally subsample Cr, Cb components by a factor of 2.
V4L2_JPEG_CHROMA_SUBSAMPLING_420	Subsample Cr, Cb components horizontally and vertically by 2.
V4L2_JPEG_CHROMA_SUBSAMPLING_411	Horizontally subsample Cr, Cb components by a factor of 4.
V4L2_JPEG_CHROMA_SUBSAMPLING_410	Subsample Cr, Cb components horizontally by 4 and vertically by 2.
V4L2_JPEG_CHROMA_SUBSAMPLING_GRAY	Use only luminance component.

V4L2_CID_JPEG_RESTART_INTERVAL (integer) The restart interval determines an interval of inserting RSTm markers ($m = 0..7$). The purpose of these markers is to additionally reinitialize the encoder process, in order to process blocks of an image independently. For the lossy compression processes the restart interval unit is MCU (Minimum Coded Unit) and its value is contained in DRI (Define Restart Interval) marker. If V4L2_CID_JPEG_RESTART_INTERVAL control is set to 0, DRI and RSTm markers will not be inserted.

V4L2_CID_JPEG_COMPRESSION_QUALITY (integer) Determines trade-off between image quality and size. It provides simpler method for applications to control image quality, without a need for direct reconfiguration of luminance and chrominance quantization tables. In cases where a driver uses quantization tables configured directly by an application, using interfaces defined elsewhere, V4L2_CID_JPEG_COMPRESSION_QUALITY control should be set by driver to 0.

The value range of this control is driver-specific. Only positive, non-zero values are meaningful. The recommended range is 1 - 100, where larger values correspond to better image quality.

V4L2_CID_JPEG_ACTIVE_MARKER (bitmask) Specify which JPEG markers are included in compressed stream. This control is valid only for encoders.

V4L2_JPEG_ACTIVE_MARKER_APP0	Application data segment APP ₀ .
V4L2_JPEG_ACTIVE_MARKER_APP1	Application data segment APP ₁ .
V4L2_JPEG_ACTIVE_MARKER_COM	Comment segment.
V4L2_JPEG_ACTIVE_MARKER_DQT	Quantization tables segment.
V4L2_JPEG_ACTIVE_MARKER_DHT	Huffman tables segment.

For more details about JPEG specification, refer to *ITU-T.81*, *JFIF*, *W3C JPEG JFIF*.

3.2.1.18 Digital Video Control Reference

The Digital Video control class is intended to control receivers and transmitters for *VGA*, *DVI* (Digital Visual Interface), *HDMI* (*HDMI*) and DisplayPort (*DP*). These controls are generally expected to be private to the receiver or transmitter subdevice that implements them, so they are only exposed on the `/dev/v4l-subdev*` device node.

Note: Note that these devices can have multiple input or output pads which are hooked up to e.g. HDMI connectors. Even though the subdevice will receive or transmit video from/to only one of those pads, the other pads can still be active when it comes to EDID (Extended Display Identification Data, *EDID*) and HDCP (High-bandwidth Digital Content Protection System, *HDCP*) processing, allowing the device to do the fairly slow EDID/HDCP handling in advance. This allows for quick switching between connectors.

These pads appear in several of the controls in this section as bitmasks, one bit for each pad. Bit 0 corresponds to pad 0, bit 1 to pad 1, etc. The maximum value of the control is the set of valid pads.

Digital Video Control IDs

V4L2_CID_DV_CLASS (class) The Digital Video class descriptor.

V4L2_CID_DV_TX_HOTPLUG (bitmask) Many connectors have a hotplug pin which is high if EDID information is available from the source. This control shows the state of the hotplug pin as seen by the transmitter. Each bit corresponds to an output pad on the transmitter. If an output pad does not have an associated hotplug pin, then the bit for that pad will be 0. This read-only control is applicable to DVI-D, HDMI and DisplayPort connectors.

V4L2_CID_DV_TX_RXSENSE (bitmask) Rx Sense is the detection of pull-ups on the TMDS clock lines. This normally means that the sink has left/entered standby (i.e. the transmitter can sense that the receiver is ready to receive video). Each bit corresponds to an output pad on the transmitter. If an output pad does not have an associated Rx Sense, then the bit for that pad will be 0. This read-only control is applicable to DVI-D and HDMI devices.

V4L2_CID_DV_TX_EDID_PRESENT (bitmask) When the transmitter sees the hotplug signal from the receiver it will attempt to read the EDID. If set, then the transmitter has read at least the first block (= 128 bytes). Each bit corresponds to an output pad on the transmitter. If an output pad does not support EDIDs, then the bit for that pad will be 0. This read-only control is applicable to VGA, DVI-A/D, HDMI and DisplayPort connectors.

V4L2_CID_DV_TX_MODE (enum)

enum v4l2_dv_tx_mode - HDMI transmitters can transmit in DVI-D mode (just video) or in HDMI mode (video + audio + auxiliary data). This control selects which mode to use: `V4L2_DV_TX_MODE_DVI_D` or `V4L2_DV_TX_MODE_HDMI`. This control is applicable to HDMI connectors.

V4L2_CID_DV_TX_RGB_RANGE (enum)

enum v4l2_dv_rgb_range - Select the quantization range for RGB output. `V4L2_DV_RANGE_AUTO` follows the RGB quantization range specified in the standard for the video interface (ie. *CEA-861-E* for HDMI). `V4L2_DV_RANGE_LIMITED` and `V4L2_DV_RANGE_FULL` override the standard to be compatible with sinks that have not implemented the standard correctly (unfortunately quite common for HDMI and DVI-D). Full range allows all possible values to be used whereas limited range sets the range to $(16 \ll (N-8)) - (235 \ll (N-8))$ where N is the number of bits per component. This control is applicable to VGA, DVI-A/D, HDMI and DisplayPort connectors.

V4L2_CID_DV_TX_IT_CONTENT_TYPE (enum)

enum v4l2_dv_it_content_type - Configures the IT Content Type of the transmitted video. This information is sent over HDMI and DisplayPort connectors as part of the AVI InfoFrame. The term 'IT Content' is used for content that originates from a computer as opposed to content from a TV broadcast or an analog source. The enum `v4l2_dv_it_content_type` defines the possible content types:

V4L2_DV_IT_CONTENT_TYPE_GRAPHICS	Graphics content. Pixel data should be passed unfiltered and without analog reconstruction.
V4L2_DV_IT_CONTENT_TYPE_PHOTO	Photo content. The content is derived from digital still pictures. The content should be passed through with minimal scaling and picture enhancements.
V4L2_DV_IT_CONTENT_TYPE_CINEMA	Cinema content.
V4L2_DV_IT_CONTENT_TYPE_GAME	Game content. Audio and video latency should be minimized.
V4L2_DV_IT_CONTENT_TYPE_NO_ITC	No IT Content information is available and the ITC bit in the AVI InfoFrame is set to 0.

V4L2_CID_DV_RX_POWER_PRESENT (bitmask) Detects whether the receiver receives power from the source (e.g. HDMI carries 5V on one of the pins). This is often used to power an eeprom which contains EDID information, such that the source can read the EDID even if the sink is in standby/power off. Each bit corresponds to an input pad on the receiver. If an input pad cannot detect whether power is present, then the bit for that pad will be 0. This read-only control is applicable to DVI-D, HDMI and DisplayPort connectors.

V4L2_CID_DV_RX_RGB_RANGE (enum)

enum v4l2_dv_rgb_range - Select the quantization range for RGB input. V4L2_DV_RANGE_AUTO follows the RGB quantization range specified in the standard for the video interface (ie. *CEA-861-E* for HDMI). V4L2_DV_RANGE_LIMITED and V4L2_DV_RANGE_FULL override the standard to be compatible with sources that have not implemented the standard correctly (unfortunately quite common for HDMI and DVI-D). Full range allows all possible values to be used whereas limited range sets the range to $(16 \ll (N-8)) - (235 \ll (N-8))$ where N is the number of bits per component. This control is applicable to VGA, DVI-A/D, HDMI and DisplayPort connectors.

V4L2_CID_DV_RX_IT_CONTENT_TYPE (enum)

enum v4l2_dv_it_content_type - Reads the IT Content Type of the received video. This information is sent over HDMI and DisplayPort connectors as part of the AVI InfoFrame. The term 'IT Content' is used for content that originates from a computer as opposed to content from a TV broadcast or an analog source. See V4L2_CID_DV_TX_IT_CONTENT_TYPE for the available content types.

3.2.1.19 RF Tuner Control Reference

The RF Tuner (RF_TUNER) class includes controls for common features of devices having RF tuner.

In this context, RF tuner is radio receiver circuit between antenna and demodulator. It receives radio frequency (RF) from the antenna and converts that received signal to lower intermediate frequency (IF) or baseband frequency (BB). Tuners that could do baseband output are often called Zero-IF tuners. Older tuners were typically simple PLL tuners inside a metal box, while newer ones are highly integrated chips without a metal box "silicon tuners". These controls are mostly applicable for new feature rich silicon tuners, just because older tuners does not have much adjustable features.

For more information about RF tuners see [Tuner \(radio\)](#) and [RF front end](#) from Wikipedia.

RF_TUNER Control IDs

- V4L2_CID_RF_TUNER_CLASS (class)** The RF_TUNER class descriptor. Calling *ioctl*s *VIDIOC_QUERYCTRL*, *VIDIOC_QUERY_EXT_CTRL* and *VIDIOC_QUERYMENU* for this control will return a description of this control class.
- V4L2_CID_RF_TUNER_BANDWIDTH_AUTO (boolean)** Enables/disables tuner radio channel bandwidth configuration. In automatic mode bandwidth configuration is performed by the driver.
- V4L2_CID_RF_TUNER_BANDWIDTH (integer)** Filter(s) on tuner signal path are used to filter signal according to receiving party needs. Driver configures filters to fulfill desired bandwidth requirement. Used when *V4L2_CID_RF_TUNER_BANDWIDTH_AUTO* is not set. Unit is in Hz. The range and step are driver-specific.
- V4L2_CID_RF_TUNER_LNA_GAIN_AUTO (boolean)** Enables/disables LNA automatic gain control (AGC)
- V4L2_CID_RF_TUNER_MIXER_GAIN_AUTO (boolean)** Enables/disables mixer automatic gain control (AGC)
- V4L2_CID_RF_TUNER_IF_GAIN_AUTO (boolean)** Enables/disables IF automatic gain control (AGC)
- V4L2_CID_RF_TUNER_RF_GAIN (integer)** The RF amplifier is the very first amplifier on the receiver signal path, just right after the antenna input. The difference between the LNA gain and the RF gain in this document is that the LNA gain is integrated in the tuner chip while the RF gain is a separate chip. There may be both RF and LNA gain controls in the same device. The range and step are driver-specific.
- V4L2_CID_RF_TUNER_LNA_GAIN (integer)** LNA (low noise amplifier) gain is first gain stage on the RF tuner signal path. It is located very close to tuner antenna input. Used when *V4L2_CID_RF_TUNER_LNA_GAIN_AUTO* is not set. See *V4L2_CID_RF_TUNER_RF_GAIN* to understand how RF gain and LNA gain differs from the each others. The range and step are driver-specific.
- V4L2_CID_RF_TUNER_MIXER_GAIN (integer)** Mixer gain is second gain stage on the RF tuner signal path. It is located inside mixer block, where RF signal is down-converted by the mixer. Used when *V4L2_CID_RF_TUNER_MIXER_GAIN_AUTO* is not set. The range and step are driver-specific.
- V4L2_CID_RF_TUNER_IF_GAIN (integer)** IF gain is last gain stage on the RF tuner signal path. It is located on output of RF tuner. It controls signal level of intermediate frequency output or baseband output. Used when *V4L2_CID_RF_TUNER_IF_GAIN_AUTO* is not set. The range and step are driver-specific.
- V4L2_CID_RF_TUNER_PLL_LOCK (boolean)** Is synthesizer PLL locked? RF tuner is receiving given frequency when that control is set. This is a read-only control.

3.2.1.20 FM Transmitter Control Reference

The FM Transmitter (FM_TX) class includes controls for common features of FM transmissions capable devices. Currently this class includes parameters for audio compression, pilot tone generation, audio deviation limiter, RDS transmission and tuning power features.

FM_TX Control IDs

V4L2_CID_FM_TX_CLASS (class) The FM_TX class descriptor. Calling *ioctl*s *VIDIOC_QUERYCTRL*, *VIDIOC_QUERY_EXT_CTRL* and *VIDIOC_QUERYMENU* for this control will return a description of this control class.

V4L2_CID_RDS_TX_DEVIATION (integer) Configures RDS signal frequency deviation level in Hz. The range and step are driver-specific.

V4L2_CID_RDS_TX_PI (integer) Sets the RDS Programme Identification field for transmission.

V4L2_CID_RDS_TX_PTY (integer) Sets the RDS Programme Type field for transmission. This encodes up to 31 pre-defined programme types.

V4L2_CID_RDS_TX_PS_NAME (string) Sets the Programme Service name (PS_NAME) for transmission. It is intended for static display on a receiver. It is the primary aid to listeners in programme service identification and selection. In Annex E of *IEC 62106*, the RDS specification, there is a full description of the correct character encoding for Programme Service name strings. Also from RDS specification, PS is usually a single eight character text. However, it is also possible to find receivers which can scroll strings sized as 8 x N characters. So, this control must be configured with steps of 8 characters. The result is it must always contain a string with size multiple of 8.

V4L2_CID_RDS_TX_RADIO_TEXT (string) Sets the Radio Text info for transmission. It is a textual description of what is being broadcasted. RDS Radio Text can be applied when broadcaster wishes to transmit longer PS names, programme-related information or any other text. In these cases, RadioText should be used in addition to *V4L2_CID_RDS_TX_PS_NAME*. The encoding for Radio Text strings is also fully described in Annex E of *IEC 62106*. The length of Radio Text strings depends on which RDS Block is being used to transmit it, either 32 (2A block) or 64 (2B block). However, it is also possible to find receivers which can scroll strings sized as 32 x N or 64 x N characters. So, this control must be configured with steps of 32 or 64 characters. The result is it must always contain a string with size multiple of 32 or 64.

V4L2_CID_RDS_TX_MONO_STEREO (boolean) Sets the Mono/Stereo bit of the Decoder Identification code. If set, then the audio was recorded as stereo.

V4L2_CID_RDS_TX_ARTIFICIAL_HEAD (boolean) Sets the *Artificial Head* bit of the Decoder Identification code. If set, then the audio was recorded using an artificial head.

V4L2_CID_RDS_TX_COMPRESSED (boolean) Sets the Compressed bit of the Decoder Identification code. If set, then the audio is compressed.

V4L2_CID_RDS_TX_DYNAMIC_PTY (boolean) Sets the Dynamic PTY bit of the Decoder Identification code. If set, then the PTY code is dynamically switched.

V4L2_CID_RDS_TX_TRAFFIC_ANNOUNCEMENT (boolean) If set, then a traffic announcement is in progress.

V4L2_CID_RDS_TX_TRAFFIC_PROGRAM (boolean) If set, then the tuned programme carries traffic announcements.

V4L2_CID_RDS_TX_MUSIC_SPEECH (boolean) If set, then this channel broadcasts music. If cleared, then it broadcasts speech. If the transmitter doesn't make this distinction, then it should be set.

V4L2_CID_RDS_TX_ALT_FREQS_ENABLE (boolean) If set, then transmit alternate frequencies.

V4L2_CID_RDS_TX_ALT_FREQS (__u32 array) The alternate frequencies in kHz units. The RDS standard allows for up to 25 frequencies to be defined. Drivers may support fewer frequencies so check the array size.

V4L2_CID_AUDIO_LIMITER_ENABLED (boolean) Enables or disables the audio deviation limiter feature. The limiter is useful when trying to maximize the audio volume, minimize receiver-generated distortion and prevent overmodulation.

V4L2_CID_AUDIO_LIMITER_RELEASE_TIME (integer) Sets the audio deviation limiter feature release time. Unit is in useconds. Step and range are driver-specific.

V4L2_CID_AUDIO_LIMITER_DEVIATION (integer) Configures audio frequency deviation level in Hz. The range and step are driver-specific.

V4L2_CID_AUDIO_COMPRESSION_ENABLED (boolean) Enables or disables the audio compression feature. This feature amplifies signals below the threshold by a fixed gain and compresses audio signals above the threshold by the ratio of Threshold/(Gain + Threshold).

V4L2_CID_AUDIO_COMPRESSION_GAIN (integer) Sets the gain for audio compression feature. It is a dB value. The range and step are driver-specific.

V4L2_CID_AUDIO_COMPRESSION_THRESHOLD (integer) Sets the threshold level for audio compression feature. It is a dB value. The range and step are driver-specific.

V4L2_CID_AUDIO_COMPRESSION_ATTACK_TIME (integer) Sets the attack time for audio compression feature. It is a useconds value. The range and step are driver-specific.

V4L2_CID_AUDIO_COMPRESSION_RELEASE_TIME (integer) Sets the release time for audio compression feature. It is a useconds value. The range and step are driver-specific.

V4L2_CID_PILOT_TONE_ENABLED (boolean) Enables or disables the pilot tone generation feature.

V4L2_CID_PILOT_TONE_DEVIATION (integer) Configures pilot tone frequency deviation level. Unit is in Hz. The range and step are driver-specific.

V4L2_CID_PILOT_TONE_FREQUENCY (integer) Configures pilot tone frequency value. Unit is in Hz. The range and step are driver-specific.

V4L2_CID_TUNE_PREEMPHASIS (enum)

enum v4l2_preemphasis - Configures the pre-emphasis value for broadcasting. A pre-emphasis filter is applied to the broadcast to accentuate the high audio frequencies. Depending on the region, a time constant of either 50 or 75 useconds is used. The enum v4l2_preemphasis defines possible values for pre-emphasis. Here they are:

V4L2_PREEMPHASIS_DISABLED	No pre-emphasis is applied.
V4L2_PREEMPHASIS_50_uS	A pre-emphasis of 50 uS is used.
V4L2_PREEMPHASIS_75_uS	A pre-emphasis of 75 uS is used.

V4L2_CID_TUNE_POWER_LEVEL (integer) Sets the output power level for signal transmission. Unit is in dBuV. Range and step are driver-specific.

V4L2_CID_TUNE_ANTENNA_CAPACITOR (integer) This selects the value of antenna tuning capacitor manually or automatically if set to zero. Unit, range and step are driver-specific.

For more details about RDS specification, refer to *IEC 62106* document, from CENELEC.

3.2.1.21 FM Receiver Control Reference

The FM Receiver (FM_RX) class includes controls for common features of FM Reception capable devices.

FM_RX Control IDs

V4L2_CID_FM_RX_CLASS (class) The FM_RX class descriptor. Calling *ioctl*s *VIDIOC_QUERYCTRL*, *VIDIOC_QUERY_EXT_CTRL* and *VIDIOC_QUERYMENU* for this control will return a description of this control class.

V4L2_CID_RDS_RECEPTION (boolean) Enables/disables RDS reception by the radio tuner

V4L2_CID_RDS_RX_PTY (integer) Gets RDS Programme Type field. This encodes up to 31 pre-defined programme types.

V4L2_CID_RDS_RX_PS_NAME (string) Gets the Programme Service name (PS_NAME). It is intended for static display on a receiver. It is the primary aid to listeners in programme service identification and selection. In Annex E of *IEC 62106*, the RDS specification, there is a full description of the correct character encoding for Programme Service name strings. Also from RDS specification, PS is usually a single eight character text. However, it is also possible to find receivers which can scroll strings sized as 8 x N characters. So, this control must be configured with steps of 8 characters. The result is it must always contain a string with size multiple of 8.

V4L2_CID_RDS_RX_RADIO_TEXT (string) Gets the Radio Text info. It is a textual description of what is being broadcasted. RDS Radio Text can be applied when broadcaster wishes to transmit longer PS names, programme-related information or any other text. In these cases, RadioText can be used in addition to *V4L2_CID_RDS_RX_PS_NAME*. The encoding for Radio Text strings is also fully described in Annex E of *IEC 62106*. The length of Radio Text strings depends on which RDS Block is being used to transmit it, either 32 (2A block) or 64 (2B block). However, it is also possible to find receivers which can scroll strings sized as 32 x N or 64 x N characters. So, this control must be configured with steps of 32 or 64 characters. The result is it must always contain a string with size multiple of 32 or 64.

V4L2_CID_RDS_RX_TRAFFIC_ANNOUNCEMENT (boolean) If set, then a traffic announcement is in progress.

V4L2_CID_RDS_RX_TRAFFIC_PROGRAM (boolean) If set, then the tuned programme carries traffic announcements.

V4L2_CID_RDS_RX_MUSIC_SPEECH (boolean) If set, then this channel broadcasts music. If cleared, then it broadcasts speech. If the transmitter doesn't make this distinction, then it will be set.

V4L2_CID_TUNE_DEEMPHASIS (enum)

enum v4l2_deemphasis - Configures the de-emphasis value for reception. A de-emphasis filter is applied to the broadcast to accentuate the high audio frequencies. Depending on the region, a time constant of either 50 or 75 useconds is used. The enum v4l2_deemphasis defines possible values for de-emphasis. Here they are:

V4L2_DEEMPHASIS_DISABLED	No de-emphasis is applied.
V4L2_DEEMPHASIS_50_uS	A de-emphasis of 50 uS is used.
V4L2_DEEMPHASIS_75_uS	A de-emphasis of 75 uS is used.

3.2.1.22 Detect Control Reference

The Detect class includes controls for common features of various motion or object detection capable devices.

Detect Control IDs

V4L2_CID_DETECT_CLASS (class) The Detect class descriptor. Calling *ioctl VIDIOC_QUERYCTRL*, *VIDIOC_QUERY_EXT_CTRL* and *VIDIOC_QUERYMENU* for this control will return a description of this control class.

V4L2_CID_DETECT_MD_MODE (menu) Sets the motion detection mode.

V4L2_DETECT_MD_MODE_DISABLED	Disable motion detection.
V4L2_DETECT_MD_MODE_GLOBAL	Use a single motion detection threshold.
V4L2_DETECT_MD_MODE_THRESHOLD_GRID	The image is divided into a grid, each cell with its own motion detection threshold. These thresholds are set through the V4L2_CID_DETECT_MD_THRESHOLD_GRID matrix control.
V4L2_DETECT_MD_MODE_REGION_GRID	The image is divided into a grid, each cell with its own region value that specifies which per-region motion detection thresholds should be used. Each region has its own thresholds. How these per-region thresholds are set up is driver-specific. The region values for the grid are set through the V4L2_CID_DETECT_MD_REGION_GRID matrix control.

V4L2_CID_DETECT_MD_GLOBAL_THRESHOLD (integer) Sets the global motion detection threshold to be used with the V4L2_DETECT_MD_MODE_GLOBAL motion detection mode.

V4L2_CID_DETECT_MD_THRESHOLD_GRID (__u16 matrix) Sets the motion detection thresholds for each cell in the grid. To be used with the V4L2_DETECT_MD_MODE_THRESHOLD_GRID motion detection mode. Matrix element (0, 0) represents the cell at the top-left of the grid.

V4L2_CID_DETECT_MD_REGION_GRID (__u8 matrix) Sets the motion detection region value for each cell in the grid. To be used with the V4L2_DETECT_MD_MODE_REGION_GRID motion detection mode. Matrix element (0, 0) represents the cell at the top-left of the grid.

3.2.1.23 Colorimetry Control Reference

The Colorimetry class includes controls for High Dynamic Range imaging for representing colors in digital images and video. The controls should be used for video and image encoding and decoding as well as in HDMI receivers and transmitters.

Colorimetry Control IDs

V4L2_CID_COLORIMETRY_CLASS (class) The Colorimetry class descriptor. Calling *ioctl*s *VIDIOC_QUERYCTRL*, *VIDIOC_QUERY_EXT_CTRL* and *VIDIOC_QUERYMENU* for this control will return a description of this control class.

V4L2_CID_COLORIMETRY_HDR10_CLL_INFO (struct) The Content Light Level defines upper bounds for the nominal target brightness light level of the pictures.

`v4l2_ctrl_hdr10_cll_info`

Table 42:

<code>__u16</code>	<code>max_content_light_level</code>	The upper bound for the maximum light level among all in
<code>__u16</code>	<code>max_pic_average_light_level</code>	The upper bound for the maximum average light level am

V4L2_CID_COLORIMETRY_HDR10_MASTERING_DISPLAY (struct) The mastering display defines the color volume (the color primaries, white point and luminance range) of a display considered to be the mastering display for the current video content.

`v4l2_ctrl_hdr10_mastering_display`

<code>__u16</code>	<code>display_primaries_x[3]</code>	Specifies the normalized x chromaticity coordinate of the color
<code>__u16</code>	<code>display_primaries_y[3]</code>	Specifies the normalized y chromaticity coordinate of the color
<code>__u16</code>	<code>white_point_x</code>	Specifies the normalized x chromaticity coordinate of the white
<code>__u16</code>	<code>white_point_y</code>	Specifies the normalized y chromaticity coordinate of the white
<code>__u32</code>	<code>max_luminance</code>	Specifies the nominal maximum display luminance of the maste
<code>__u32</code>	<code>min_luminance</code>	specifies the nominal minimum display luminance of the master

3.2.1.24 Guidelines for Video4Linux pixel format 4CCs

Guidelines for Video4Linux 4CC codes defined using `v4l2_fourcc()` are specified in this document. First of the characters defines the nature of the pixel format, compression and colour space. The interpretation of the other three characters depends on the first one.

Existing 4CCs may not obey these guidelines.

Raw bayer

The following first characters are used by raw bayer formats:

- B: raw bayer, uncompressed
- b: raw bayer, DPCM compressed
- a: A-law compressed
- u: u-law compressed

2nd character: pixel order

- B: BGGR
- G: GBRG
- g: GRBG
- R: RGGG

3rd character: uncompressed bits-per-pixel 0-9, A-

4th character: compressed bits-per-pixel 0-9, A-

3.2.1.25 Data Formats

Data Format Negotiation

Different devices exchange different kinds of data with applications, for example video images, raw or sliced VBI data, RDS datagrams. Even within one kind many different formats are possible, in particular there is an abundance of image formats. Although drivers must provide a default and the selection persists across closing and reopening a device, applications should always negotiate a data format before engaging in data exchange. Negotiation means the application asks for a particular format and the driver selects and reports the best the hardware can do to satisfy the request. Of course applications can also just query the current selection.

A single mechanism exists to negotiate all data formats using the aggregate struct `v4l2_format` and the `VIDIOC_G_FMT` and `VIDIOC_S_FMT` ioctls. Additionally the `VIDIOC_TRY_FMT` ioctl can be used to examine what the hardware *could* do, without actually selecting a new data format. The data formats supported by the V4L2 API are covered in the respective device section in *Interfaces*. For a closer look at image formats see *Image Formats*.

The `VIDIOC_S_FMT` ioctl is a major turning-point in the initialization sequence. Prior to this point multiple panel applications can access the same device concurrently to select the current input, change controls or modify other properties. The first `VIDIOC_S_FMT` assigns a logical stream (video data, VBI data etc.) exclusively to one file descriptor.

Exclusive means no other application, more precisely no other file descriptor, can grab this stream or change device properties inconsistent with the negotiated parameters. A video standard change for example, when the new standard uses a different number of scan lines, can invalidate the selected image format. Therefore only the file descriptor owning the stream can make invalidating changes. Accordingly multiple file descriptors which grabbed different logical streams prevent each other from interfering with their settings. When for example video overlay is about to start or already in progress, simultaneous video capturing may be restricted to the same cropping and image size.

When applications omit the `VIDIOC_S_FMT` ioctl its locking side effects are implied by the next step, the selection of an I/O method with the `ioctl VIDIOC_REQBUFS` ioctl or implicit with the first `read()` or `write()` call.

Generally only one logical stream can be assigned to a file descriptor, the exception being drivers permitting simultaneous video capturing and overlay using the same file descriptor for compatibility with V4L and earlier versions of V4L2. Switching the logical stream or returning into “panel mode” is possible by closing and reopening the device. Drivers *may* support a switch using `VIDIOC_S_FMT`.

All drivers exchanging data with applications must support the `VIDIOC_G_FMT` and `VIDIOC_S_FMT` ioctl. Implementation of the `VIDIOC_TRY_FMT` is highly recommended but optional.

Image Format Enumeration

Apart of the generic format negotiation functions a special ioctl to enumerate all image formats supported by video capture, overlay or output devices is available.¹

The `ioctl VIDIOC_ENUM_FMT` ioctl must be supported by all drivers exchanging image data with applications.

Important: Drivers are not supposed to convert image formats in kernel space. They must enumerate only formats directly supported by the hardware. If necessary driver writers should publish an example conversion routine or library for integration into applications.

3.2.1.26 Single- and multi-planar APIs

Some devices require data for each input or output video frame to be placed in discontinuous memory buffers. In such cases, one video frame has to be addressed using more than one memory address, i.e. one pointer per “plane”. A plane is a sub-buffer of the current frame. For examples of such formats see *Image Formats*.

Initially, V4L2 API did not support multi-planar buffers and a set of extensions has been introduced to handle them. Those extensions constitute what is being referred to as the “multi-planar API”.

Some of the V4L2 API calls and structures are interpreted differently, depending on whether single- or multi-planar API is being used. An application can choose whether to use one or the other by passing a corresponding buffer type to its ioctl calls. Multi-planar versions of buffer types are suffixed with an `_MPLANE` string. For a list of available multi-planar buffer types see `enum v4l2_buf_type`.

¹ Enumerating formats an application has no a-priori knowledge of (otherwise it could explicitly ask for them and need not enumerate) seems useless, but there are applications serving as proxy between drivers and the actual video applications for which this is useful.

Multi-planar formats

Multi-planar API introduces new multi-planar formats. Those formats use a separate set of FourCC codes. It is important to distinguish between the multi-planar API and a multi-planar format. Multi-planar API calls can handle all single-planar formats as well (as long as they are passed in multi-planar API structures), while the single-planar API cannot handle multi-planar formats.

Calls that distinguish between single and multi-planar APIs

VIDIOC_QUERYCAP Two additional multi-planar capabilities are added. They can be set together with non-multi-planar ones for devices that handle both single- and multi-planar formats.

VIDIOC_G_FMT, VIDIOC_S_FMT, VIDIOC_TRY_FMT New structures for describing multi-planar formats are added: struct `v4l2_pix_format_plane` and struct `v4l2_plane_pix_format`. Drivers may define new multi-planar formats, which have distinct FourCC codes from the existing single-planar ones.

VIDIOC_QBUF, VIDIOC_DQBUF, VIDIOC_QUERYBUF A new struct `v4l2_plane` structure for describing planes is added. Arrays of this structure are passed in the new `m.planes` field of struct `v4l2_buffer`.

VIDIOC_REQBUFS Will allocate multi-planar buffers as requested.

3.2.1.27 Cropping, composing and scaling - the SELECTION API

Introduction

Some video capture devices can sample a subsection of a picture and shrink or enlarge it to an image of arbitrary size. Next, the devices can insert the image into larger one. Some video output devices can crop part of an input image, scale it up or down and insert it at an arbitrary scan line and horizontal offset into a video signal. We call these abilities cropping, scaling and composing.

On a video *capture* device the source is a video signal, and the cropping target determine the area actually sampled. The sink is an image stored in a memory buffer. The composing area specifies which part of the buffer is actually written to by the hardware.

On a video *output* device the source is an image in a memory buffer, and the cropping target is a part of an image to be shown on a display. The sink is the display or the graphics screen. The application may select the part of display where the image should be displayed. The size and position of such a window is controlled by the compose target.

Rectangles for all cropping and composing targets are defined even if the device does supports neither cropping nor composing. Their size and position will be fixed in such a case. If the device does not support scaling then the cropping and composing rectangles have the same size.

Selection targets

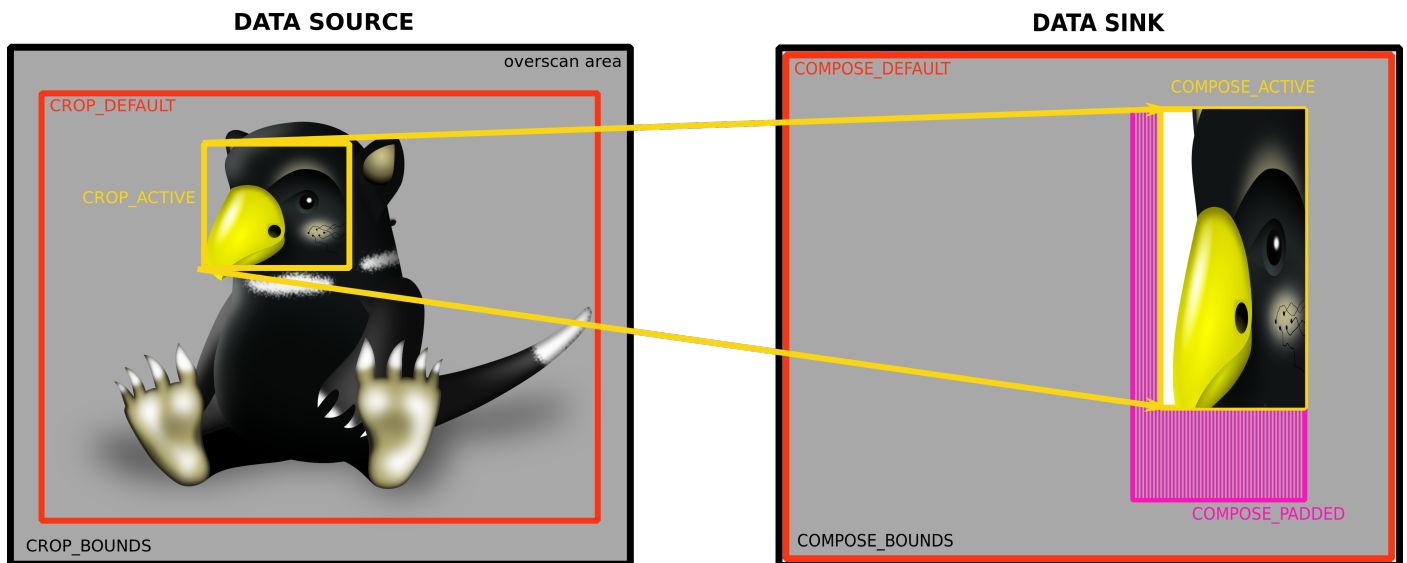


Fig. 2: Cropping and composing targets
Targets used by a cropping, composing and scaling process

See *Selection targets* for more information.

Configuration

Applications can use the *selection API* to select an area in a video signal or a buffer, and to query for default settings and hardware limits.

Video hardware can have various cropping, composing and scaling limitations. It may only scale up or down, support only discrete scaling factors, or have different scaling abilities in the horizontal and vertical directions. Also it may not support scaling at all. At the same time the cropping/composing rectangles may have to be aligned, and both the source and the sink may have arbitrary upper and lower size limits. Therefore, as usual, drivers are expected to adjust the requested parameters and return the actual values selected. An application can control the rounding behaviour using *constraint flags*.

Configuration of video capture

See figure *Cropping and composing targets* for examples of the selection targets available for a video capture device. It is recommended to configure the cropping targets before to the composing targets.

The range of coordinates of the top left corner, width and height of areas that can be sampled is given by the `V4L2_SEL_TGT_CROP_BOUNDS` target. It is recommended for the driver developers to put the top/left corner at position $(0, 0)$. The rectangle's coordinates are expressed in pixels.

The top left corner, width and height of the source rectangle, that is the area actually sampled, is given by the `V4L2_SEL_TGT_CROP` target. It uses the same coordinate system as `V4L2_SEL_TGT_CROP_BOUNDS`. The active cropping area must lie completely inside the capture

boundaries. The driver may further adjust the requested size and/or position according to hardware limitations.

Each capture device has a default source rectangle, given by the `V4L2_SEL_TGT_CROP_DEFAULT` target. This rectangle shall cover what the driver writer considers the complete picture. Drivers shall set the active crop rectangle to the default when the driver is first loaded, but not later.

The composing targets refer to a memory buffer. The limits of composing coordinates are obtained using `V4L2_SEL_TGT_COMPOSE_BOUNDS`. All coordinates are expressed in pixels. The rectangle's top/left corner must be located at position $(0,0)$. The width and height are equal to the image size set by `VIDIOC_S_FMT`.

The part of a buffer into which the image is inserted by the hardware is controlled by the `V4L2_SEL_TGT_COMPOSE` target. The rectangle's coordinates are also expressed in the same coordinate system as the bounds rectangle. The composing rectangle must lie completely inside bounds rectangle. The driver must adjust the composing rectangle to fit to the bounding limits. Moreover, the driver can perform other adjustments according to hardware limitations. The application can control rounding behaviour using *constraint flags*.

For capture devices the default composing rectangle is queried using `V4L2_SEL_TGT_COMPOSE_DEFAULT`. It is usually equal to the bounding rectangle.

The part of a buffer that is modified by the hardware is given by `V4L2_SEL_TGT_COMPOSE_PADDED`. It contains all pixels defined using `V4L2_SEL_TGT_COMPOSE` plus all padding data modified by hardware during insertion process. All pixels outside this rectangle *must not* be changed by the hardware. The content of pixels that lie inside the padded area but outside active area is undefined. The application can use the padded and active rectangles to detect where the rubbish pixels are located and remove them if needed.

Configuration of video output

For output devices targets and ioctls are used similarly to the video capture case. The *composing* rectangle refers to the insertion of an image into a video signal. The cropping rectangles refer to a memory buffer. It is recommended to configure the composing targets before to the cropping targets.

The cropping targets refer to the memory buffer that contains an image to be inserted into a video signal or graphical screen. The limits of cropping coordinates are obtained using `V4L2_SEL_TGT_CROP_BOUNDS`. All coordinates are expressed in pixels. The top/left corner is always point $(0,0)$. The width and height is equal to the image size specified using `VIDIOC_S_FMT` ioctl.

The top left corner, width and height of the source rectangle, that is the area from which image data are processed by the hardware, is given by the `V4L2_SEL_TGT_CROP`. Its coordinates are expressed in the same coordinate system as the bounds rectangle. The active cropping area must lie completely inside the crop boundaries and the driver may further adjust the requested size and/or position according to hardware limitations.

For output devices the default cropping rectangle is queried using `V4L2_SEL_TGT_CROP_DEFAULT`. It is usually equal to the bounding rectangle.

The part of a video signal or graphics display where the image is inserted by the hardware is controlled by `V4L2_SEL_TGT_COMPOSE` target. The rectangle's coordinates are expressed in pixels. The composing rectangle must lie completely inside the bounds rectangle. The driver

must adjust the area to fit to the bounding limits. Moreover, the driver can perform other adjustments according to hardware limitations.

The device has a default composing rectangle, given by the `V4L2_SEL_TGT_COMPOSE_DEFAULT` target. This rectangle shall cover what the driver writer considers the complete picture. It is recommended for the driver developers to put the top/left corner at position $(0,0)$. Drivers shall set the active composing rectangle to the default one when the driver is first loaded.

The devices may introduce additional content to video signal other than an image from memory buffers. It includes borders around an image. However, such a padded area is driver-dependent feature not covered by this document. Driver developers are encouraged to keep padded rectangle equal to active one. The padded target is accessed by the `V4L2_SEL_TGT_COMPOSE_PADDED` identifier. It must contain all pixels from the `V4L2_SEL_TGT_COMPOSE` target.

Scaling control

An application can detect if scaling is performed by comparing the width and the height of rectangles obtained using `V4L2_SEL_TGT_CROP` and `V4L2_SEL_TGT_COMPOSE` targets. If these are not equal then the scaling is applied. The application can compute the scaling ratios using these values.

Comparison with old cropping API

The selection API was introduced to cope with deficiencies of the older *CROP API*, that was designed to control simple capture devices. Later the cropping API was adopted by video output drivers. The ioctls used to select a part of the display were the video signal is inserted. It should be considered as an API abuse because the described operation is actually the composing. The selection API makes a clear distinction between composing and cropping operations by setting the appropriate targets.

The CROP API lacks any support for composing to and cropping from an image inside a memory buffer. The application could configure a capture device to fill only a part of an image by abusing V4L2 API. Cropping a smaller image from a larger one is achieved by setting the field bytesperline at struct `v4l2_pix_format`. Introducing an image offsets could be done by modifying field `m_userptr` at struct `v4l2_buffer` before calling `VIDIOC_QBUF`. Those operations should be avoided because they are not portable (endianness), and do not work for macroblock and Bayer formats and mmap buffers.

The selection API deals with configuration of buffer cropping/composing in a clear, intuitive and portable way. Next, with the selection API the concepts of the padded target and constraints flags are introduced. Finally, struct `v4l2_crop` and struct `v4l2_cropcap` have no reserved fields. Therefore there is no way to extend their functionality. The new struct `v4l2_selection` provides a lot of place for future extensions.

Driver developers are encouraged to implement only selection API. The former cropping API would be simulated using the new one.

Examples

(A video capture device is assumed; change `V4L2_BUF_TYPE_VIDEO_CAPTURE` for other devices; change target to `V4L2_SEL_TGT_COMPOSE_*` family to configure composing area)

Example: Resetting the cropping parameters

```

struct v4l2_selection sel = {
    .type = V4L2_BUF_TYPE_VIDEO_CAPTURE,
    .target = V4L2_SEL_TGT_CROP_DEFAULT,
};
ret = ioctl(fd, VIDIOC_G_SELECTION, &sel);
if (ret)
    exit(-1);
sel.target = V4L2_SEL_TGT_CROP;
ret = ioctl(fd, VIDIOC_S_SELECTION, &sel);
if (ret)
    exit(-1);

```

Setting a composing area on output of size of *at most* half of limit placed at a center of a display.

Example: Simple downscaling

```

struct v4l2_selection sel = {
    .type = V4L2_BUF_TYPE_VIDEO_OUTPUT,
    .target = V4L2_SEL_TGT_COMPOSE_BOUNDS,
};
struct v4l2_rect r;

ret = ioctl(fd, VIDIOC_G_SELECTION, &sel);
if (ret)
    exit(-1);
/* setting smaller compose rectangle */
r.width = sel.r.width / 2;
r.height = sel.r.height / 2;
r.left = sel.r.width / 4;
r.top = sel.r.height / 4;
sel.r = r;
sel.target = V4L2_SEL_TGT_COMPOSE;
sel.flags = V4L2_SEL_FLAG_LE;
ret = ioctl(fd, VIDIOC_S_SELECTION, &sel);
if (ret)
    exit(-1);

```

A video output device is assumed; change `V4L2_BUF_TYPE_VIDEO_OUTPUT` for other devices

Example: Querying for scaling factors

```
struct v4l2_selection compose = {
    .type = V4L2_BUF_TYPE_VIDEO_OUTPUT,
    .target = V4L2_SEL_TGT_COMPOSE,
};
struct v4l2_selection crop = {
    .type = V4L2_BUF_TYPE_VIDEO_OUTPUT,
    .target = V4L2_SEL_TGT_CROP,
};
double hscale, vscale;

ret = ioctl(fd, VIDIOC_G_SELECTION, &compose);
if (ret)
    exit(-1);
ret = ioctl(fd, VIDIOC_G_SELECTION, &crop);
if (ret)
    exit(-1);

/* computing scaling factors */
hscale = (double)compose.r.width / crop.r.width;
vscale = (double)compose.r.height / crop.r.height;
```

3.2.1.28 Image Cropping, Insertion and Scaling - the CROP API

Note: The CROP API is mostly superseded by the newer *SELECTION API*. The new API should be preferred in most cases, with the exception of pixel aspect ratio detection, which is implemented by *VIDIOC_CROPCAP* and has no equivalent in the SELECTION API. See *Comparison with old cropping API* for a comparison of the two APIs.

Some video capture devices can sample a subsection of the picture and shrink or enlarge it to an image of arbitrary size. We call these abilities cropping and scaling. Some video output devices can scale an image up or down and insert it at an arbitrary scan line and horizontal offset into a video signal.

Applications can use the following API to select an area in the video signal, query the default area and the hardware limits.

Note: Despite their name, the *VIDIOC_CROPCAP*, *VIDIOC_G_CROP* and *VIDIOC_S_CROP* ioctls apply to input as well as output devices.

Scaling requires a source and a target. On a video capture or overlay device the source is the video signal, and the cropping ioctls determine the area actually sampled. The target are images read by the application or overlaid onto the graphics screen. Their size (and position for an overlay) is negotiated with the *VIDIOC_G_FMT* and *VIDIOC_S_FMT* ioctls.

On a video output device the source are the images passed in by the application, and their size is again negotiated with the *VIDIOC_G_FMT* and *VIDIOC_S_FMT* ioctls, or may be encoded in a compressed video stream. The target is the video signal, and the cropping ioctls determine the area where the images are inserted.

Source and target rectangles are defined even if the device does not support scaling or the `VIDIOC_G_CROP` and `VIDIOC_S_CROP` ioctls. Their size (and position where applicable) will be fixed in this case.

Note: All capture and output devices that support the CROP or SELECTION API will also support the `VIDIOC_CROPCAP` ioctl.

Cropping Structures

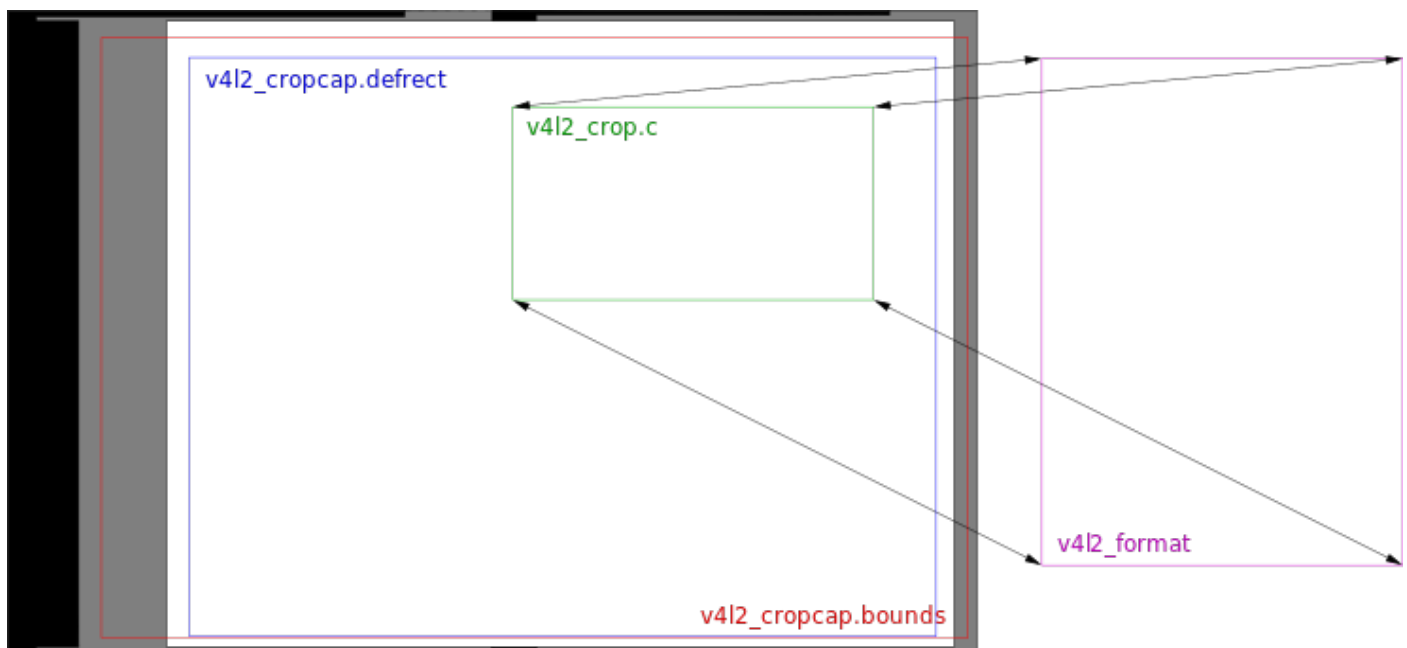


Fig. 3: Image Cropping, Insertion and Scaling
The cropping, insertion and scaling process

For capture devices the coordinates of the top left corner, width and height of the area which can be sampled is given by the bounds substructure of the struct `v4l2_cropcap` returned by the `VIDIOC_CROPCAP` ioctl. To support a wide range of hardware this specification does not define an origin or units. However by convention drivers should horizontally count unscaled samples relative to 0H (the leading edge of the horizontal sync pulse, see *Figure 4.1. Line synchronization*). Vertically ITU-R line numbers of the first field (see ITU R-525 line numbering for *525 lines* and for *625 lines*), multiplied by two if the driver can capture both fields.

The top left corner, width and height of the source rectangle, that is the area actually sampled, is given by struct `v4l2_crop` using the same coordinate system as struct `v4l2_cropcap`. Applications can use the `VIDIOC_G_CROP` and `VIDIOC_S_CROP` ioctls to get and set this rectangle. It must lie completely within the capture boundaries and the driver may further adjust the requested size and/or position according to hardware limitations.

Each capture device has a default source rectangle, given by the `defrect` substructure of struct `v4l2_cropcap`. The center of this rectangle shall align with the center of the active picture area of the video signal, and cover what the driver writer considers the complete picture. Drivers shall reset the source rectangle to the default when the driver is first loaded, but not later.

For output devices these structures and ioctls are used accordingly, defining the *target* rectangle where the images will be inserted into the video signal.

Scaling Adjustments

Video hardware can have various cropping, insertion and scaling limitations. It may only scale up or down, support only discrete scaling factors, or have different scaling abilities in horizontal and vertical direction. Also it may not support scaling at all. At the same time the struct `v4l2_crop` rectangle may have to be aligned, and both the source and target rectangles may have arbitrary upper and lower size limits. In particular the maximum width and height in struct `v4l2_crop` may be smaller than the struct `v4l2_cropcap` bounds area. Therefore, as usual, drivers are expected to adjust the requested parameters and return the actual values selected.

Applications can change the source or the target rectangle first, as they may prefer a particular image size or a certain area in the video signal. If the driver has to adjust both to satisfy hardware limitations, the last requested rectangle shall take priority, and the driver should preferably adjust the opposite one. The `VIDIOC_TRY_FMT` ioctl however shall not change the driver state and therefore only adjust the requested rectangle.

Suppose scaling on a video capture device is restricted to a factor 1:1 or 2:1 in either direction and the target image size must be a multiple of 16×16 pixels. The source cropping rectangle is set to defaults, which are also the upper limit in this example, of 640×400 pixels at offset 0, 0. An application requests an image size of 300×225 pixels, assuming video will be scaled down from the “full picture” accordingly. The driver sets the image size to the closest possible values 304×224 , then chooses the cropping rectangle closest to the requested size, that is 608×224 ($224 \times 2:1$ would exceed the limit 400). The offset 0, 0 is still valid, thus unmodified. Given the default cropping rectangle reported by `VIDIOC_CROPCAP` the application can easily propose another offset to center the cropping rectangle.

Now the application may insist on covering an area using a picture aspect ratio closer to the original request, so it asks for a cropping rectangle of 608×456 pixels. The present scaling factors limit cropping to 640×384 , so the driver returns the cropping size 608×384 and adjusts the image size to closest possible 304×192 .

Examples

Source and target rectangles shall remain unchanged across closing and reopening a device, such that piping data into or out of a device will work without special preparations. More advanced applications should ensure the parameters are suitable before starting I/O.

Note: On the next two examples, a video capture device is assumed; change `V4L2_BUF_TYPE_VIDEO_CAPTURE` for other types of device.

Example: Resetting the cropping parameters

```

struct v4l2_cropcap cropcap;
struct v4l2_crop crop;

memset (&cropcap, 0, sizeof (cropcap));
cropcap.type = V4L2_BUF_TYPE_VIDEO_CAPTURE;

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

memset (&crop, 0, sizeof (crop));
crop.type = V4L2_BUF_TYPE_VIDEO_CAPTURE;
crop.c = cropcap.defrect;

/* Ignore if cropping is not supported (EINVAL). */

if (-1 == ioctl (fd, VIDIOC_S_CROP, &crop)
    && errno != EINVAL) {
    perror ("VIDIOC_S_CROP");
    exit (EXIT_FAILURE);
}

```

Example: Simple downscaling

```

struct v4l2_cropcap cropcap;
struct v4l2_format format;

reset_cropping_parameters ();

/* Scale down to 1/4 size of full picture. */

memset (&format, 0, sizeof (format)); /* defaults */

format.type = V4L2_BUF_TYPE_VIDEO_CAPTURE;

format.fmt.pix.width = cropcap.defrect.width >> 1;
format.fmt.pix.height = cropcap.defrect.height >> 1;
format.fmt.pix.pixelformat = V4L2_PIX_FMT_YUYV;

if (-1 == ioctl (fd, VIDIOC_S_FMT, &format)) {
    perror ("VIDIOC_S_FORMAT");
    exit (EXIT_FAILURE);
}

/* We could check the actual image size now, the actual scaling factor
or if the driver can scale at all. */

```

Example: Selecting an output area

Note: This example assumes an output device.

```
struct v4l2_cropcap cropcap;
struct v4l2_crop crop;

memset (&cropcap, 0, sizeof (cropcap));
cropcap.type = V4L2_BUF_TYPE_VIDEO_OUTPUT;

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

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

crop.type = V4L2_BUF_TYPE_VIDEO_OUTPUT;
crop.c = cropcap.defrect;

/* Scale the width and height to 50 % of their original size
   and center the output. */

crop.c.width /= 2;
crop.c.height /= 2;
crop.c.left += crop.c.width / 2;
crop.c.top += crop.c.height / 2;

/* Ignore if cropping is not supported (EINVAL). */

if (-1 == ioctl (fd, VIDIOC_S_CROP, &crop)
    && errno != EINVAL) {
    perror ("VIDIOC_S_CROP");
    exit (EXIT_FAILURE);
}
```

Example: Current scaling factor and pixel aspect

Note: This example assumes a video capture device.

```
struct v4l2_cropcap cropcap;
struct v4l2_crop crop;
struct v4l2_format format;
double hscale, vscale;
double aspect;
int dwidth, dheight;

memset (&cropcap, 0, sizeof (cropcap));
cropcap.type = V4L2_BUF_TYPE_VIDEO_CAPTURE;

if (-1 == ioctl (fd, VIDIOC_CROPCAP, &cropcap)) {
```

```

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

memset (&crop, 0, sizeof (crop));
crop.type = V4L2_BUF_TYPE_VIDEO_CAPTURE;

if (-1 == ioctl (fd, VIDIOC_G_CROP, &crop)) {
    if (errno != EINVAL) {
        perror ("VIDIOC_G_CROP");
        exit (EXIT_FAILURE);
    }

    /* Cropping not supported. */
    crop.c = cropcap.defrect;
}

memset (&format, 0, sizeof (format));
format.fmt.type = V4L2_BUF_TYPE_VIDEO_CAPTURE;

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

/* The scaling applied by the driver. */

hscale = format.fmt.pix.width / (double) crop.c.width;
vscale = format.fmt.pix.height / (double) crop.c.height;

aspect = cropcap.pixelaspect.numerator /
    (double) cropcap.pixelaspect.denominator;
aspect = aspect * hscale / vscale;

/* Devices following ITU-R BT.601 do not capture
   square pixels. For playback on a computer monitor
   we should scale the images to this size. */

dwidth = format.fmt.pix.width / aspect;
dheight = format.fmt.pix.height;

```

3.2.1.29 Streaming Parameters

Streaming parameters are intended to optimize the video capture process as well as I/O. Presently applications can request a high quality capture mode with the `VIDIOC_S_PARM` ioctl.

The current video standard determines a nominal number of frames per second. If less than this number of frames is to be captured or output, applications can request frame skipping or duplicating on the driver side. This is especially useful when using the `read()` or `write()`, which are not augmented by timestamps or sequence counters, and to avoid unnecessary data copying.

Finally these ioctls can be used to determine the number of buffers used internally by a driver in read/write mode. For implications see the section discussing the `read()` function.

To get and set the streaming parameters applications call the `VIDIOC_G_PARM` and `VID-`

`IOC_S_PARM` ioctl, respectively. They take a pointer to a struct `v4l2_streamparm`, which contains a union holding separate parameters for input and output devices.

These ioctls are optional, drivers need not implement them. If so, they return the `EINVAL` error code.

3.2.2 Image Formats

The V4L2 API was primarily designed for devices exchanging image data with applications. The struct `v4l2_pix_format` and struct `v4l2_pix_format_mplane` structures define the format and layout of an image in memory. The former is used with the single-planar API, while the latter is used with the multi-planar version (see *Single- and multi-planar APIs*). Image formats are negotiated with the `VIDIOC_S_FMT` ioctl. (The explanations here focus on video capturing and output, for overlay frame buffer formats see also `VIDIOC_G_FBUF`.)

3.2.2.1 Single-planar format structure

`v4l2_pix_format`

Table 44: struct `v4l2_pix_format`

<code>__u32</code>	<code>width</code>	Image width in pixels.
<code>__u32</code>	<code>height</code>	Image height in pixels. If <code>field</code> is one of <code>V4L2_FIELD_TOP</code> , <code>V4L2_FIELD_BOTTOM</code> or <code>V4L2_FIELD_ALTERNATE</code> then <code>height</code> refers to the number of lines in the field, otherwise it refers to the number of lines in the frame (which is twice the field height for interlaced formats).
<p>Applications set these fields to request an image size, drivers return the closest possible values. In case of planar formats the <code>width</code> and <code>height</code> applies to the largest plane. To avoid ambiguities drivers must return values rounded up to a multiple of the scale factor of any smaller planes. For example when the image format is YUV 4:2:0, <code>width</code> and <code>height</code> must be multiples of two.</p> <p>For compressed formats that contain the resolution information encoded inside the stream, when fed to a stateful mem2mem decoder, the fields may be zero to rely on the decoder to detect the right values. For more details see <i>Memory-to-Memory Stateful Video Decoder Interface</i> and format descriptions.</p> <p>For compressed formats on the CAPTURE side of a stateful mem2mem encoder, the fields must be zero, since the coded size is expected to be calculated internally by the encoder itself, based on the OUTPUT side. For more details see <i>Memory-to-Memory Stateful Video Encoder Interface</i> and format descriptions.</p>		
<code>__u32</code>	<code>pixelformat</code>	The pixel format or type of compression, set by the application. This is a little endian <i>four character code</i> . V4L2 defines standard RGB formats in <i>RGB Formats</i> , YUV formats in <i>YUV Formats</i> , and reserved codes in <i>Reserved Image Formats</i>

Continued on next page

Table 44 - continued from previous page

__u32	field	Field order, from enum <i>v4l2_field</i> . Video images are typically interlaced. Applications can request to capture or output only the top or bottom field, or both fields interlaced or sequentially stored in one buffer or alternating in separate buffers. Drivers return the actual field order selected. For more details on fields see <i>Field Order</i> .
__u32	bytesperline	Distance in bytes between the leftmost pixels in two adjacent lines.
<p>Both applications and drivers can set this field to request padding bytes at the end of each line. Drivers however may ignore the value requested by the application, returning width times bytes per pixel or a larger value required by the hardware. That implies applications can just set this field to zero to get a reasonable default.</p> <p>Video hardware may access padding bytes, therefore they must reside in accessible memory. Consider cases where padding bytes after the last line of an image cross a system page boundary. Input devices may write padding bytes, the value is undefined. Output devices ignore the contents of padding bytes.</p> <p>When the image format is planar the bytesperline value applies to the first plane and is divided by the same factor as the width field for the other planes. For example the Cb and Cr planes of a YUV 4:2:0 image have half as many padding bytes following each line as the Y plane. To avoid ambiguities drivers must return a bytesperline value rounded up to a multiple of the scale factor.</p> <p>For compressed formats the bytesperline value makes no sense. Applications and drivers must set this to 0 in that case.</p>		
__u32	sizeimage	<p>Size in bytes of the buffer to hold a complete image, set by the driver. Usually this is bytesperline times height. When the image consists of variable length compressed data this is the number of bytes required by the codec to support the worst-case compression scenario.</p> <p>The driver will set the value for uncompressed images. Clients are allowed to set the sizeimage field for variable length compressed data flagged with <i>V4L2_FMT_FLAG_COMPRESSED</i> at <i>ioctl VIDIOC_ENUM_FMT</i>, but the driver may ignore it and set the value itself, or it may modify the provided value based on alignment requirements or minimum/maximum size requirements. If the client wants to leave this to the driver, then it should set sizeimage to 0.</p>

Continued on next page

Table 44 - continued from previous page

__u32	colorspace	Image colorspace, from enum <i>v4l2_colorspace</i> . This information supplements the pixel format and must be set by the driver for capture streams and by the application for output streams, see <i>Colorspaces</i> . If the application sets the flag <code>V4L2_PIX_FMT_FLAG_SET_CSC</code> then the application can set this field for a capture stream to request a specific colorspace for the captured image data. If the driver cannot handle requested conversion, it will return another supported colorspace. The driver indicates that colorspace conversion is supported by setting the flag <code>V4L2_FMT_FLAG_CSC_COLORSPACE</code> in the corresponding struct <code>v4l2_fmtdesc</code> during enumeration. See <i>Image Format Description Flags</i> .
__u32	priv	<p>This field indicates whether the remaining fields of the struct <i>v4l2_pix_format</i>, also called the extended fields, are valid. When set to <code>V4L2_PIX_FMT_PRIV_MAGIC</code>, it indicates that the extended fields have been correctly initialized. When set to any other value it indicates that the extended fields contain undefined values.</p> <p>Applications that wish to use the pixel format extended fields must first ensure that the feature is supported by querying the device for the <i>V4L2_CAP_EXT_PIX_FORMAT</i> capability. If the capability isn't set the pixel format extended fields are not supported and using the extended fields will lead to undefined results.</p> <p>To use the extended fields, applications must set the <code>priv</code> field to <code>V4L2_PIX_FMT_PRIV_MAGIC</code>, initialize all the extended fields and zero the unused bytes of the struct <code>v4l2_format</code> <code>raw_data</code> field.</p> <p>When the <code>priv</code> field isn't set to <code>V4L2_PIX_FMT_PRIV_MAGIC</code> drivers must act as if all the extended fields were set to zero. On return drivers must set the <code>priv</code> field to <code>V4L2_PIX_FMT_PRIV_MAGIC</code> and all the extended fields to applicable values.</p>
__u32	flags	Flags set by the application or driver, see <i>Format Flags</i> .
union {	(anonymous)	

Continued on next page

Table 44 - continued from previous page

__u32	ycbcr_enc	Y'CbCr encoding, from enum <i>v4l2_ycbcr_encoding</i> . This information supplements the colorspace and must be set by the driver for capture streams and by the application for output streams, see <i>Colorspaces</i> . If the application sets the flag V4L2_PIX_FMT_FLAG_SET_CSC then the application can set this field for a capture stream to request a specific Y'CbCr encoding for the captured image data. If the driver cannot handle requested conversion, it will return another supported encoding. This field is ignored for HSV pixelformats. The driver indicates that ycbcr_enc conversion is supported by setting the flag V4L2_FMT_FLAG_CSC_YCBCR_ENC in the corresponding struct v4l2_fmtdesc during enumeration. See <i>Image Format Description Flags</i> .
__u32	hsv_enc	HSV encoding, from enum <i>v4l2_hsv_encoding</i> . This information supplements the colorspace and must be set by the driver for capture streams and by the application for output streams, see <i>Colorspaces</i> . If the application sets the flag V4L2_PIX_FMT_FLAG_SET_CSC then the application can set this field for a capture stream to request a specific HSV encoding for the captured image data. If the driver cannot handle requested conversion, it will return another supported encoding. This field is ignored for non-HSV pixelformats. The driver indicates that hsv_enc conversion is supported by setting the flag V4L2_FMT_FLAG_CSC_HSV_ENC in the corresponding struct v4l2_fmtdesc during enumeration. See <i>Image Format Description Flags</i> .
}		
__u32	quantization	Quantization range, from enum <i>v4l2_quantization</i> . This information supplements the colorspace and must be set by the driver for capture streams and by the application for output streams, see <i>Colorspaces</i> . If the application sets the flag V4L2_PIX_FMT_FLAG_SET_CSC then the application can set this field for a capture stream to request a specific quantization range for the captured image data. If the driver cannot handle requested conversion, it will return another supported quantization. The driver indicates that quantization conversion is supported by setting the flag V4L2_FMT_FLAG_CSC_QUANTIZATION in the corresponding struct v4l2_fmtdesc during enumeration. See <i>Image Format Description Flags</i> .

Continued on next page

Table 44 - continued from previous page

__u32	xfer_func	Transfer function, from enum <i>v4l2_xfer_func</i> . This information supplements the <i>colospace</i> and must be set by the driver for capture streams and by the application for output streams, see <i>Colorspaces</i> . If the application sets the flag <code>V4L2_PIX_FMT_FLAG_SET_CSC</code> then the application can set this field for a capture stream to request a specific transfer function for the captured image data. If the driver cannot handle requested conversion, it will return another supported transfer function. The driver indicates that <code>xfer_func</code> conversion is supported by setting the flag <code>V4L2_FMT_FLAG_CSC_XFER_FUNC</code> in the corresponding struct <code>v4l2_fmtdesc</code> during enumeration. See <i>Image Format Description Flags</i> .
-------	-----------	--

Table 45: Format Flags

V4L2_PIX_FMT_FLAG_PREMUL_ALPHA	0x00000001	The color values are premultiplied by the alpha channel value. For example, if a light blue pixel with 50% transparency was described by RGBA values (128, 192, 255, 128), the same pixel described with premultiplied colors would be described by RGBA values (64, 96, 128, 128)
V4L2_PIX_FMT_FLAG_SET_CSC	0x00000002	Set by the application. It is only used for capture and is ignored for output streams. If set, then request the device to do colorspace conversion from the received colorspace to the requested colorspace values. If the colorimetry field (<code>colospace</code> , <code>xfer_func</code> , <code>ycbcr_enc</code> , <code>hsv_enc</code> or <code>quantization</code>) is set to <code>*_DEFAULT</code> , then that colorimetry setting will remain unchanged from what was received. So in order to change the quantization, only the quantization field shall be set to non default value (<code>V4L2_QUANTIZATION_FULL_RANGE</code> or <code>V4L2_QUANTIZATION_LIM_RANGE</code>) and all other colorimetry fields shall be set to <code>*_DEFAULT</code> . To check which conversions are supported by the hardware for the current pixel format, see <i>Image Format Description Flags</i> .

3.2.2.2 Multi-planar format structures

The struct `v4l2_plane_pix_format` structures define size and layout for each of the planes in a multi-planar format. The struct `v4l2_pix_format_mplane` structure contains information common to all planes (such as image width and height) and an array of struct `v4l2_plane_pix_format` structures, describing all planes of that format.

`v4l2_plane_pix_format`

Table 46: struct `v4l2_plane_pix_format`

<code>__u32</code>	<code>sizeimage</code>	Maximum size in bytes required for image data in this plane, set by the driver. When the image consists of variable length compressed data this is the number of bytes required by the codec to support the worst-case compression scenario. The driver will set the value for uncompressed images. Clients are allowed to set the <code>sizeimage</code> field for variable length compressed data flagged with <code>V4L2_FMT_FLAG_COMPRESSED</code> at <code>ioctl VIDIOC_ENUM_FMT</code> , but the driver may ignore it and set the value itself, or it may modify the provided value based on alignment requirements or minimum/maximum size requirements. If the client wants to leave this to the driver, then it should set <code>sizeimage</code> to 0.
<code>__u32</code>	<code>bytesperline</code>	Distance in bytes between the leftmost pixels in two adjacent lines. See struct <code>v4l2_pix_format</code> .
<code>__u16</code>	<code>reserved[6]</code>	Reserved for future extensions. Should be zeroed by drivers and applications.

`v4l2_pix_format_mplane`

Table 47: struct v4l2_pix_format_mplane

__u32	width	Image width in pixels. See struct <i>v4l2_pix_format</i> .
__u32	height	Image height in pixels. See struct <i>v4l2_pix_format</i> .
__u32	pixelformat	The pixel format. Both single- and multi-planar four character codes can be used.
__u32	field	Field order, from enum <i>v4l2_field</i> . See struct <i>v4l2_pix_format</i> .
__u32	colospace	Colorspace encoding, from enum <i>v4l2_colospace</i> . See struct <i>v4l2_pix_format</i> .
struct <i>v4l2_plane_pix_format</i>	plane_fmt[VIDEO_MAX_PLANES]	An array of structures describing format of each plane this pixel format consists of. The number of valid entries in this array has to be put in the num_planes field.
__u8	num_planes	Number of planes (i.e. separate memory buffers) for this format and the number of valid entries in the plane_fmt array.
__u8	flags	Flags set by the application or driver, see <i>Format Flags</i> .
union {	(anonymous)	
__u8	ycbcr_enc	Y'CbCr encoding, from enum <i>v4l2_ycbcr_encoding</i> . See struct <i>v4l2_pix_format</i> .
__u8	hsv_enc	HSV encoding, from enum <i>v4l2_hsv_encoding</i> . See struct <i>v4l2_pix_format</i> .
}		
__u8	quantization	Quantization range, from enum <i>v4l2_quantization</i> . See struct <i>v4l2_pix_format</i> .
__u8	xfer_func	Transfer function, from enum <i>v4l2_xfer_func</i> . See struct <i>v4l2_pix_format</i> .
__u8	reserved[7]	Reserved for future extensions. Should be zeroed by drivers and applications.

3.2.2.3 Standard Image Formats

In order to exchange images between drivers and applications, it is necessary to have standard image data formats which both sides will interpret the same way. V4L2 includes several such formats, and this section is intended to be an unambiguous specification of the standard image data formats in V4L2.

V4L2 drivers are not limited to these formats, however. Driver-specific formats are possible. In that case the application may depend on a codec to convert images to one of the standard formats when needed. But the data can still be stored and retrieved in the proprietary format. For example, a device may support a proprietary compressed format. Applications can still capture and save the data in the compressed format, saving much disk space, and later use a codec to convert the images to the X Windows screen format when the video is to be displayed.

Even so, ultimately, some standard formats are needed, so the V4L2 specification would not be complete without well-defined standard formats.

The V4L2 standard formats are mainly uncompressed formats. The pixels are always arranged in memory from left to right, and from top to bottom. The first byte of data in the image buffer is always for the leftmost pixel of the topmost row. Following that is the pixel immediately to its right, and so on until the end of the top row of pixels. Following the rightmost pixel of the row there may be zero or more bytes of padding to guarantee that each row of pixel data has a certain alignment. Following the pad bytes, if any, is data for the leftmost pixel of the second row from the top, and so on. The last row has just as many pad bytes after it as the other rows.

In V4L2 each format has an identifier which looks like `PIX_FMT_XXX`, defined in the `videodev2.h` header file. These identifiers represent *four character (FourCC) codes* which are also listed below, however they are not the same as those used in the Windows world.

For some formats, data is stored in separate, discontinuous memory buffers. Those formats are identified by a separate set of FourCC codes and are referred to as “multi-planar formats”. For example, a `YUV422` frame is normally stored in one memory buffer, but it can also be placed in two or three separate buffers, with Y component in one buffer and CbCr components in another in the 2-planar version or with each component in its own buffer in the 3-planar case. Those sub-buffers are referred to as “planes”.

3.2.2.4 Indexed Format

In this format each pixel is represented by an 8 bit index into a 256 entry ARGB palette. It is intended for *Video Output Overlays* only. There are no ioctls to access the palette, this must be done with ioctls of the Linux framebuffer API.

Table 48: Indexed Image Format

Identifier	Code	Bit	Byte 0							
			7	6	5	4	3	2	1	0
V4L2_PIX_FMT_PAL8	'PAL8'		i ₇	i ₆	i ₅	i ₄	i ₃	i ₂	i ₁	i ₀

3.2.2.5 RGB Formats

These formats encode each pixel as a triplet of RGB values. They are packed formats, meaning that the RGB values for one pixel are stored consecutively in memory and each pixel consumes an integer number of bytes. When the number of bits required to store a pixel is not aligned to a byte boundary, the data is padded with additional bits to fill the remaining byte.

The formats differ by the number of bits per RGB component (typically but not always the same for all components), the order of components in memory, and the presence of an alpha component or additional padding bits.

The usage and value of the alpha bits in formats that support them (named ARGB or a permutation thereof, collectively referred to as alpha formats) depend on the device type and hardware operation. *Capture* devices (including capture queues of mem-to-mem devices) fill the alpha component in memory. When the device captures an alpha channel the alpha component will have a meaningful value. Otherwise, when the device doesn't capture an alpha channel but can set the alpha bit to a user-configurable value, the `V4L2_CID_ALPHA_COMPONENT` control is used to specify that alpha value, and the alpha component of all pixels will be set to the value specified by that control. Otherwise a corresponding format without an alpha component (XRGB or XBGR) must be used instead of an alpha format.

Output devices (including output queues of mem-to-mem devices and *video output overlay* devices) read the alpha component from memory. When the device processes the alpha channel the alpha component must be filled with meaningful values by applications. Otherwise a corresponding format without an alpha component (XRGB or XBGR) must be used instead of an alpha format.

Formats that contain padding bits are named XRGB (or a permutation thereof). The padding bits contain undefined values and must be ignored by applications, devices and drivers, for both *Video Capture Interface* and *Video Output Interface* devices.

Note:

- In all the tables that follow, bit 7 is the most significant bit in a byte.
- ‘r’, ‘g’ and ‘b’ denote bits of the red, green and blue components respectively. ‘a’ denotes bits of the alpha component (if supported by the format), and ‘x’ denotes padding bits.

Less Than 8 Bits Per Component

These formats store an RGB triplet in one, two or four bytes. They are named based on the order of the RGB components as seen in a 8-, 16- or 32-bit word, which is then stored in memory in little endian byte order (unless otherwise noted by the presence of bit 31 in the 4CC value), and on the number of bits for each component. For instance, the RGB565 format stores a pixel in a 16-bit word [15:0] laid out at as [R₄ R₃ R₂ R₁ R₀ G₅ G₄ G₃ G₂ G₁ G₀ B₄ B₃ B₂ B₁ B₀], and stored in memory in two bytes, [R₄ R₃ R₂ R₁ R₀ G₅ G₄ G₃] followed by [G₂ G₁ G₀ B₄ B₃ B₂ B₁ B₀].

Table 49: RGB Formats With Less Than 8 Bits Per Component

Identifier	Code	Byte 0 in memory								Byte 1								Byte 2								Byte 3											
		7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0				
V4L2_PIX_FMT_RGB332	‘RGB1’	r ₂	r ₁	r ₀	g ₂	g ₁	g ₀	b ₁	b ₀																												
V4L2_PIX_FMT_ARGB444	‘AR12’	g ₃	g ₂	g ₁	g ₀	b ₃	b ₂	b ₁	b ₀	a ₃	a ₂	a ₁	a ₀	r ₃	r ₂	r ₁	r ₀																				
V4L2_PIX_FMT_XRGB444	‘XR12’	g ₃	g ₂	g ₁	g ₀	b ₃	b ₂	b ₁	b ₀	x	x	x	x	r ₃	r ₂	r ₁	r ₀																				
V4L2_PIX_FMT_RGBA444	‘RA12’	b ₃	b ₂	b ₁	b ₀	a ₃	a ₂	a ₁	a ₀	r ₃	r ₂	r ₁	r ₀	g ₃	g ₂	g ₁	g ₀																				
V4L2_PIX_FMT_RGBX444	‘RX12’	b ₃	b ₂	b ₁	b ₀	x	x	x	x	r ₃	r ₂	r ₁	r ₀	g ₃	g ₂	g ₁	g ₀																				
V4L2_PIX_FMT_ABGR444	‘AB12’	g ₃	g ₂	g ₁	g ₀	r ₃	r ₂	r ₁	r ₀	a ₃	a ₂	a ₁	a ₀	b ₃	b ₂	b ₁	b ₀																				
V4L2_PIX_FMT_XBGR444	‘XB12’	g ₃	g ₂	g ₁	g ₀	r ₃	r ₂	r ₁	r ₀	x	x	x	x	b ₃	b ₂	b ₁	b ₀																				
V4L2_PIX_FMT_BGRA444	‘BA12’	r ₃	r ₂	r ₁	r ₀	a ₃	a ₂	a ₁	a ₀	b ₃	b ₂	b ₁	b ₀	g ₃	g ₂	g ₁	g ₀																				
V4L2_PIX_FMT_BGRX444	‘BX12’	r ₃	r ₂	r ₁	r ₀	x	x	x	x	b ₃	b ₂	b ₁	b ₀	g ₃	g ₂	g ₁	g ₀																				
V4L2_PIX_FMT_ARGB555	‘AR15’	g ₂	g ₁	g ₀	b ₄	b ₃	b ₂	b ₁	b ₀	a	r ₄	r ₃	r ₂	r ₁	r ₀	g ₄	g ₃																				
V4L2_PIX_FMT_XRGB555	‘XR15’	g ₂	g ₁	g ₀	b ₄	b ₃	b ₂	b ₁	b ₀	x	r ₄	r ₃	r ₂	r ₁	r ₀	g ₄	g ₃																				
V4L2_PIX_FMT_RGBA555	‘RA15’	g ₁	g ₀	b ₄	b ₃	b ₂	b ₁	b ₀	a	r ₄	r ₃	r ₂	r ₁	r ₀	g ₄	g ₃	g ₂																				
V4L2_PIX_FMT_RGBX555	‘RX15’	g ₁	g ₀	b ₄	b ₃	b ₂	b ₁	b ₀	x	r ₄	r ₃	r ₂	r ₁	r ₀	g ₄	g ₃	g ₂																				
V4L2_PIX_FMT_ABGR555	‘AB15’	g ₂	g ₁	g ₀	r ₄	r ₃	r ₂	r ₁	r ₀	a	b ₄	b ₃	b ₂	b ₁	b ₀	g ₄	g ₃	g ₂																			
V4L2_PIX_FMT_XBGR555	‘XB15’	g ₂	g ₁	g ₀	r ₄	r ₃	r ₂	r ₁	r ₀	x	b ₄	b ₃	b ₂	b ₁	b ₀	g ₄	g ₃	g ₂																			
V4L2_PIX_FMT_BGRA555	‘BA15’	g ₁	g ₀	r ₄	r ₃	r ₂	r ₁	r ₀	a	b ₄	b ₃	b ₂	b ₁	b ₀	g ₄	g ₃	g ₂																				
V4L2_PIX_FMT_BGRX555	‘BX15’	g ₁	g ₀	r ₄	r ₃	r ₂	r ₁	r ₀	x	b ₄	b ₃	b ₂	b ₁	b ₀	g ₄	g ₃	g ₂																				
V4L2_PIX_FMT_RGB565	‘RGBP’	g ₂	g ₁	g ₀	b ₄	b ₃	b ₂	b ₁	b ₀	r ₄	r ₃	r ₂	r ₁	r ₀	g ₅	g ₄	g ₃																				
V4L2_PIX_FMT_ARGB555X	‘AR15’ (1 << 31)	a	r ₄	r ₃	r ₂	r ₁	r ₀	g ₄	g ₃	g ₂	g ₁	g ₀	b ₄	b ₃	b ₂	b ₁	b ₀																				
V4L2_PIX_FMT_XRGB555X	‘XR15’ (1 << 31)	x	r ₄	r ₃	r ₂	r ₁	r ₀	g ₄	g ₃	g ₂	g ₁	g ₀	b ₄	b ₃	b ₂	b ₁	b ₀																				
V4L2_PIX_FMT_RGB565X	‘RGRB’	r ₄	r ₃	r ₂	r ₁	r ₀	g ₅	g ₄	g ₃	g ₂	g ₁	g ₀	b ₄	b ₃	b ₂	b ₁	b ₀																				
V4L2_PIX_FMT_BGR666	‘BGRH’	b ₅	b ₄	b ₃	b ₂	b ₁	b ₀	g ₅	g ₄	g ₃	g ₂	g ₁	g ₀	r ₅	r ₄	r ₃	r ₂	r ₁	r ₀	x	x	x	x	x	x	x	x	x	x	x	x						

3.2.2.6 Raw Bayer Formats

Description

The raw Bayer formats are used by image sensors before much if any processing is performed on the image. The formats contain green, red and blue components, with alternating lines of red and green, and blue and green pixels in different orders. See also [the Wikipedia article on Bayer filter](#).

**V4L2_PIX_FMT_SRGGB8 ('RGGB'), V4L2_PIX_FMT_SGRBG8 ('GRBG'),
V4L2_PIX_FMT_SGBRG8 ('GBRG'), V4L2_PIX_FMT_SBGGR8 ('BA81'),**

8-bit Bayer formats

Description

These four pixel formats are raw sRGB / Bayer formats with 8 bits per sample. Each sample is stored in a byte. Each n-pixel row contains n/2 green samples and n/2 blue or red samples, with alternating red and blue rows. They are conventionally described as GRGR... BGBG..., RGRG... GBGB..., etc. Below is an example of a small V4L2_PIX_FMT_SBGGR8 image:

Byte Order. Each cell is one byte.

start + 0:	B ₀₀	G ₀₁	B ₀₂	G ₀₃
start + 4:	G ₁₀	R ₁₁	G ₁₂	R ₁₃
start + 8:	B ₂₀	G ₂₁	B ₂₂	G ₂₃
start + 12:	G ₃₀	R ₃₁	G ₃₂	R ₃₃

**V4L2_PIX_FMT_SRGGB10 ('RG10'), V4L2_PIX_FMT_SGRBG10 ('BA10'),
V4L2_PIX_FMT_SGBRG10 ('GB10'), V4L2_PIX_FMT_SBGGR10 ('BG10'),**

V4L2_PIX_FMT_SGRBG10 V4L2_PIX_FMT_SGBRG10 V4L2_PIX_FMT_SBGGR10 10-bit Bayer formats expanded to 16 bits

Description

These four pixel formats are raw sRGB / Bayer formats with 10 bits per sample. Each sample is stored in a 16-bit word, with 6 unused high bits filled with zeros. Each n-pixel row contains n/2 green samples and n/2 blue or red samples, with alternating red and blue rows. Bytes are stored in memory in little endian order. They are conventionally described as GRGR... BGBG..., RGRG... GBGB..., etc. Below is an example of one of these formats:

Byte Order. Each cell is one byte, the 6 most significant bits in the high bytes are 0.

start + 0:	B _{00low}	B _{00high}	G _{01low}	G _{01high}	B _{02low}	B _{02high}	G _{03low}	G _{03high}
start + 8:	G _{10low}	G _{10high}	R _{11low}	R _{11high}	G _{12low}	G _{12high}	R _{13low}	R _{13high}
start + 16:	B _{20low}	B _{20high}	G _{21low}	G _{21high}	B _{22low}	B _{22high}	G _{23low}	G _{23high}
start + 24:	G _{30low}	G _{30high}	R _{31low}	R _{31high}	G _{32low}	G _{32high}	R _{33low}	R _{33high}

V4L2_PIX_FMT_SRGG10P ('pRAA'), **V4L2_PIX_FMT_SGRBG10P** ('pgAA'),
V4L2_PIX_FMT_SGBRG10P ('pGAA'), **V4L2_PIX_FMT_SBGGR10P** ('pBAA'),

V4L2_PIX_FMT_SGRBG10P V4L2_PIX_FMT_SGBRG10P V4L2_PIX_FMT_SBGGR10P 10-bit packed Bayer formats

Description

These four pixel formats are packed raw sRGB / Bayer formats with 10 bits per sample. Every four consecutive samples are packed into 5 bytes. Each of the first 4 bytes contain the 8 high order bits of the pixels, and the 5th byte contains the 2 least significant bits of each pixel, in the same order.

Each n-pixel row contains n/2 green samples and n/2 blue or red samples, with alternating green-red and green-blue rows. They are conventionally described as GRGR... BGBG..., RGRG... GBGB..., etc. Below is an example of a small V4L2_PIX_FMT_SBGGR10P image:

Byte Order. Each cell is one byte.

start + 0:	B _{00high}	G _{01high}	B _{02high}	G _{03high}	G _{03low} (bits 7-6) B _{02low} (bits 5-4) G _{01low} (bits 3-2) B _{00low} (bits 1-0)
start + 5:	G _{10high}	R _{11high}	G _{12high}	R _{13high}	R _{13low} (bits 7-6) G _{12low} (bits 5-4) R _{11low} (bits 3-2) G _{10low} (bits 1-0)
start + 10:	B _{20high}	G _{21high}	B _{22high}	G _{23high}	G _{23low} (bits 7-6) B _{22low} (bits 5-4) G _{21low} (bits 3-2) B _{20low} (bits 1-0)
start + 15:	G _{30high}	R _{31high}	G _{32high}	R _{33high}	R _{33low} (bits 7-6) G _{32low} (bits 5-4) R _{31low} (bits 3-2) G _{30low} (bits 1-0)

V4L2_PIX_FMT_SBGGR10ALAW8 ('aBA8'), **V4L2_PIX_FMT_SGBRG10ALAW8** ('aGA8'),
V4L2_PIX_FMT_SGRBG10ALAW8 ('agA8'), **V4L2_PIX_FMT_SRGG10ALAW8** ('aRA8'),

V4L2_PIX_FMT_SGBRG10ALAW8 V4L2_PIX_FMT_SGRBG10ALAW8
V4L2_PIX_FMT_SRGG10ALAW8 10-bit Bayer formats compressed to 8 bits

Description

These four pixel formats are raw sRGB / Bayer formats with 10 bits per color compressed to 8 bits each, using the A-LAW algorithm. Each color component consumes 8 bits of memory. In other respects this format is similar to *V4L2_PIX_FMT_SRGG8* ('RGGB'), *V4L2_PIX_FMT_SGRBG8* ('GRBG'), *V4L2_PIX_FMT_SGBRG8* ('GBRG'), *V4L2_PIX_FMT_SBGGR8* ('BA81'),.

V4L2_PIX_FMT_SBGGR10DPCM8 ('bBA8'), V4L2_PIX_FMT_SGBRG10DPCM8 ('bGA8'), V4L2_PIX_FMT_SGRBG10DPCM8 ('BD10'), V4L2_PIX_FMT_SRGG10DPCM8 ('bRA8'),

man V4L2_PIX_FMT_SBGGR10DPCM8(2)

V4L2_PIX_FMT_SGBRG10DPCM8 V4L2_PIX_FMT_SGRBG10DPCM8
 V4L2_PIX_FMT_SRGG10DPCM8 10-bit Bayer formats compressed to 8 bits

Description

These four pixel formats are raw sRGB / Bayer formats with 10 bits per colour compressed to 8 bits each, using DPCM compression. DPCM, differential pulse-code modulation, is lossy. Each colour component consumes 8 bits of memory. In other respects this format is similar to *V4L2_PIX_FMT_SRGG10 ('RG10'), V4L2_PIX_FMT_SGRBG10 ('BA10'), V4L2_PIX_FMT_SGBRG10 ('GB10'), V4L2_PIX_FMT_SBGGR10 ('BG10')*.

V4L2_PIX_FMT_IPU3_SBGGR10 ('ip3b'), V4L2_PIX_FMT_IPU3_SGBRG10 ('ip3g'), V4L2_PIX_FMT_IPU3_SGRBG10 ('ip3G'), V4L2_PIX_FMT_IPU3_SRGG10 ('ip3r')

10-bit Bayer formats

Description

These four pixel formats are used by Intel IPU3 driver, they are raw sRGB / Bayer formats with 10 bits per sample with every 25 pixels packed to 32 bytes leaving 6 most significant bits padding in the last byte. The format is little endian.

In other respects this format is similar to *V4L2_PIX_FMT_SRGG10 ('RG10'), V4L2_PIX_FMT_SGRBG10 ('BA10'), V4L2_PIX_FMT_SGBRG10 ('GB10'), V4L2_PIX_FMT_SBGGR10 ('BG10')*. Below is an example of a small image in V4L2_PIX_FMT_IPU3_SBGGR10 format.

Byte Order. Each cell is one byte.

start + 0:	B0000low	G0001low(bits 7-2) B0000high(bits 1-0)	B0002low(bits 7-4) G0001high(bits 3-0)	G0003low(bits 7-6) B0002high(bits 5-0)
start + 4:	G0003high	B0004low	G0005low(bits 7-2) B0004high(bits 1-0)	B0006low(bits 7-4) G0005high(bits 3-0)
start + 8:	G0007low(bits 7-6) B0006high(bits 5-0)	G0007high	B0008low	G0009low(bits 7-2) B0008high(bits 1-0)
start + 12:	B0010low(bits 7-4) G0009high(bits 3-0)	G0011low(bits 7-6) B0010high(bits 5-0)	G0011high	B0012low
start + 16:	G0013low(bits 7-2) B0012high(bits 1-0)	B0014low(bits 7-4) G0013high(bits 3-0)	G0015low(bits 7-6) B0014high(bits 5-0)	G0015high
start + 20	B0016low	G0017low(bits 7-2) B0016high(bits 1-0)	B0018low(bits 7-4) G0017high(bits 3-0)	G0019low(bits 7-6) B0018high(bits 5-0)

Continued on next page

Table 52 - continued from previous page

start + 24:	G0019high	B0020low	G0021low(bits 7-2) B0020high(bits 1-0)	B0022low(bits 7-4) G0021high(bits 3-0)
start + 28:	G0023low(bits 7-6) B0022high(bits 5-0)	G0023high	B0024low	B0024high(bits 1-0)
start + 32:	G0100low	R0101low(bits 7-2) G0100high(bits 1-0)	G0102low(bits 7-4) R0101high(bits 3-0)	R0103low(bits 7-6) G0102high(bits 5-0)
start + 36:	R0103high	G0104low	R0105low(bits 7-2) G0104high(bits 1-0)	G0106low(bits 7-4) R0105high(bits 3-0)
start + 40:	R0107low(bits 7-6) G0106high(bits 5-0)	R0107high	G0108low	R0109low(bits 7-2) G0108high(bits 1-0)
start + 44:	G0110low(bits 7-4) R0109high(bits 3-0)	R0111low(bits 7-6) G0110high(bits 5-0)	R0111high	G0112low
start + 48:	R0113low(bits 7-2) G0112high(bits 1-0)	G0114low(bits 7-4) R0113high(bits 3-0)	R0115low(bits 7-6) G0114high(bits 5-0)	R0115high
start + 52:	G0116low	R0117low(bits 7-2) G0116high(bits 1-0)	G0118low(bits 7-4) R0117high(bits 3-0)	R0119low(bits 7-6) G0118high(bits 5-0)
start + 56:	R0119high	G0120low	R0121low(bits 7-2) G0120high(bits 1-0)	G0122low(bits 7-4) R0121high(bits 3-0)
start + 60:	R0123low(bits 7-6) G0122high(bits 5-0)	R0123high	G0124low	G0124high(bits 1-0)
start + 64:	B0200low	G0201low(bits 7-2) B0200high(bits 1-0)	B0202low(bits 7-4) G0201high(bits 3-0)	G0203low(bits 7-6) B0202high(bits 5-0)
start + 68:	G0203high	B0204low	G0205low(bits 7-2) B0204high(bits 1-0)	B0206low(bits 7-4) G0205high(bits 3-0)
start + 72:	G0207low(bits 7-6) B0206high(bits 5-0)	G0207high	B0208low	G0209low(bits 7-2) B0208high(bits 1-0)
start + 76:	B0210low(bits 7-4) G0209high(bits 3-0)	G0211low(bits 7-6) B0210high(bits 5-0)	G0211high	B0212low
start + 80:	G0213low(bits 7-2) B0212high(bits 1-0)	B0214low(bits 7-4) G0213high(bits 3-0)	G0215low(bits 7-6) B0214high(bits 5-0)	G0215high
start + 84:	B0216low	G0217low(bits 7-2) B0216high(bits 1-0)	B0218low(bits 7-4) G0217high(bits 3-0)	G0219low(bits 7-6) B0218high(bits 5-0)
start + 88:	G0219high	B0220low	G0221low(bits 7-2) B0220high(bits 1-0)	B0222low(bits 7-4) G0221high(bits 3-0)
start + 92:	G0223low(bits 7-6) B0222high(bits 5-0)	G0223high	B0224low	B0224high(bits 1-0)

Continued on next page

Table 52 - continued from previous page

start + 96:	G _{0300low}	R _{0301low} (bits 7-2) G _{0300high} (bits 1-0)	G _{0302low} (bits 7-4) R _{0301high} (bits 3-0)	R _{0303low} (bits 7-6) G _{0302high} (bits 5-0)
start + 100:	R _{0303high}	G _{0304low}	R _{0305low} (bits 7-2) G _{0304high} (bits 1-0)	G _{0306low} (bits 7-4) R _{0305high} (bits 3-0)
start + 104:	R _{0307low} (bits 7-6) G _{0306high} (bits 5-0)	R _{0307high}	G _{0308low}	R _{0309low} (bits 7-2) G _{0308high} (bits 1-0)
start + 108:	G _{0310low} (bits 7-4) R _{0309high} (bits 3-0)	R _{0311low} (bits 7-6) G _{0310high} (bits 5-0)	R _{0311high}	G _{0312low}
start + 112:	R _{0313low} (bits 7-2) G _{0312high} (bits 1-0)	G _{0314low} (bits 7-4) R _{0313high} (bits 3-0)	R _{0315low} (bits 7-6) G _{0314high} (bits 5-0)	R _{0315high}
start + 116:	G _{0316low}	R _{0317low} (bits 7-2) G _{0316high} (bits 1-0)	G _{0318low} (bits 7-4) R _{0317high} (bits 3-0)	R _{0319low} (bits 7-6) G _{0318high} (bits 5-0)
start + 120:	R _{0319high}	G _{0320low}	R _{0321low} (bits 7-2) G _{0320high} (bits 1-0)	G _{0322low} (bits 7-4) R _{0321high} (bits 3-0)
start + 124:	R _{0323low} (bits 7-6) G _{0322high} (bits 5-0)	R _{0323high}	G _{0324low}	G _{0324high} (bits 1-0)

**V4L2_PIX_FMT_SRGB12 ('RG12'), V4L2_PIX_FMT_SGRBG12 ('BA12'),
V4L2_PIX_FMT_SGBRG12 ('GB12'), V4L2_PIX_FMT_SBGGR12 ('BG12'),**

V4L2_PIX_FMT_SGRBG12 V4L2_PIX_FMT_SGBRG12 V4L2_PIX_FMT_SBGGR12 12-bit Bayer formats expanded to 16 bits

Description

These four pixel formats are raw sRGB / Bayer formats with 12 bits per colour. Each colour component is stored in a 16-bit word, with 4 unused high bits filled with zeros. Each n-pixel row contains n/2 green samples and n/2 blue or red samples, with alternating red and blue rows. Bytes are stored in memory in little endian order. They are conventionally described as GRGR... BGBG..., RGRG... GBGB..., etc. Below is an example of a small V4L2_PIX_FMT_SBGGR12 image:

Byte Order. Each cell is one byte, the 4 most significant bits in the high bytes are 0.

start + 0:	B _{00low}	B _{00high}	G _{01low}	G _{01high}	B _{02low}	B _{02high}	G _{03low}	G _{03high}
start + 8:	G _{10low}	G _{10high}	R _{11low}	R _{11high}	G _{12low}	G _{12high}	R _{13low}	R _{13high}
start + 16:	B _{20low}	B _{20high}	G _{21low}	G _{21high}	B _{22low}	B _{22high}	G _{23low}	G _{23high}
start + 24:	G _{30low}	G _{30high}	R _{31low}	R _{31high}	G _{32low}	G _{32high}	R _{33low}	R _{33high}

V4L2_PIX_FMT_SRGGB12P ('pRCC'), **V4L2_PIX_FMT_SGRBG12P** ('pgCC'),
V4L2_PIX_FMT_SGBRG12P ('pGCC'), **V4L2_PIX_FMT_SBGGR12P** ('pBCC'),

12-bit packed Bayer formats

Description

These four pixel formats are packed raw sRGB / Bayer formats with 12 bits per colour. Every two consecutive samples are packed into three bytes. Each of the first two bytes contain the 8 high order bits of the pixels, and the third byte contains the four least significant bits of each pixel, in the same order.

Each n-pixel row contains n/2 green samples and n/2 blue or red samples, with alternating green-red and green-blue rows. They are conventionally described as GRGR... BGBG..., RGRG... GBGB..., etc. Below is an example of a small V4L2_PIX_FMT_SBGGR12P image:

Byte Order. Each cell is one byte.

start + 0:	B _{00high}	G _{01high}	G _{01low} (bits 7-4) B _{00low} (bits 3-0)	B _{02high}	G _{03high}	G _{03low} (bits 7-4) B _{02low} (bits 3-0)
start + 6:	G _{10high}	R _{11high}	R _{11low} (bits 7-4) G _{10low} (bits 3-0)	G _{12high}	R _{13high}	R _{13low} (bits 3-2) G _{12low} (bits 3-0)
start + 12:	B _{20high}	G _{21high}	G _{21low} (bits 7-4) B _{20low} (bits 3-0)	B _{22high}	G _{23high}	G _{23low} (bits 7-4) B _{22low} (bits 3-0)
start + 18:	G _{30high}	R _{31high}	R _{31low} (bits 7-4) G _{30low} (bits 3-0)	G _{32high}	R _{33high}	R _{33low} (bits 3-2) G _{32low} (bits 3-0)

V4L2_PIX_FMT_SRGGB14 ('RG14'), **V4L2_PIX_FMT_SGRBG14** ('GR14'),
V4L2_PIX_FMT_SGBRG14 ('GB14'), **V4L2_PIX_FMT_SBGGR14** ('BG14'),

14-bit Bayer formats expanded to 16 bits

Description

These four pixel formats are raw sRGB / Bayer formats with 14 bits per colour. Each sample is stored in a 16-bit word, with two unused high bits filled with zeros. Each n-pixel row contains n/2 green samples and n/2 blue or red samples, with alternating red and blue rows. Bytes are stored in memory in little endian order. They are conventionally described as GRGR... BGBG..., RGRG... GBGB..., etc. Below is an example of a small V4L2_PIX_FMT_SBGGR14 image:

Byte Order. Each cell is one byte, the two most significant bits in the high bytes are zero.

start + 0:	B _{00low}	B _{00high}	G _{01low}	G _{01high}	B _{02low}	B _{02high}	G _{03low}	G _{03high}
start + 8:	G _{10low}	G _{10high}	R _{11low}	R _{11high}	G _{12low}	G _{12high}	R _{13low}	R _{13high}
start + 16:	B _{20low}	B _{20high}	G _{21low}	G _{21high}	B _{22low}	B _{22high}	G _{23low}	G _{23high}
start + 24:	G _{30low}	G _{30high}	R _{31low}	R _{31high}	G _{32low}	G _{32high}	R _{33low}	R _{33high}

V4L2_PIX_FMT_SRGG14P ('pREE'), **V4L2_PIX_FMT_SGRBG14P** ('pgEE'),
V4L2_PIX_FMT_SGBRG14P ('pGEE'), **V4L2_PIX_FMT_SBGGR14P** ('pBEE'),

man V4L2_PIX_FMT_SRGG14P(2)

V4L2_PIX_FMT_SGRBG14P V4L2_PIX_FMT_SGBRG14P V4L2_PIX_FMT_SBGGR14P 14-bit packed Bayer formats

Description

These four pixel formats are packed raw sRGB / Bayer formats with 14 bits per colour. Every four consecutive samples are packed into seven bytes. Each of the first four bytes contain the eight high order bits of the pixels, and the three following bytes contains the six least significant bits of each pixel, in the same order.

Each n-pixel row contains n/2 green samples and n/2 blue or red samples, with alternating green-red and green-blue rows. They are conventionally described as GRGR... BGBG..., RGRG... GBGB..., etc. Below is an example of one of these formats:

Byte Order. Each cell is one byte.

start + 0	B00high	G01high	B02high	G03high	G01low bits 1-0 (bits 7-6) B00low bits 5-0 (bits 5-0)	B02low bits 3-0 (bits 7-4) G01low bits 5-2 (bits 3-0)	G03low bits 5-0 (bits 7-2) B02low bits 5-4 (bits 1-0)
start + 7	G10high	R11high	G12high	R13high	R11low bits 1-0 (bits 7-6) G10low bits 5-0 (bits 5-0)	G12low bits 3-0 (bits 7-4) R11low bits 5-2 (bits 3-0)	R13low bits 5-0 (bits 7-2) G12low bits 5-4 (bits 1-0)
start + 14	B20high	G21high	B22high	G23high	G21low bits 1-0 (bits 7-6) B20low bits 5-0 (bits 5-0)	B22low bits 3-0 (bits 7-4) G21low bits 5-2 (bits 3-0)	G23low bits 5-0 (bits 7-2) B22low bits 5-4 (bits 1-0)
start + 21	G30high	R31high	G32high	R33high	R31low bits 1-0 (bits 7-6) G30low bits 5-0 (bits 5-0)	G32low bits 3-0 (bits 7-4) R31low bits 5-2 (bits 3-0)	R33low bits 5-0 (bits 7-2) G32low bits 5-4 (bits 1-0)

V4L2_PIX_FMT_SRGG16 ('RG16'), **V4L2_PIX_FMT_SGRBG16** ('GR16'),
V4L2_PIX_FMT_SGBRG16 ('GB16'), **V4L2_PIX_FMT_SBGGR16** ('BYR2'),

16-bit Bayer formats

Description

These four pixel formats are raw sRGB / Bayer formats with 16 bits per sample. Each sample is stored in a 16-bit word. Each n-pixel row contains n/2 green samples and n/2 blue or red samples, with alternating red and blue rows. Bytes are stored in memory in little endian order. They are conventionally described as GRGR... BGBG..., RGRG... GBGB..., etc. Below is an example of a small V4L2_PIX_FMT_SBGGR16 image:

Byte Order. Each cell is one byte.

start + 0:	B00low	B00high	G01low	G01high	B02low	B02high	G03low	G03high
start + 8:	G10low	G10high	R11low	R11high	G12low	G12high	R13low	R13high
start + 16:	B20low	B20high	G21low	G21high	B22low	B22high	G23low	G23high
start + 24:	G30low	G30high	R31low	R31high	G32low	G32high	R33low	R33high

3.2.2.7 YUV Formats

YUV is the format native to TV broadcast and composite video signals. It separates the brightness information (Y) from the color information (U and V or Cb and Cr). The color information consists of red and blue *color difference* signals, this way the green component can be reconstructed by subtracting from the brightness component. See *Colorspaces* for conversion examples. YUV was chosen because early television would only transmit brightness information. To add color in a way compatible with existing receivers a new signal carrier was added to transmit the color difference signals.

Subsampling

YUV formats commonly encode images with a lower resolution for the chroma components than for the luma component. This compression technique, taking advantage of the human eye being more sensitive to luminance than color differences, is called chroma subsampling.

While many combinations of subsampling factors in the horizontal and vertical direction are possible, common factors are 1 (no subsampling), 2 and 4, with horizontal subsampling always larger than or equal to vertical subsampling. Common combinations are named as follows.

- 4:4:4: No subsampling
- 4:2:2: Horizontal subsampling by 2, no vertical subsampling
- 4:2:0: Horizontal subsampling by 2, vertical subsampling by 2
- 4:1:1: Horizontal subsampling by 4, no vertical subsampling
- 4:1:0: Horizontal subsampling by 4, vertical subsampling by 4

Subsampling the chroma component effectively creates chroma values that can be located in different spatial locations:

- The subsampled chroma value may be calculated by simply averaging the chroma value of two consecutive pixels. It effectively models the chroma of a pixel sited between the two original pixels. This is referred to as centered or interstitially sited chroma.
- The other option is to subsample chroma values in a way that place them in the same spatial sites as the pixels. This may be performed by skipping every other chroma sample (creating aliasing artifacts), or with filters using an odd number of taps. This is referred to as co-sited chroma.

The following examples show different combination of chroma siting in a 4x4 image.

Table 53: 4:2:2 subsampling, interstitially sited

	0		1		2		3
0	Y	C	Y		Y	C	Y
1	Y	C	Y		Y	C	Y
2	Y	C	Y		Y	C	Y
3	Y	C	Y		Y	C	Y

Table 54: 4:2:2 subsampling, co-sited

	0		1		2		3
0	Y/C		Y		Y/C		Y
1	Y/C		Y		Y/C		Y
2	Y/C		Y		Y/C		Y
3	Y/C		Y		Y/C		Y

Table 55: 4:2:0 subsampling, horizontally interstitially sited, vertically co-sited

	0		1		2		3
0	Y	C	Y		Y	C	Y
1	Y		Y		Y		Y
2	Y	C	Y		Y	C	Y
3	Y		Y		Y		Y

Table 56: 4:1:0 subsampling, horizontally and vertically interstitially sited

	0		1		2		3
0	Y		Y		Y		Y
1	Y		Y		Y		Y
				C			
2	Y		Y		Y		Y
3	Y		Y		Y		Y

Packed YUV formats

Similarly to the packed RGB formats, the packed YUV formats store the Y, Cb and Cr components consecutively in memory. They may apply subsampling to the chroma components and thus differ in how they interlave the three components.

Note:

- In all the tables that follow, bit 7 is the most significant bit in a byte.
 - ‘Y’, ‘Cb’ and ‘Cr’ denote bits of the luma, blue chroma (also known as ‘U’) and red chroma (also known as ‘V’) components respectively. ‘A’ denotes bits of the alpha component (if supported by the format), and ‘X’ denotes padding bits.
-

4:4:4 Subsampling

These formats do not subsample the chroma components and store each pixels as a full triplet of Y, Cb and Cr values.

The next table lists the packed YUV 4:4:4 formats with less than 8 bits per component. They are named based on the order of the Y, Cb and Cr components as seen in a 16-bit word, which is then stored in memory in little endian byte order, and on the number of bits for each component. For instance the YUV565 format stores a pixel in a 16-bit word [15:0] laid out as [Y'₄₋₀ Cb₅₋₀ Cr₄₋₀], and stored in memory in two bytes, [Cb₂₋₀ Cr₄₋₀] followed by [Y'₄₋₀ Cb₅₋₃].

Table 57: Packed YUV 4:4:4 Image Formats (less than 8bpc)

Identifier	Code	Byte 0 in memory								Byte 1							
		7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
V4L2_PIX_FMT_YUV444	'Y444'	Cb ₃	Cb ₂	Cb ₁	Cb ₀	Cr ₃	Cr ₂	Cr ₁	Cr ₀	a ₃	a ₂	a ₁	a ₀	Y' ₃	Y' ₂	Y' ₁	Y' ₀
V4L2_PIX_FMT_YUV555	'YUVO'	Cb ₂	Cb ₁	Cb ₀	Cr ₄	Cr ₃	Cr ₂	Cr ₁	Cr ₀	a	Y' ₄	Y' ₃	Y' ₂	Y' ₁	Y' ₀	Cb ₄	Cb ₃
V4L2_PIX_FMT_YUV565	'YUVP'	Cb ₂	Cb ₁	Cb ₀	Cr ₄	Cr ₃	Cr ₂	Cr ₁	Cr ₀	Y' ₄	Y' ₃	Y' ₂	Y' ₁	Y' ₀	Cb ₅	Cb ₄	Cb ₃

Note: For the YUV444 and YUV555 formats, the value of alpha bits is undefined when reading from the driver, ignored when writing to the driver, except when alpha blending has been negotiated for a *Video Overlay* or *Video Output Overlay*.

The next table lists the packed YUV 4:4:4 formats with 8 bits per component. They are named based on the order of the Y, Cb and Cr components as stored in memory, and on the total number of bits per pixel. For instance, the VUYX32 format stores a pixel with Cr₇₋₀ in the first byte, Cb₇₋₀ in the second byte and Y'₇₋₀ in the third byte.

Table 58: Packed YUV Image Formats (8bpc)

Identifier	Code	Byte 0	Byte 1	Byte 2	Byte 3
V4L2_PIX_FMT_YUV32	'YUV4'	A ₇₋₀	Y' ₇₋₀	Cb ₇₋₀	Cr ₇₋₀
V4L2_PIX_FMT_AYUV32	'AYUV'	A ₇₋₀	Y' ₇₋₀	Cb ₇₋₀	Cr ₇₋₀
V4L2_PIX_FMT_XYUV32	'XYUV'	X ₇₋₀	Y' ₇₋₀	Cb ₇₋₀	Cr ₇₋₀
V4L2_PIX_FMT_VUYA32	'VUYA'	Cr ₇₋₀	Cb ₇₋₀	Y' ₇₋₀	A ₇₋₀
V4L2_PIX_FMT_VUYX32	'VUYX'	Cr ₇₋₀	Cb ₇₋₀	Y' ₇₋₀	X ₇₋₀
V4L2_PIX_FMT_YUV24	'YUV3'	Y' ₇₋₀	Cb ₇₋₀	Cr ₇₋₀	-

Note:

- The alpha component is expected to contain a meaningful value that can be used by drivers and applications.
- The padding bits contain undefined values that must be ignored by all applications and drivers.

4:2:2 Subsampling

These formats, commonly referred to as YUYV or YUY2, subsample the chroma components horizontally by 2, storing 2 pixels in 4 bytes.

Table 59: Packed YUV 4:2:2 Formats

Identifier	Code	Byte 0	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7
V4L2_PIX_FMT_UYVY	'UYVY'	Cb ₀	Y' ₀	Cr ₀	Y' ₁	Cb ₂	Y' ₂	Cr ₂	Y' ₃
V4L2_PIX_FMT_VYUY	'VYUY'	Cr ₀	Y' ₀	Cb ₀	Y' ₁	Cr ₂	Y' ₂	Cb ₂	Y' ₃
V4L2_PIX_FMT_YUYV	'YUYV'	Y' ₀	Cb ₀	Y' ₁	Cr ₀	Y' ₂	Cb ₂	Y' ₃	Cr ₂
V4L2_PIX_FMT_YVYU	'YVYU'	Y' ₀	Cr ₀	Y' ₁	Cb ₀	Y' ₂	Cr ₂	Y' ₃	Cb ₂

Color Sample Location: Chroma samples are *interstitially sited* horizontally.

4:1:1 Subsampling

This format subsamples the chroma components horizontally by 4, storing 8 pixels in 12 bytes.

Table 60: Packed YUV 4:1:1 Formats

Identifier	Code	Byte 0	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7	Byte 8	Byte 9	Byte 10	Byte 11
V4L2_PIX_FMT_Y41P	'Y41P'	Cb ₀	Y' ₀	Cr ₀	Y' ₁	Cb ₄	Y' ₂	Cr ₄	Y' ₃	Y' ₄	Y' ₅	Y' ₆	Y' ₇

Note: Do not confuse V4L2_PIX_FMT_Y41P with *V4L2_PIX_FMT_YUV411P*. Y41P is derived from “YUV 4:1:1 **packed**”, while YUV411P stands for “YUV 4:1:1 **planar**”.

Color Sample Location: Chroma samples are *interstitially sited* horizontally.

Planar YUV formats

Planar formats split luma and chroma data in separate memory regions. They exist in two variants:

- Semi-planar formats use two planes. The first plane is the luma plane and stores the Y components. The second plane is the chroma plane and stores the Cb and Cr components interleaved.
- Fully planar formats use three planes to store the Y, Cb and Cr components separately.

Within a plane, components are stored in pixel order, which may be linear or tiled. Padding may be supported at the end of the lines, and the line stride of the chroma planes may be constrained by the line stride of the luma plane.

Some planar formats allow planes to be placed in independent memory locations. They are identified by an ‘M’ suffix in their name (such as in V4L2_PIX_FMT_NV12M). Those formats are intended to be used only in drivers and applications that support the multi-planar API, described

in *Single- and multi-planar APIs*. Unless explicitly documented as supporting non-contiguous planes, formats require the planes to follow each other immediately in memory.

Semi-Planar YUV Formats

These formats are commonly referred to as NV formats (NV12, NV16, ...). They use two planes, and store the luma components in the first plane and the chroma components in the second plane. The Cb and Cr components are interleaved in the chroma plane, with Cb and Cr always stored in pairs. The chroma order is exposed as different formats.

For memory contiguous formats, the number of padding pixels at the end of the chroma lines is identical to the padding of the luma lines. Without horizontal subsampling, the chroma line stride (in bytes) is thus equal to twice the luma line stride. With horizontal subsampling by 2, the chroma line stride is equal to the luma line stride. Vertical subsampling doesn't affect the line stride.

For non-contiguous formats, no constraints are enforced by the format on the relationship between the luma and chroma line padding and stride.

All components are stored with the same number of bits per component.

Table 61: Overview of Semi-Planar YUV Formats

Identifier	Code	Bits per component	Subsampling	Chroma order ¹	Contiguous ²	Tiling ³
V4L2_PIX_FMT_NV12	'NV12'	8	4:2:0	Cb, Cr	Yes	Linear
V4L2_PIX_FMT_NV21	'NV21'	8	4:2:0	Cr, Cr	Yes	Linear
V4L2_PIX_FMT_NV12M	'NM12'	8	4:2:0	Cb, Cr	No	Linear
V4L2_PIX_FMT_NV21M	'NM21'	8	4:2:0	Cr, Cr	No	Linear
V4L2_PIX_FMT_NV12MT	'TM12'	8	4:2:0	Cb, Cr	No	64x32 tiles Horizontal Z order
V4L2_PIX_FMT_NV12MT_16X16	'VM12'	8	4:2:2	Cb, Cr	No	16x16 tiles
V4L2_PIX_FMT_NV16	'NV16'	8	4:2:2	Cb, Cr	Yes	Linear
V4L2_PIX_FMT_NV61	'NV61'	8	4:2:2	Cr, Cr	Yes	Linear
V4L2_PIX_FMT_NV16M	'NM16'	8	4:2:2	Cb, Cr	No	Linear
V4L2_PIX_FMT_NV61M	'NM61'	8	4:2:2	Cr, Cr	No	Linear
V4L2_PIX_FMT_NV24	'NV24'	8	4:4:4	Cb, Cr	Yes	Linear
V4L2_PIX_FMT_NV42	'NV42'	8	4:4:4	Cr, Cr	Yes	Linear

Color Sample Location: Chroma samples are *interstitially sited* horizontally.

¹ Order of chroma samples in the second plane

² Indicates if planes have to be contiguous in memory or can be disjoint

³ Macroblock size in pixels

NV12, NV21, NV12M and NV21M

Semi-planar YUV 4:2:0 formats. The chroma plane is subsampled by 2 in each direction. Chroma lines contain half the number of pixels and the same number of bytes as luma lines, and the chroma plane contains half the number of lines of the luma plane.

Table 62: Sample 4x4 NV12 Image

start + 0:	Y' ₀₀	Y' ₀₁	Y' ₀₂	Y' ₀₃
start + 4:	Y' ₁₀	Y' ₁₁	Y' ₁₂	Y' ₁₃
start + 8:	Y' ₂₀	Y' ₂₁	Y' ₂₂	Y' ₂₃
start + 12:	Y' ₃₀	Y' ₃₁	Y' ₃₂	Y' ₃₃
start + 16:	Cb ₀₀	Cr ₀₀	Cb ₀₁	Cr ₀₁
start + 20:	Cb ₁₀	Cr ₁₀	Cb ₁₁	Cr ₁₁

Table 63: Sample 4x4 NV12M Image

start0 + 0:	Y' ₀₀	Y' ₀₁	Y' ₀₂	Y' ₀₃
start0 + 4:	Y' ₁₀	Y' ₁₁	Y' ₁₂	Y' ₁₃
start0 + 8:	Y' ₂₀	Y' ₂₁	Y' ₂₂	Y' ₂₃
start0 + 12:	Y' ₃₀	Y' ₃₁	Y' ₃₂	Y' ₃₃
start1 + 0:	Cb ₀₀	Cr ₀₀	Cb ₀₁	Cr ₀₁
start1 + 4:	Cb ₁₀	Cr ₁₀	Cb ₁₁	Cr ₁₁

Tiled NV12

Semi-planar YUV 4:2:0 formats, using macroblock tiling. The chroma plane is subsampled by 2 in each direction. Chroma lines contain half the number of pixels and the same number of bytes as luma lines, and the chroma plane contains half the number of lines of the luma plane. Each tile follows the previous one linearly in memory (from left to right, top to bottom).

V4L2_PIX_FMT_NV12MT_16X16 is similar to V4L2_PIX_FMT_NV12M but stores pixels in 2D 16x16 tiles, and stores tiles linearly in memory. The line stride and image height must be aligned to a multiple of 16. The layouts of the luma and chroma planes are identical.

V4L2_PIX_FMT_NV12MT is similar to V4L2_PIX_FMT_NV12M but stores pixels in 2D 64x32 tiles, and stores 2x2 groups of tiles in Z-order in memory, alternating Z and mirrored Z shapes horizontally. The line stride must be a multiple of 128 pixels to ensure an integer number of Z shapes. The image height must be a multiple of 32 pixels. If the vertical resolution is an odd number of tiles, the last row of tiles is stored in linear order. The layouts of the luma and chroma planes are identical.

V4L2_PIX_FMT_NV12_4L4 stores pixel in 4x4 tiles, and stores tiles linearly in memory. The line stride and image height must be aligned to a multiple of 4. The layouts of the luma and chroma planes are identical.

V4L2_PIX_FMT_NV12_16L16 stores pixel in 16x16 tiles, and stores tiles linearly in memory. The line stride and image height must be aligned to a multiple of 16. The layouts of the luma and chroma planes are identical.

V4L2_PIX_FMT_NV12_32L32 stores pixel in 32x32 tiles, and stores tiles linearly in memory. The

line stride and image height must be aligned to a multiple of 32. The layouts of the luma and chroma planes are identical.

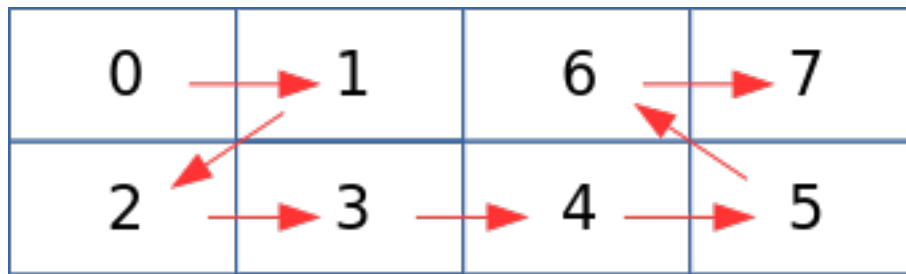


Fig. 4: V4L2_PIX_FMT_NV12MT macroblock Z shape memory layout

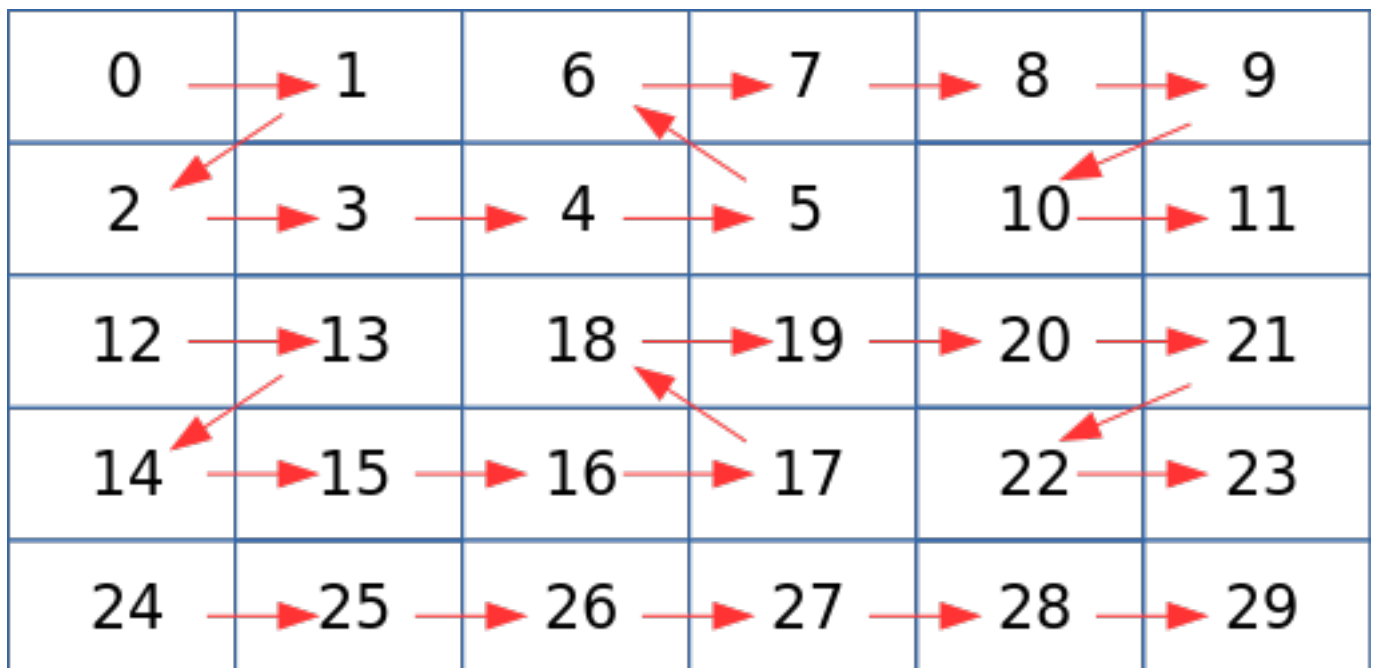


Fig. 5: Example V4L2_PIX_FMT_NV12MT memory layout of tiles

NV16, NV61, NV16M and NV61M

Semi-planar YUV 4:2:2 formats. The chroma plane is subsampled by 2 in the horizontal direction. Chroma lines contain half the number of pixels and the same number of bytes as luma lines, and the chroma plane contains the same number of lines as the luma plane.

Table 64: Sample 4x4 NV16 Image

start + 0:	Y' ₀₀	Y' ₀₁	Y' ₀₂	Y' ₀₃
start + 4:	Y' ₁₀	Y' ₁₁	Y' ₁₂	Y' ₁₃
start + 8:	Y' ₂₀	Y' ₂₁	Y' ₂₂	Y' ₂₃
start + 12:	Y' ₃₀	Y' ₃₁	Y' ₃₂	Y' ₃₃
start + 16:	Cb ₀₀	Cr ₀₀	Cb ₀₁	Cr ₀₁
start + 20:	Cb ₁₀	Cr ₁₀	Cb ₁₁	Cr ₁₁
start + 24:	Cb ₂₀	Cr ₂₀	Cb ₂₁	Cr ₂₁
start + 28:	Cb ₃₀	Cr ₃₀	Cb ₃₁	Cr ₃₁

Table 65: Sample 4x4 NV16M Image

start0 + 0:	Y'00	Y'01	Y'02	Y'03
start0 + 4:	Y'10	Y'11	Y'12	Y'13
start0 + 8:	Y'20	Y'21	Y'22	Y'23
start0 + 12:	Y'30	Y'31	Y'32	Y'33
start1 + 0:	Cb00	Cr00	Cb02	Cr02
start1 + 4:	Cb10	Cr10	Cb12	Cr12
start1 + 8:	Cb20	Cr20	Cb22	Cr22
start1 + 12:	Cb30	Cr30	Cb32	Cr32

NV24 and NV42

Semi-planar YUV 4:4:4 formats. The chroma plane is not subsampled. Chroma lines contain the same number of pixels and twice the number of bytes as luma lines, and the chroma plane contains the same number of lines as the luma plane.

Table 66: Sample 4x4 NV24 Image

start + 0:	Y'00	Y'01	Y'02	Y'03				
start + 4:	Y'10	Y'11	Y'12	Y'13				
start + 8:	Y'20	Y'21	Y'22	Y'23				
start + 12:	Y'30	Y'31	Y'32	Y'33				
start + 16:	Cb00	Cr00	Cb01	Cr01	Cb02	Cr02	Cb03	Cr03
start + 24:	Cb10	Cr10	Cb11	Cr11	Cb12	Cr12	Cb13	Cr13
start + 32:	Cb20	Cr20	Cb21	Cr21	Cb22	Cr22	Cb23	Cr23
start + 40:	Cb30	Cr30	Cb31	Cr31	Cb32	Cr32	Cb33	Cr33

Fully Planar YUV Formats

These formats store the Y, Cb and Cr components in three separate planes. The luma plane comes first, and the order of the two chroma planes varies between formats. The two chroma planes always use the same subsampling.

For memory contiguous formats, the number of padding pixels at the end of the chroma lines is identical to the padding of the luma lines. The chroma line stride (in bytes) is thus equal to the luma line stride divided by the horizontal subsampling factor. Vertical subsampling doesn't affect the line stride.

For non-contiguous formats, no constraints are enforced by the format on the relationship between the luma and chroma line padding and stride.

All components are stored with the same number of bits per component.

Table 67: Overview of Fully Planar YUV Formats

Identifier	Code	Bits per component	Subsampling	Planes order ⁴	Contiguous ⁵
V4L2_PIX_FMT_YUV410	'YUV9'	8	4:1:0	Y, Cb, Cr	Yes
V4L2_PIX_FMT_YVU410	'YVU9'	8	4:1:0	Y, Cr, Cb	Yes
V4L2_PIX_FMT_YUV411P	'411P'	8	4:1:1	Y, Cb, Cr	Yes
V4L2_PIX_FMT_YUV420M	'YM12'	8	4:2:0	Y, Cb, Cr	No
V4L2_PIX_FMT_YVU420M	'YM21'	8	4:2:0	Y, Cr, Cb	No
V4L2_PIX_FMT_YUV420	'YU12'	8	4:2:0	Y, Cb, Cr	Yes
V4L2_PIX_FMT_YVU420	'YV12'	8	4:2:0	Y, Cr, Cb	Yes
V4L2_PIX_FMT_YUV422P	'422P'	8	4:2:2	Y, Cb, Cr	Yes
V4L2_PIX_FMT_YUV422M	'YM16'	8	4:2:2	Y, Cb, Cr	No
V4L2_PIX_FMT_YVU422M	'YM61'	8	4:2:2	Y, Cr, Cb	No
V4L2_PIX_FMT_YUV444M	'YM24'	8	4:4:4	Y, Cb, Cr	No
V4L2_PIX_FMT_YVU444M	'YM42'	8	4:4:4	Y, Cr, Cb	No

Color Sample Location: Chroma samples are *interstitially sited* horizontally.

YUV410 and YVU410

Planar YUV 4:1:0 formats. The chroma planes are subsampled by 4 in each direction. Chroma lines contain a quarter of the number of pixels and bytes of the luma lines, and the chroma planes contain a quarter of the number of lines of the luma plane.

Table 68: Sample 4x4 YUV410 Image

start + 0:	Y'00	Y'01	Y'02	Y'03
start + 4:	Y'10	Y'11	Y'12	Y'13
start + 8:	Y'20	Y'21	Y'22	Y'23
start + 12:	Y'30	Y'31	Y'32	Y'33
start + 16:	Cr00			
start + 17:	Cb00			

⁴ Order of luma and chroma planes

⁵ Indicates if planes have to be contiguous in memory or can be disjoint

YUV411P

Planar YUV 4:1:1 formats. The chroma planes are subsampled by 4 in the horizontal direction. Chroma lines contain a quarter of the number of pixels and bytes of the luma lines, and the chroma planes contain the same number of lines as the luma plane.

Table 69: Sample 4x4 YUV411P Image

start + 0:	Y' ₀₀	Y' ₀₁	Y' ₀₂	Y' ₀₃
start + 4:	Y' ₁₀	Y' ₁₁	Y' ₁₂	Y' ₁₃
start + 8:	Y' ₂₀	Y' ₂₁	Y' ₂₂	Y' ₂₃
start + 12:	Y' ₃₀	Y' ₃₁	Y' ₃₂	Y' ₃₃
start + 16:	Cb ₀₀			
start + 17:	Cb ₁₀			
start + 18:	Cb ₂₀			
start + 19:	Cb ₃₀			
start + 20:	Cr ₀₀			
start + 21:	Cr ₁₀			
start + 22:	Cr ₂₀			
start + 23:	Cr ₃₀			

YUV420, YVU420, YUV420M and YVU420M

Planar YUV 4:2:0 formats. The chroma planes are subsampled by 2 in each direction. Chroma lines contain half of the number of pixels and bytes of the luma lines, and the chroma planes contain half of the number of lines of the luma plane.

Table 70: Sample 4x4 YUV420 Image

start + 0:	Y' ₀₀	Y' ₀₁	Y' ₀₂	Y' ₀₃
start + 4:	Y' ₁₀	Y' ₁₁	Y' ₁₂	Y' ₁₃
start + 8:	Y' ₂₀	Y' ₂₁	Y' ₂₂	Y' ₂₃
start + 12:	Y' ₃₀	Y' ₃₁	Y' ₃₂	Y' ₃₃
start + 16:	Cr ₀₀	Cr ₀₁		
start + 18:	Cr ₁₀	Cr ₁₁		
start + 20:	Cb ₀₀	Cb ₀₁		
start + 22:	Cb ₁₀	Cb ₁₁		

Table 71: Sample 4x4 YUV420M Image

start0 + 0:	Y' ₀₀	Y' ₀₁	Y' ₀₂	Y' ₀₃
start0 + 4:	Y' ₁₀	Y' ₁₁	Y' ₁₂	Y' ₁₃
start0 + 8:	Y' ₂₀	Y' ₂₁	Y' ₂₂	Y' ₂₃
start0 + 12:	Y' ₃₀	Y' ₃₁	Y' ₃₂	Y' ₃₃
start1 + 0:	Cb ₀₀	Cb ₀₁		
start1 + 2:	Cb ₁₀	Cb ₁₁		
start2 + 0:	Cr ₀₀	Cr ₀₁		
start2 + 2:	Cr ₁₀	Cr ₁₁		

YUV422P, YUV422M and YVU422M

Planar YUV 4:2:2 formats. The chroma planes are subsampled by 2 in the horizontal direction. Chroma lines contain half of the number of pixels and bytes of the luma lines, and the chroma planes contain the same number of lines as the luma plane.

Table 72: Sample 4x4 YUV422P Image

start + 0:	Y' ₀₀	Y' ₀₁	Y' ₀₂	Y' ₀₃
start + 4:	Y' ₁₀	Y' ₁₁	Y' ₁₂	Y' ₁₃
start + 8:	Y' ₂₀	Y' ₂₁	Y' ₂₂	Y' ₂₃
start + 12:	Y' ₃₀	Y' ₃₁	Y' ₃₂	Y' ₃₃
start + 16:	Cb ₀₀	Cb ₀₁		
start + 18:	Cb ₁₀	Cb ₁₁		
start + 20:	Cb ₂₀	Cb ₂₁		
start + 22:	Cb ₃₀	Cb ₃₁		
start + 24:	Cr ₀₀	Cr ₀₁		
start + 26:	Cr ₁₀	Cr ₁₁		
start + 28:	Cr ₂₀	Cr ₂₁		
start + 30:	Cr ₃₀	Cr ₃₁		

Table 73: Sample 4x4 YUV422M Image

start0 + 0:	Y' ₀₀	Y' ₀₁	Y' ₀₂	Y' ₀₃
start0 + 4:	Y' ₁₀	Y' ₁₁	Y' ₁₂	Y' ₁₃
start0 + 8:	Y' ₂₀	Y' ₂₁	Y' ₂₂	Y' ₂₃
start0 + 12:	Y' ₃₀	Y' ₃₁	Y' ₃₂	Y' ₃₃
start1 + 0:	Cb ₀₀	Cb ₀₁		
start1 + 2:	Cb ₁₀	Cb ₁₁		
start1 + 4:	Cb ₂₀	Cb ₂₁		
start1 + 6:	Cb ₃₀	Cb ₃₁		
start2 + 0:	Cr ₀₀	Cr ₀₁		
start2 + 2:	Cr ₁₀	Cr ₁₁		
start2 + 4:	Cr ₂₀	Cr ₂₁		
start2 + 6:	Cr ₃₀	Cr ₃₁		

YUV444M and YVU444M

Planar YUV 4:4:4 formats. The chroma planes are not subsampled. Chroma lines contain the same number of pixels and bytes of the luma lines, and the chroma planes contain the same number of lines as the luma plane.

Table 74: Sample 4x4 YUV444M Image

start0 + 0:	Y'00	Y'01	Y'02	Y'03
start0 + 4:	Y'10	Y'11	Y'12	Y'13
start0 + 8:	Y'20	Y'21	Y'22	Y'23
start0 + 12:	Y'30	Y'31	Y'32	Y'33
start1 + 0:	Cb00	Cb01	Cb02	Cb03
start1 + 4:	Cb10	Cb11	Cb12	Cb13
start1 + 8:	Cb20	Cb21	Cb22	Cb23
start1 + 12:	Cb20	Cb21	Cb32	Cb33
start2 + 0:	Cr00	Cr01	Cr02	Cr03
start2 + 4:	Cr10	Cr11	Cr12	Cr13
start2 + 8:	Cr20	Cr21	Cr22	Cr23
start2 + 12:	Cr30	Cr31	Cr32	Cr33

Luma-Only Formats

This family of formats only store the luma component of a Y'CbCr image. They are often referred to as greyscale formats.

Note:

- In all the tables that follow, bit 7 is the most significant bit in a byte.
 - Formats are described with the minimum number of pixels needed to create a byte-aligned repeating pattern. ... indicates repetition of the pattern.
 - $Y'_x[9:2]$ denotes bits 9 to 2 of the Y' value for pixel at column x .
 - 0 denotes padding bits set to 0.
-

Table 75: Luma-Only Image Formats

Identifier	Code	Byte 0	Byte 1	Byte 2	Byte 3	Byte 4
V4L2_PIX_FMT_GREY	'GREY'	Y ₀ [7:0]
V4L2_PIX_FMT_Y10	'Y10 '	Y ₀ [7:0]	000000 Y ₀ [9:8]
V4L2_PIX_FMT_Y10BPACK	'Y10B'	Y ₀ [9:2]	Y ₀ [1:0] Y ₁ [9:4]	Y ₁ [3:0] Y ₂ [9:6]	Y ₂ [5:0] Y ₃ [9:8]	Y ₃ [7:0]
V4L2_PIX_FMT_Y10P	'Y10P'	Y ₀ [7:0]	Y ₁ [9:8]	Y ₂ [9:2]	Y ₃ [9:2]	Y ₃ [1:0] Y ₂ [1:0] Y ₁ [1:0] Y ₀ [1:0]
V4L2_PIX_FMT_Y12	'Y12 '	Y ₀ [7:0]	0000 Y ₀ [11:8]
V4L2_PIX_FMT_Y14	'Y14 '	Y ₀ [7:0]	00 Y ₀ [13:8]
V4L2_PIX_FMT_Y16	'Y16 '	Y ₀ [7:0]	Y ₀ [15:8]
V4L2_PIX_FMT_Y16_BE	'Y16 ' (1U << 31)	Y ₀ [15:8]	Y ₀ [7:0]

Note: For the Y16 and Y16_BE formats, the actual sampling precision may be lower than 16 bits. For example, 10 bits per pixel uses values in the range 0 to 1023.

V4L2_PIX_FMT_Y8I ('Y8I ')

Interleaved grey-scale image, e.g. from a stereo-pair

Description

This is a grey-scale image with a depth of 8 bits per pixel, but with pixels from 2 sources interleaved. Each pixel is stored in a 16-bit word. E.g. the R200 RealSense camera stores pixel from the left sensor in lower and from the right sensor in the higher 8 bits.

Byte Order. Each cell is one byte.

start + 0:	Y' ₀₀ left	Y' ₀₀ right	Y' ₀₁ left	Y' ₀₁ right	Y' ₀₂ left	Y' ₀₂ right	Y' ₀₃ left	Y' ₀₃ right
start + 8:	Y' ₁₀ left	Y' ₁₀ right	Y' ₁₁ left	Y' ₁₁ right	Y' ₁₂ left	Y' ₁₂ right	Y' ₁₃ left	Y' ₁₃ right
start + 16:	Y' ₂₀ left	Y' ₂₀ right	Y' ₂₁ left	Y' ₂₁ right	Y' ₂₂ left	Y' ₂₂ right	Y' ₂₃ left	Y' ₂₃ right
start + 24:	Y' ₃₀ left	Y' ₃₀ right	Y' ₃₁ left	Y' ₃₁ right	Y' ₃₂ left	Y' ₃₂ right	Y' ₃₃ left	Y' ₃₃ right

V4L2_PIX_FMT_Y12I ('Y12I')

Interleaved grey-scale image, e.g. from a stereo-pair

Description

This is a grey-scale image with a depth of 12 bits per pixel, but with pixels from 2 sources interleaved and bit-packed. Each pixel is stored in a 24-bit word in the little-endian order. On a little-endian machine these pixels can be deinterlaced using

```
__u8 *buf;
left0 = 0xffff & *(__u16 *)buf;
right0 = *(__u16 *) (buf + 1) >> 4;
```

Bit-packed representation. pixels cross the byte boundary and have a ratio of 3 bytes for each interleaved pixel.

Y' _{0left} [7:0]	Y' _{0right} [3:0]	Y' _{0left} [11:8]	Y' _{0right} [11:4]
---------------------------	----------------------------	----------------------------	-----------------------------

V4L2_PIX_FMT_UV8 ('UV8')

UV plane interleaved

Description

In this format there is no Y plane, Only CbCr plane. ie (UV interleaved)

Byte Order. Each cell is one byte.

start + 0:	Cb ₀₀	Cr ₀₀	Cb ₀₁	Cr ₀₁
start + 4:	Cb ₁₀	Cr ₁₀	Cb ₁₁	Cr ₁₁
start + 8:	Cb ₂₀	Cr ₂₀	Cb ₂₁	Cr ₂₁
start + 12:	Cb ₃₀	Cr ₃₀	Cb ₃₁	Cr ₃₁

V4L2_PIX_FMT_M420 ('M420')

Format with ½ horizontal and vertical chroma resolution, also known as YUV 4:2:0. Hybrid plane line-interleaved layout.

Description

M420 is a YUV format with ½ horizontal and vertical chroma subsampling (YUV 4:2:0). Pixels are organized as interleaved luma and chroma planes. Two lines of luma data are followed by one line of chroma data.

The luma plane has one byte per pixel. The chroma plane contains interleaved CbCr pixels subsampled by ½ in the horizontal and vertical directions. Each CbCr pair belongs to four pixels. For example, Cb₀/Cr₀ belongs to Y'₀₀, Y'₀₁, Y'₁₀, Y'₁₁.

All line lengths are identical: if the Y lines include pad bytes so do the CbCr lines.

Byte Order. Each cell is one byte.

start + 0:	Y' ₀₀	Y' ₀₁	Y' ₀₂	Y' ₀₃
start + 4:	Y' ₁₀	Y' ₁₁	Y' ₁₂	Y' ₁₃
start + 8:	Cb ₀₀	Cr ₀₀	Cb ₀₁	Cr ₀₁
start + 16:	Y' ₂₀	Y' ₂₁	Y' ₂₂	Y' ₂₃
start + 20:	Y' ₃₀	Y' ₃₁	Y' ₃₂	Y' ₃₃
start + 24:	Cb ₁₀	Cr ₁₀	Cb ₁₁	Cr ₁₁

Color Sample Location: Chroma samples are *interstitially sited* horizontally and vertically.

3.2.2.8 HSV Formats

These formats store the color information of the image in a geometrical representation. The colors are mapped into a cylinder, where the angle is the HUE, the height is the VALUE and the distance to the center is the SATURATION. This is a very useful format for image segmentation algorithms.

Packed HSV formats

Description

The *hue* (h) is measured in degrees, the equivalence between degrees and LSBs depends on the hsv-encoding used, see *Colorspaces*. The *saturation* (s) and the *value* (v) are measured in percentage of the cylinder: 0 being the smallest value and 255 the maximum.

The values are packed in 24 or 32 bit formats.

Table 76: Packed HSV Image Formats

Identifier	Code	Byte 0 in memory								Byte 1								Byte 2								Byte 3							
		Bit 7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
V4L2_PIX_FMT_HSV32	'HSV4'									h ₇	h ₆	h ₅	h ₄	h ₃	h ₂	h ₁	h ₀	s ₇	s ₆	s ₅	s ₄	s ₃	s ₂	s ₁	s ₀	v ₇	v ₆	v ₅	v ₄	v ₃	v ₂	v ₁	v ₀
V4L2_PIX_FMT_HSV24	'HSV3'	h ₇	h ₆	h ₅	h ₄	h ₃	h ₂	h ₁	h ₀	s ₇	s ₆	s ₅	s ₄	s ₃	s ₂	s ₁	s ₀	v ₇	v ₆	v ₅	v ₄	v ₃	v ₂	v ₁	v ₀								

Bit 7 is the most significant bit.

3.2.2.9 Depth Formats

Depth data provides distance to points, mapped onto the image plane

V4L2_PIX_FMT_INZI ('INZI')

Infrared 10-bit linked with Depth 16-bit images

Description

Proprietary multi-planar format used by Intel SR300 Depth cameras, comprise of Infrared image followed by Depth data. The pixel definition is 32-bpp, with the Depth and Infrared Data split into separate continuous planes of identical dimensions.

The first plane - Infrared data - is stored according to *V4L2_PIX_FMT_Y10* greyscale format. Each pixel is 16-bit cell, with actual data stored in the 10 LSBs with values in range 0 to 1023. The six remaining MSBs are padded with zeros.

The second plane provides 16-bit per-pixel Depth data arranged in *V4L2_PIX_FMT_Z16* format.

Frame Structure. Each cell is a 16-bit word with more significant data stored at higher memory address (byte order is little-endian).

Ir _{0,0}	Ir _{0,1}	Ir _{0,2}
...					
Infrared Data					
...					
...	Ir _{n-1,n-3}	Ir _{n-1,n-2}	Ir _{n-1,n-1}
Depth _{0,0}	Depth _{0,1}	Depth _{0,2}
...					
Depth Data					
...					
...	Depth _{n-1,n-3}	Depth _{n-1,n-2}	Depth _{n-1,n-1}

V4L2_PIX_FMT_Z16 ('Z16')

16-bit depth data with distance values at each pixel

Description

This is a 16-bit format, representing depth data. Each pixel is a distance to the respective point in the image coordinates. Distance unit can vary and has to be negotiated with the device separately. Each pixel is stored in a 16-bit word in the little endian byte order.

Byte Order. Each cell is one byte.

start + 0:	Z _{00low}	Z _{00high}	Z _{01low}	Z _{01high}	Z _{02low}	Z _{02high}	Z _{03low}	Z _{03high}
start + 8:	Z _{10low}	Z _{10high}	Z _{11low}	Z _{11high}	Z _{12low}	Z _{12high}	Z _{13low}	Z _{13high}
start + 16:	Z _{20low}	Z _{20high}	Z _{21low}	Z _{21high}	Z _{22low}	Z _{22high}	Z _{23low}	Z _{23high}
start + 24:	Z _{30low}	Z _{30high}	Z _{31low}	Z _{31high}	Z _{32low}	Z _{32high}	Z _{33low}	Z _{33high}

V4L2_PIX_FMT_CNF4 ('CNF4')

Depth sensor confidence information as a 4 bits per pixel packed array

Description

Proprietary format used by Intel RealSense Depth cameras containing depth confidence information in range 0-15 with 0 indicating that the sensor was unable to resolve any signal and 15 indicating maximum level of confidence for the specific sensor (actual error margins might change from sensor to sensor).

Every two consecutive pixels are packed into a single byte. Bits 0-3 of byte n refer to confidence value of depth pixel $2*n$, bits 4-7 to confidence value of depth pixel $2*n+1$.

Bit-packed representation.

$Y'_{01[3:0]}(\text{bits } 7-4)$	$Y'_{00[3:0]}(\text{bits } 3-0)$	$Y'_{03[3:0]}(\text{bits } 7-4)$	$Y'_{02[3:0]}(\text{bits } 3-0)$
----------------------------------	----------------------------------	----------------------------------	----------------------------------

3.2.2.10 Compressed Formats

Table 77: Compressed Image Formats

Identifier	Code	Details
V4L2_PIX_FMT_JPEG	'JPEG'	TBD. See also VIDIOC_G_JPEGCOMP , VIDIOC_S_JPEGCOMP .
V4L2_PIX_FMT_MPEG	'MPEG'	MPEG multiplexed stream. The actual format is determined by extended control V4L2_CID_MPEG_STREAM_TYPE, see Codec Control IDs .
V4L2_PIX_FMT_H264	'H264'	H264 Access Unit. The decoder expects one Access Unit per buffer. The encoder generates one Access Unit per buffer. If ioctl VIDIOC_ENUM_FMT reports V4L2_FMT_FLAG_CONTINUOUS_BYTESTREAM then the decoder has no requirements since it can parse all the information from the raw bytestream.
V4L2_PIX_FMT_H264_NO_SC	'AVC1'	H264 video elementary stream without start codes.
V4L2_PIX_FMT_H264_MVC	'M264'	H264 MVC video elementary stream.

Continued on next page

Table 77 - continued from previous page

Identifier	Code	Details
V4L2_PIX_FMT_H264_SLICE	'S264'	H264 parsed slice data, including slice headers, either with or without the start code, as extracted from the H264 bitstream. This format is adapted for stateless video decoders that implement an H264 pipeline with the <i>Memory-to-memory Stateless Video Decoder Interface</i> . This pixel format has two modifiers that must be set at least once through the <code>V4L2_CID_STATELESS_H264_DECODE_MODE</code> and <code>V4L2_CID_STATELESS_H264_START_CODE</code> controls. In addition, metadata associated with the frame to decode are required to be passed through the <code>V4L2_CID_STATELESS_H264_SPS</code> , <code>V4L2_CID_STATELESS_H264_PPS</code> , <code>V4L2_CID_STATELESS_H264_SCALING_MATRIX</code> , <code>V4L2_CID_STATELESS_H264_SLICE_PARAMS</code> and <code>V4L2_CID_STATELESS_H264_DECODE_PARAMS</code> controls. See the <i>associated Codec Control IDs</i> . Exactly one output and one capture buffer must be provided for use with this pixel format. The output buffer must contain the appropriate number of macroblocks to decode a full corresponding frame to the matching capture buffer. The syntax for this format is documented in <i>ITU-T Rec. H.264 Specification (04/2017 Edition)</i> , section 7.3.2.8 "Slice layer without partitioning Rbsp syntax" and the following sections.
V4L2_PIX_FMT_H263	'H263'	H263 video elementary stream.
V4L2_PIX_FMT_MPEG1	'MPG1'	MPEG1 Picture. Each buffer starts with a Picture header, followed by other headers as needed and ending with the Picture data. If <code>ioctl VIDIOC_ENUM_FMT</code> reports <code>V4L2_FMT_FLAG_CONTINUOUS_BYTESTREAM</code> then the decoder has no requirements since it can parse all the information from the raw bytestream.
V4L2_PIX_FMT_MPEG2	'MPG2'	MPEG2 Picture. Each buffer starts with a Picture header, followed by other headers as needed and ending with the Picture data. If <code>ioctl VIDIOC_ENUM_FMT</code> reports <code>V4L2_FMT_FLAG_CONTINUOUS_BYTESTREAM</code> then the decoder has no requirements since it can parse all the information from the raw bytestream.
V4L2_PIX_FMT_MPEG2_SLICE	'MG2S'	MPEG-2 parsed slice data, as extracted from the MPEG-2 bitstream. This format is adapted for stateless video decoders that implement a MPEG-2 pipeline with the <i>Memory-to-memory Stateless Video Decoder Interface</i> . Metadata associated with the frame to decode is required to be passed through the <code>V4L2_CID_STATELESS_MPEG2_SEQUENCE</code> and <code>V4L2_CID_STATELESS_MPEG2_PICTURE</code> controls. Quantisation matrices can optionally be specified through the <code>V4L2_CID_STATELESS_MPEG2_QUANTISATION</code> control. See the <i>associated Codec Control IDs</i> . Exactly one output and one capture buffer must be provided for use with this pixel format. The output buffer must contain the appropriate number of macroblocks to decode a full corresponding frame to the matching capture buffer.
V4L2_PIX_FMT_MPEG4	'MPG4'	MPEG4 video elementary stream.

Continued on next page

Table 77 - continued from previous page

Identifier	Code	Details
V4L2_PIX_FMT_XVID	'XVID'	Xvid video elementary stream.
V4L2_PIX_FMT_VC1_ANNEX_G	'VC1G'	VC1, SMPTE 421M Annex G compliant stream.
V4L2_PIX_FMT_VC1_ANNEX_L	'VC1L'	VC1, SMPTE 421M Annex L compliant stream.
V4L2_PIX_FMT_VP8	'VP80'	VP8 compressed video frame. The encoder generates one compressed frame per buffer, and the decoder requires one compressed frame per buffer.
V4L2_PIX_FMT_VP8_FRAME	'VP8F'	VP8 parsed frame, including the frame header, as extracted from the container. This format is adapted for stateless video decoders that implement an VP8 pipeline with the <i>Memory-to-memory Stateless Video Decoder Interface</i> . Metadata associated with the frame to decode is required to be passed through the V4L2_CID_STATELESS_VP8_FRAME control. See the <i>associated Codec Control IDs</i> . Exactly one output and one capture buffer must be provided for use with this pixel format. The output buffer must contain the appropriate number of macroblocks to decode a full corresponding frame to the matching capture buffer.
V4L2_PIX_FMT_VP9	'VP90'	VP9 compressed video frame. The encoder generates one compressed frame per buffer, and the decoder requires one compressed frame per buffer.
V4L2_PIX_FMT_VP9_FRAME	'VP9F'	VP9 parsed frame, including the frame header, as extracted from the container. This format is adapted for stateless video decoders that implement a VP9 pipeline with the <i>Memory-to-memory Stateless Video Decoder Interface</i> . Metadata associated with the frame to decode is required to be passed through the V4L2_CID_STATELESS_VP9_FRAME and the V4L2_CID_STATELESS_VP9_COMPRESSED_HDR controls. See the <i>associated Codec Control IDs</i> . Exactly one output and one capture buffer must be provided for use with this pixel format. The output buffer must contain the appropriate number of macroblocks to decode a full corresponding frame to the matching capture buffer.
V4L2_PIX_FMT_HEVC	'HEVC'	HEVC/H.265 Access Unit. The decoder expects one Access Unit per buffer. The encoder generates one Access Unit per buffer. If <i>ioctl VIDIOC_ENUM_FMT</i> reports V4L2_FMT_FLAG_CONTINUOUS_BYTESTREAM then the decoder has no requirements since it can parse all the information from the raw bytestream.

Continued on next page

Table 77 - continued from previous page

Identifier	Code	Details
V4L2_PIX_FMT_HEVC_SLICE	'S265'	<p>HEVC parsed slice data, as extracted from the HEVC bitstream. This format is adapted for stateless video decoders that implement a HEVC pipeline (using the <i>Video Memory-To-Memory Interface</i> and <i>Request API</i>). This pixel format has two modifiers that must be set at least once through the <code>V4L2_CID_MPEG_VIDEO_HEVC_DECODE_MODE</code> and <code>V4L2_CID_MPEG_VIDEO_HEVC_START_CODE</code> controls. Metadata associated with the frame to decode is required to be passed through the following controls: <code>V4L2_CID_MPEG_VIDEO_HEVC_SPS</code>, <code>V4L2_CID_MPEG_VIDEO_HEVC_PPS</code>, and <code>V4L2_CID_MPEG_VIDEO_HEVC_SLICE_PARAMS</code>. See the <i>associated Codec Control IDs</i>. Buffers associated with this pixel format must contain the appropriate number of macroblocks to decode a full corresponding frame.</p> <hr/> <p>Note: This format is not yet part of the public kernel API and it is expected to change.</p> <hr/>
V4L2_PIX_FMT_FWHT	'FWHT'	<p>Video elementary stream using a codec based on the Fast Walsh Hadamard Transform. This codec is implemented by the <code>vicodec</code> ('Virtual Codec') driver. See the <code>codecfwht.h</code> header for more details. <code>ioctl VIDIOC_ENUM_FMT</code> reports <code>V4L2_FMT_FLAG_CONTINUOUS_BYTESTREAM</code> since the decoder can parse all the information from the raw bytestream.</p>
V4L2_PIX_FMT_FWHT_STATELESS	'SFWH'	<p>Same format as <code>V4L2_PIX_FMT_FWHT</code> but requires stateless codec implementation. Metadata associated with the frame to decode is required to be passed through the <code>V4L2_CID_STATELESS_FWHT_PARAMS</code> control. See the <i>associated Codec Control ID</i>.</p>

3.2.2.11 SDR Formats

These formats are used for *SDR* interface only.

V4L2_SDR_FMT_CU8 ('CU08')

Complex unsigned 8-bit IQ sample

Description

This format contains sequence of complex number samples. Each complex number consist two parts, called In-phase and Quadrature (IQ). Both I and Q are represented as a 8 bit unsigned number. I value comes first and Q value after that.

Byte Order. Each cell is one byte.

start + 0:	I' ₀
start + 1:	Q' ₀

V4L2_SDR_FMT_CU16LE ('CU16')

Complex unsigned 16-bit little endian IQ sample

Description

This format contains sequence of complex number samples. Each complex number consist two parts, called In-phase and Quadrature (IQ). Both I and Q are represented as a 16 bit unsigned little endian number. I value comes first and Q value after that.

Byte Order. Each cell is one byte.

start + 0:	$I'_0[7:0]$	$I'_0[15:8]$
start + 2:	$Q'_0[7:0]$	$Q'_0[15:8]$

V4L2_SDR_FMT_CS8 ('CS08')

Complex signed 8-bit IQ sample

Description

This format contains sequence of complex number samples. Each complex number consist two parts, called In-phase and Quadrature (IQ). Both I and Q are represented as a 8 bit signed number. I value comes first and Q value after that.

Byte Order. Each cell is one byte.

start + 0:	I'_0
start + 1:	Q'_0

V4L2_SDR_FMT_CS14LE ('CS14')

Complex signed 14-bit little endian IQ sample

Description

This format contains sequence of complex number samples. Each complex number consist two parts, called In-phase and Quadrature (IQ). Both I and Q are represented as a 14 bit signed little endian number. I value comes first and Q value after that. 14 bit value is stored in 16 bit space with unused high bits padded with 0.

Byte Order. Each cell is one byte.

start + 0:	$I'_0[7:0]$	$I'_0[13:8]$
start + 2:	$Q'_0[7:0]$	$Q'_0[13:8]$

V4L2_SDR_FMT_RU12LE ('RU12')

Real unsigned 12-bit little endian sample

Description

This format contains sequence of real number samples. Each sample is represented as a 12 bit unsigned little endian number. Sample is stored in 16 bit space with unused high bits padded with 0.

Byte Order. Each cell is one byte.

start + 0:	I' ₀ [7:0]	I' ₀ [11:8]
------------	-----------------------	------------------------

V4L2_SDR_FMT_PCU16BE ('PC16')

Planar complex unsigned 16-bit big endian IQ sample

Description

This format contains a sequence of complex number samples. Each complex number consist of two parts called In-phase and Quadrature (IQ). Both I and Q are represented as a 16 bit unsigned big endian number stored in 32 bit space. The remaining unused bits within the 32 bit space will be padded with 0. I value starts first and Q value starts at an offset equalling half of the buffer size (i.e.) offset = buffersize/2. Out of the 16 bits, bit 15:2 (14 bit) is data and bit 1:0 (2 bit) can be any value.

Byte Order. Each cell is one byte.

Offset:	Byte B0	Byte B1	Byte B2	Byte B3
start + 0:	I' ₀ [13:6]	I' ₀ [5:0]; B1[1:0]=pad	pad	pad
start + 4:	I' ₁ [13:6]	I' ₁ [5:0]; B1[1:0]=pad	pad	pad
...				
start + offset:	Q' ₀ [13:6]	Q' ₀ [5:0]; B1[1:0]=pad	pad	pad
start + offset + 4:	Q' ₁ [13:6]	Q' ₁ [5:0]; B1[1:0]=pad	pad	pad

V4L2_SDR_FMT_PCU18BE ('PC18')

Planar complex unsigned 18-bit big endian IQ sample

Description

This format contains a sequence of complex number samples. Each complex number consist of two parts called In-phase and Quadrature (IQ). Both I and Q are represented as a 18 bit unsigned big endian number stored in 32 bit space. The remaining unused bits within the 32 bit space will be padded with 0. I value starts first and Q value starts at an offset equalling half of the buffer size (i.e.) $\text{offset} = \text{buffersize}/2$. Out of the 18 bits, bit 17:2 (16 bit) is data and bit 1:0 (2 bit) can be any value.

Byte Order. Each cell is one byte.

Offset:	Byte B0	Byte B1	Byte B2	Byte B3
start + 0:	$I'_0[17:10]$	$I'_0[9:2]$	$I'_0[1:0]; B2[5:0]=\text{pad}$	pad
start + 4:	$I'_1[17:10]$	$I'_1[9:2]$	$I'_1[1:0]; B2[5:0]=\text{pad}$	pad
...				
start + offset:	$Q'_0[17:10]$	$Q'_0[9:2]$	$Q'_0[1:0]; B2[5:0]=\text{pad}$	pad
start + offset + 4:	$Q'_1[17:10]$	$Q'_1[9:2]$	$Q'_1[1:0]; B2[5:0]=\text{pad}$	pad

V4L2_SDR_FMT_PCU20BE ('PC20')

Planar complex unsigned 20-bit big endian IQ sample

Description

This format contains a sequence of complex number samples. Each complex number consist of two parts called In-phase and Quadrature (IQ). Both I and Q are represented as a 20 bit unsigned big endian number stored in 32 bit space. The remaining unused bits within the 32 bit space will be padded with 0. I value starts first and Q value starts at an offset equalling half of the buffer size (i.e.) $\text{offset} = \text{buffersize}/2$. Out of the 20 bits, bit 19:2 (18 bit) is data and bit 1:0 (2 bit) can be any value.

Byte Order. Each cell is one byte.

Offset:	Byte B0	Byte B1	Byte B2	Byte B3
start + 0:	$I'_0[19:12]$	$I'_0[11:4]$	$I'_0[3:0]; B2[3:0]=\text{pad}$	pad
start + 4:	$I'_1[19:12]$	$I'_1[11:4]$	$I'_1[3:0]; B2[3:0]=\text{pad}$	pad
...				
start + offset:	$Q'_0[19:12]$	$Q'_0[11:4]$	$Q'_0[3:0]; B2[3:0]=\text{pad}$	pad
start + offset + 4:	$Q'_1[19:12]$	$Q'_1[11:4]$	$Q'_1[3:0]; B2[3:0]=\text{pad}$	pad

3.2.2.12 Touch Formats

These formats are used for *Touch Devices* interface only.

V4L2_TCH_FMT_DELTA_TD16 ('TD16')

man V4L2_TCH_FMT_DELTA_TD16(2)

16-bit signed little endian Touch Delta

Description

This format represents delta data from a touch controller.

Delta values may range from -32768 to 32767. Typically the values will vary through a small range depending on whether the sensor is touched or not. The full value may be seen if one of the touchscreen nodes has a fault or the line is not connected.

Byte Order. Each cell is one byte.

start + 0:	D'00low	D'00high	D'01low	D'01high	D'02low	D'02high	D'03low	D'03high
start + 8:	D'10low	D'10high	D'11low	D'11high	D'12low	D'12high	D'13low	D'13high
start + 16:	D'20low	D'20high	D'21low	D'21high	D'22low	D'22high	D'23low	D'23high
start + 24:	D'30low	D'30high	D'31low	D'31high	D'32low	D'32high	D'33low	D'33high

V4L2_TCH_FMT_DELTA_TD08 ('TD08')

man V4L2_TCH_FMT_DELTA_TD08(2)

8-bit signed Touch Delta

Description

This format represents delta data from a touch controller.

Delta values may range from -128 to 127. Typically the values will vary through a small range depending on whether the sensor is touched or not. The full value may be seen if one of the touchscreen nodes has a fault or the line is not connected.

Byte Order. Each cell is one byte.

start + 0:	D'00	D'01	D'02	D'03
start + 4:	D'10	D'11	D'12	D'13
start + 8:	D'20	D'21	D'22	D'23
start + 12:	D'30	D'31	D'32	D'33

V4L2_TCH_FMT_TU16 ('TU16')

man V4L2_TCH_FMT_TU16(2)

16-bit unsigned little endian raw touch data

Description

This format represents unsigned 16-bit data from a touch controller.

This may be used for output for raw and reference data. Values may range from 0 to 65535.

Byte Order. Each cell is one byte.

start + 0:	R' _{00low}	R' _{00high}	R' _{01low}	R' _{01high}	R' _{02low}	R' _{02high}	R' _{03low}	R' _{03high}
start + 8:	R' _{10low}	R' _{10high}	R' _{11low}	R' _{11high}	R' _{12low}	R' _{12high}	R' _{13low}	R' _{13high}
start + 16:	R' _{20low}	R' _{20high}	R' _{21low}	R' _{21high}	R' _{22low}	R' _{22high}	R' _{23low}	R' _{23high}
start + 24:	R' _{30low}	R' _{30high}	R' _{31low}	R' _{31high}	R' _{32low}	R' _{32high}	R' _{33low}	R' _{33high}

V4L2_TCH_FMT_TU08 ('TU08')

man V4L2_TCH_FMT_TU08(2)

8-bit unsigned raw touch data

Description

This format represents unsigned 8-bit data from a touch controller.

This may be used for output for raw and reference data. Values may range from 0 to 255.

Byte Order. Each cell is one byte.

start + 0:	R' ₀₀	R' ₀₁	R' ₀₂	R' ₀₃
start + 4:	R' ₁₀	R' ₁₁	R' ₁₂	R' ₁₃
start + 8:	R' ₂₀	R' ₂₁	R' ₂₂	R' ₂₃
start + 12:	R' ₃₀	R' ₃₁	R' ₃₂	R' ₃₃

3.2.2.13 Metadata Formats

These formats are used for the *Metadata Interface* interface only.

V4L2_META_FMT_D4XX ('D4XX')

Intel D4xx UVC Cameras Metadata

Description

Intel D4xx (D435 and other) cameras include per-frame metadata in their UVC payload headers, following the Microsoft(R) UVC extension proposal [1]. That means, that the private D4XX metadata, following the standard UVC header, is organised in blocks. D4XX cameras implement several standard block types, proposed by Microsoft, and several proprietary ones. Supported standard metadata types are MetadataId_CaptureStats (ID 3), MetadataId_CameraExtrinsics (ID 4), and MetadataId_CameraIntrinsics (ID 5). For their description see [1]. This document describes proprietary metadata types, used by D4xx cameras.

V4L2_META_FMT_D4XX buffers follow the metadata buffer layout of V4L2_META_FMT_UVC with the only difference, that it also includes proprietary payload header data. D4xx cameras use bulk transfers and only send one payload per frame, therefore their headers cannot be larger than 255 bytes.

Below are proprietary Microsoft style metadata types, used by D4xx cameras, where all fields are in little endian order:

Table 78: D4xx metadata

Field	Description
<i>Depth Control</i>	
__u32 ID	0x80000000
__u32 Size	Size in bytes (currently 56)
__u32 Version	Version of this structure. The documentation herein corresponds to version xxx. The version number will be incremented when new fields are added.
__u32 Flags	A bitmask of flags: see [2] below
__u32 Gain	Gain value in internal units, same as the V4L2_CID_GAIN control, used to capture the frame
__u32 Exposure	Exposure time (in microseconds) used to capture the frame
__u32 Laser power	Power of the laser LED 0-360, used for depth measurement
__u32 AE mode	0: manual; 1: automatic exposure
__u32 Exposure priority	Exposure priority value: 0 - constant frame rate
__u32 AE ROI left	Left border of the AE Region of Interest (all ROI values are in pixels and lie between 0 and maximum width or height respectively)
__u32 AE ROI right	Right border of the AE Region of Interest
__u32 AE ROI top	Top border of the AE Region of Interest
__u32 AE ROI bottom	Bottom border of the AE Region of Interest
__u32 Preset	Preset selector value, default: 0, unless changed by the user
__u32 Laser mode	0: off, 1: on
<i>Capture Timing</i>	
__u32 ID	0x80000001
__u32 Size	Size in bytes (currently 40)

Continued on next page

Table 78 – continued from previous page

Field	Description
__u32 Version	Version of this structure. The documentation herein corresponds to version xxx. The version number will be incremented when new fields are added.
__u32 Flags	A bitmask of flags: see [3] below
__u32 Frame counter	Monotonically increasing counter
__u32 Optical time	Time in microseconds from the beginning of a frame till its middle
__u32 Readout time	Time, used to read out a frame in microseconds
__u32 Exposure time	Frame exposure time in microseconds
__u32 Frame interval	In microseconds = 1000000 / framerate
__u32 Pipe latency	Time in microseconds from start of frame to data in USB buffer
<i>Configuration</i>	
__u32 ID	0x80000002
__u32 Size	Size in bytes (currently 40)
__u32 Version	Version of this structure. The documentation herein corresponds to version xxx. The version number will be incremented when new fields are added.
__u32 Flags	A bitmask of flags: see [4] below
__u8 Hardware type	Camera hardware version [5]
__u8 SKU ID	Camera hardware configuration [6]
__u32 Cookie	Internal synchronisation
__u16 Format	Image format code [7]
__u16 Width	Width in pixels
__u16 Height	Height in pixels
__u16 Framerate	Requested frame rate per second
__u16 Trigger	Byte 0: bit 0: depth and RGB are synchronised, bit 1: external trigger

[1] <https://docs.microsoft.com/en-us/windows-hardware/drivers/stream/uvc-extensions-1-5>

[2] Depth Control flags specify which fields are valid:

```
0x00000001 Gain
0x00000002 Exposure
0x00000004 Laser power
0x00000008 AE mode
0x00000010 Exposure priority
0x00000020 AE ROI
0x00000040 Preset
```

[3] Capture Timing flags specify which fields are valid:

```
0x00000001 Frame counter
0x00000002 Optical time
0x00000004 Readout time
0x00000008 Exposure time
0x00000010 Frame interval
0x00000020 Pipe latency
```

[4] Configuration flags specify which fields are valid:

```
0x00000001 Hardware type
0x00000002 SKU ID
0x00000004 Cookie
0x00000008 Format
0x00000010 Width
0x00000020 Height
0x00000040 Framerate
0x00000080 Trigger
0x00000100 Cal count
```

[5] Camera model:

```
0 DS5
1 IVCAM2
```

[6] 8-bit camera hardware configuration bitfield:

```
[1:0] depthCamera
      00: no depth
      01: standard depth
      10: wide depth
      11: reserved
[2]  depthIsActive - has a laser projector
[3]  RGB presence
[4]  Inertial Measurement Unit (IMU) presence
[5]  projectorType
      0: HPTG
      1: Princeton
[6]  0: a projector, 1: an LED
[7]  reserved
```

[7] Image format codes per video streaming interface:

Depth:

```
1 Z16
2 Z
```

Left sensor:

```
1 Y8
2 UYVY
3 R8L8
4 Calibration
5 W10
```

Fish Eye sensor:

```
1 RAW8
```


V4L2_META_FMT_IPU3_PARAMS ('ip3p'), V4L2_META_FMT_IPU3_3A ('ip3s')

3A statistics

The IPU3 ImgU 3A statistics accelerators collect different statistics over an input Bayer frame. Those statistics are obtained from the “ipu3-imgu [01] 3a stat” metadata capture video nodes, using the `v4l2_meta_format` interface. They are formatted as described by the `ipu3_uapi_stats_3a` structure.

The statistics collected are AWB (Auto-white balance) RGBS (Red, Green, Blue and Saturation measure) cells, AWB filter response, AF (Auto-focus) filter response, and AE (Auto-exposure) histogram.

The struct `ipu3_uapi_4a_config` saves all configurable parameters.

```
struct ipu3_uapi_stats_3a {
    struct ipu3_uapi_awb_raw_buffer awb_raw_buffer;
    struct ipu3_uapi_ae_raw_buffer_aligned ae_raw_buffer[IPU3_UAPI_MAX_STRIPES];
    struct ipu3_uapi_af_raw_buffer af_raw_buffer;
    struct ipu3_uapi_awb_fr_raw_buffer awb_fr_raw_buffer;
    struct ipu3_uapi_4a_config stats_4a_config;
    __u32 ae_join_buffers;
    __u8 padding[28];
    struct ipu3_uapi_stats_3a_bubble_info_per_stripe stats_3a_bubble_per_stripe;
    struct ipu3_uapi_ff_status stats_3a_status;
};
```

Pipeline parameters

The pipeline parameters are passed to the “ipu3-imgu [01] parameters” metadata output video nodes, using the `v4l2_meta_format` interface. They are formatted as described by the `ipu3_uapi_params` structure.

Both 3A statistics and pipeline parameters described here are closely tied to the underlying camera sub-system (CSS) APIs. They are usually consumed and produced by dedicated user space libraries that comprise the important tuning tools, thus freeing the developers from being bothered with the low level hardware and algorithm details.

```
struct ipu3_uapi_params {
    /* Flags which of the settings below are to be applied */
    struct ipu3_uapi_flags use;

    /* Accelerator cluster parameters */
    struct ipu3_uapi_acc_param acc_param;

    /* ISP vector address space parameters */
    struct ipu3_uapi_isp_lin_vmem_params lin_vmem_params;
    struct ipu3_uapi_isp_tnr3_vmem_params tnr3_vmem_params;
    struct ipu3_uapi_isp_xnr3_vmem_params xnr3_vmem_params;

    /* ISP data memory (DMEM) parameters */
    struct ipu3_uapi_isp_tnr3_params tnr3_dmem_params;
    struct ipu3_uapi_isp_xnr3_params xnr3_dmem_params;
};
```

```
/* Optical black level compensation */
struct ipu3_uapi_obgrid_param obgrid_param;
};
```

Intel IPU3 ImgU uAPI data types

struct **ipu3_uapi_grid_config**
Grid plane config

Definition

```
struct ipu3_uapi_grid_config {
    __u8 width;
    __u8 height;
    __u16 block_width_log2:3;
    __u16 block_height_log2:3;
    __u16 height_per_slice:8;
    __u16 x_start;
    __u16 y_start;
    __u16 x_end;
    __u16 y_end;
};
```

Members

width Grid horizontal dimensions, in number of grid blocks(cells). For AWB, the range is (16, 80). For AF/AE, the range is (16, 32).

height Grid vertical dimensions, in number of grid cells. For AWB, the range is (16, 60). For AF/AE, the range is (16, 24).

block_width_log2 Log2 of the width of each cell in pixels. For AWB, the range is [3, 6]. For AF/AE, the range is [3, 7].

block_height_log2 Log2 of the height of each cell in pixels. For AWB, the range is [3, 6]. For AF/AE, the range is [3, 7].

height_per_slice The number of blocks in vertical axis per slice. Default 2.

x_start X value of top left corner of Region of Interest(ROI).

y_start Y value of top left corner of ROI

x_end X value of bottom right corner of ROI

y_end Y value of bottom right corner of ROI

Description

Due to the size of total amount of collected data, most statistics create a grid-based output, and the data is then divided into “slices”.

struct **ipu3_uapi_awb_raw_buffer**
AWB raw buffer

Definition

```
struct ipu3_uapi_awb_raw_buffer {
    __u8 meta_data[IPU3_UAPI_AWB_MAX_BUFFER_SIZE] ;
};
```

Members

meta_data buffer to hold auto white balance meta data which is the average values for each color channel.

struct **ipu3_uapi_awb_config_s**
AWB config

Definition

```
struct ipu3_uapi_awb_config_s {
    __u16 rgbs_thr_gr;
    __u16 rgbs_thr_r;
    __u16 rgbs_thr_gb;
    __u16 rgbs_thr_b;
    struct ipu3_uapi_grid_config grid;
};
```

Members

rgbs_thr_gr gr threshold value.

rgbs_thr_r Red threshold value.

rgbs_thr_gb gb threshold value.

rgbs_thr_b Blue threshold value.

grid *ipu3_uapi_grid_config*, the default grid resolution is 16x16 cells.

Description

The threshold is a saturation measure range [0, 8191], 8191 is default. Values over threshold may be optionally rejected for averaging.

struct **ipu3_uapi_awb_config**
AWB config wrapper

Definition

```
struct ipu3_uapi_awb_config {
    struct ipu3_uapi_awb_config_s config ;
};
```

Members

config config for auto white balance as defined by *ipu3_uapi_awb_config_s*

struct **ipu3_uapi_ae_raw_buffer**
AE global weighted histogram

Definition

```
struct ipu3_uapi_ae_raw_buffer {
    __u32 vals[IPU3_UAPI_AE_BINS * IPU3_UAPI_AE_COLORS];
};
```

Members

vals Sum of IPU3_UAPI_AE_COLORS in cell

Description

Each histogram contains IPU3_UAPI_AE_BINS bins. Each bin has 24 bit unsigned for counting the number of the pixel.

struct **ipu3_uapi_ae_raw_buffer_aligned**
AE raw buffer

Definition

```
struct ipu3_uapi_ae_raw_buffer_aligned {  
    struct ipu3_uapi_ae_raw_buffer buff ;  
};
```

Members

buff *ipu3_uapi_ae_raw_buffer* to hold full frame meta data.

struct **ipu3_uapi_ae_grid_config**
AE weight grid

Definition

```
struct ipu3_uapi_ae_grid_config {  
    __u8 width;  
    __u8 height;  
    __u8 block_width_log2:4;  
    __u8 block_height_log2:4;  
    __u8 reserved0:5;  
    __u8 ae_en:1;  
    __u8 rst_hist_array:1;  
    __u8 done_rst_hist_array:1;  
    __u16 x_start;  
    __u16 y_start;  
    __u16 x_end;  
    __u16 y_end;  
};
```

Members

width Grid horizontal dimensions. Value: [16, 32], default 16.

height Grid vertical dimensions. Value: [16, 24], default 16.

block_width_log2 Log2 of the width of the grid cell, value: [3, 7].

block_height_log2 Log2 of the height of the grid cell, value: [3, 7]. default is 3 (cell size 8x8), 4 cell per grid.

reserved0 reserved

ae_en 0: does not write to *ipu3_uapi_ae_raw_buffer_aligned* array, 1: write normally.

rst_hist_array write 1 to trigger histogram array reset.

done_rst_hist_array flag for histogram array reset done.

x_start X value of top left corner of ROI, default 0.

y_start Y value of top left corner of ROI, default 0.

x_end X value of bottom right corner of ROI

y_end Y value of bottom right corner of ROI

Description

The AE block accumulates 4 global weighted histograms(R, G, B, Y) over a defined ROI within the frame. The contribution of each pixel into the histogram, defined by *ipu3_uapi_ae_weight_elem* LUT, is indexed by a grid.

```
struct ipu3_uapi_ae_weight_elem
    AE weights LUT
```

Definition

```
struct ipu3_uapi_ae_weight_elem {
    __u32 cell0:4;
    __u32 cell1:4;
    __u32 cell2:4;
    __u32 cell3:4;
    __u32 cell4:4;
    __u32 cell5:4;
    __u32 cell6:4;
    __u32 cell7:4;
};
```

Members

cell0 weighted histogram grid value.

cell1 weighted histogram grid value.

cell2 weighted histogram grid value.

cell3 weighted histogram grid value.

cell4 weighted histogram grid value.

cell5 weighted histogram grid value.

cell6 weighted histogram grid value.

cell7 weighted histogram grid value.

Description

Use weighted grid value to give a different contribution factor to each cell. Precision u4, range [0, 15].

```
struct ipu3_uapi_ae_ccm
    AE coefficients for WB and CCM
```

Definition

```
struct ipu3_uapi_ae_ccm {
    __u16 gain_gr;
    __u16 gain_r;
    __u16 gain_b;
    __u16 gain_gb;
    __s16 mat[16];
};
```

Members

gain_gr WB gain factor for the gr channels. Default 256.

gain_r WB gain factor for the r channel. Default 256.

gain_b WB gain factor for the b channel. Default 256.

gain_gb WB gain factor for the gb channels. Default 256.

mat 4x4 matrix that transforms Bayer quad output from WB to RGB+Y.

Description

Default: 128, 0, 0, 0, 0, 128, 0, 0, 0, 0, 128, 0, 0, 0, 0, 128,

As part of the raw frame pre-process stage, the WB and color conversion need to be applied to expose the impact of these gain operations.

struct **ipu3_uapi_ae_config**
 AE config

Definition

```
struct ipu3_uapi_ae_config {
    struct ipu3_uapi_ae_grid_config grid_cfg ;
    struct ipu3_uapi_ae_weight_elem weights[IPU3_UAPI_AE_WEIGHTS] ;
    struct ipu3_uapi_ae_ccm ae_ccm ;
};
```

Members

grid_cfg config for auto exposure statistics grid. See struct *ipu3_uapi_ae_grid_config*, as Imgu did not support output auto exposure statistics, so user can ignore this configuration and use the RGB table in auto-whitebalance statistics instead.

weights IPU3_UAPI_AE_WEIGHTS is based on 32x24 blocks in the grid. Each grid cell has a corresponding value in weights LUT called grid value, global histogram is updated based on grid value and pixel value.

ae_ccm Color convert matrix pre-processing block.

Description

Calculate AE grid from image resolution, resample ae weights.

struct **ipu3_uapi_af_filter_config**
 AF 2D filter for contrast measurements

Definition

```
struct ipu3_uapi_af_filter_config {
    struct {
        __u8 a1;
        __u8 a2;
        __u8 a3;
        __u8 a4;
    } y1_coeff_0;
    struct {
        __u8 a5;
    }
```

```

    __u8 a6;
    __u8 a7;
    __u8 a8;
} y1_coeff_1;
struct {
    __u8 a9;
    __u8 a10;
    __u8 a11;
    __u8 a12;
} y1_coeff_2;
__u32 y1_sign_vec;
struct {
    __u8 a1;
    __u8 a2;
    __u8 a3;
    __u8 a4;
} y2_coeff_0;
struct {
    __u8 a5;
    __u8 a6;
    __u8 a7;
    __u8 a8;
} y2_coeff_1;
struct {
    __u8 a9;
    __u8 a10;
    __u8 a11;
    __u8 a12;
} y2_coeff_2;
__u32 y2_sign_vec;
struct {
    __u8 y_gen_rate_gr;
    __u8 y_gen_rate_r;
    __u8 y_gen_rate_b;
    __u8 y_gen_rate_gb;
} y_calc;
struct {
    __u32 reserved0:8;
    __u32 y1_nf:4;
    __u32 reserved1:4;
    __u32 y2_nf:4;
    __u32 reserved2:12;
} nf;
};

```

Members

y1_coeff_0 filter Y1, structure: 3x11, support both symmetry and anti-symmetry type. A12 is center, A1-A11 are neighbours. for analyzing low frequency content, used to calculate sum of gradients in x direction.

y1_coeff_0.a1 filter1 coefficients A1, u8, default 0.

y1_coeff_0.a2 filter1 coefficients A2, u8, default 0.

y1_coeff_0.a3 filter1 coefficients A3, u8, default 0.

y1_coeff_0.a4 filter1 coefficients A4, u8, default 0.

y1_coeff_1 Struct

y1_coeff_1.a5 filter1 coefficients A5, u8, default 0.

y1_coeff_1.a6 filter1 coefficients A6, u8, default 0.

y1_coeff_1.a7 filter1 coefficients A7, u8, default 0.

y1_coeff_1.a8 filter1 coefficients A8, u8, default 0.

y1_coeff_2 Struct

y1_coeff_2.a9 filter1 coefficients A9, u8, default 0.

y1_coeff_2.a10 filter1 coefficients A10, u8, default 0.

y1_coeff_2.a11 filter1 coefficients A11, u8, default 0.

y1_coeff_2.a12 filter1 coefficients A12, u8, default 128.

y1_sign_vec Each bit corresponds to one coefficient sign bit, 0: positive, 1: negative, default 0.

y2_coeff_0 Y2, same structure as Y1. For analyzing high frequency content.

y2_coeff_0.a1 filter2 coefficients A1, u8, default 0.

y2_coeff_0.a2 filter2 coefficients A2, u8, default 0.

y2_coeff_0.a3 filter2 coefficients A3, u8, default 0.

y2_coeff_0.a4 filter2 coefficients A4, u8, default 0.

y2_coeff_1 Struct

y2_coeff_1.a5 filter2 coefficients A5, u8, default 0.

y2_coeff_1.a6 filter2 coefficients A6, u8, default 0.

y2_coeff_1.a7 filter2 coefficients A7, u8, default 0.

y2_coeff_1.a8 filter2 coefficients A8, u8, default 0.

y2_coeff_2 Struct

y2_coeff_2.a9 filter1 coefficients A9, u8, default 0.

y2_coeff_2.a10 filter1 coefficients A10, u8, default 0.

y2_coeff_2.a11 filter1 coefficients A11, u8, default 0.

y2_coeff_2.a12 filter1 coefficients A12, u8, default 128.

y2_sign_vec Each bit corresponds to one coefficient sign bit, 0: positive, 1: negative, default 0.

y_calc Pre-processing that converts Bayer quad to RGB+Y values to be used for building histogram. Range [0, 32], default 8. Rule: $y_gen_rate_gr + y_gen_rate_r + y_gen_rate_b + y_gen_rate_gb = 32$ A single Y is calculated based on sum of Gr/R/B/Gb based on their contribution ratio.

y_calc.y_gen_rate_gr Contribution ratio Gr for Y

y_calc.y_gen_rate_r Contribution ratio R for Y

y_calc.y_gen_rate_b Contribution ratio B for Y

y_calc.y_gen_rate_gb Contribution ratio Gb for Y

nf The shift right value that should be applied during the Y1/Y2 filter to make sure the total memory needed is 2 bytes per grid cell.

nf.reserved0 reserved

nf.y1_nf Normalization factor for the convolution coeffs of y1, should be log2 of the sum of the abs values of the filter coeffs, default 7 ($2^7 = 128$).

nf.reserved1 reserved

nf.y2_nf Normalization factor for y2, should be log2 of the sum of the abs values of the filter coeffs.

nf.reserved2 reserved

struct **ipu3_uapi_af_raw_buffer**
AF meta data

Definition

```
struct ipu3_uapi_af_raw_buffer {
    __u8 y_table[IPU3_UAPI_AF_Y_TABLE_MAX_SIZE] ;
};
```

Members

y_table Each color component will be convolved separately with filter1 and filter2 and the result will be summed out and averaged for each cell.

struct **ipu3_uapi_af_config_s**
AF config

Definition

```
struct ipu3_uapi_af_config_s {
    struct ipu3_uapi_af_filter_config filter_config ;
    __u8 padding[4];
    struct ipu3_uapi_grid_config grid_cfg ;
};
```

Members

filter_config AF uses Y1 and Y2 filters as configured in [ipu3_uapi_af_filter_config](#)

padding paddings

grid_cfg See [ipu3_uapi_grid_config](#), default resolution 16x16. Use large grid size for large image and vice versa.

struct **ipu3_uapi_awb_fr_raw_buffer**
AWB filter response meta data

Definition

```
struct ipu3_uapi_awb_fr_raw_buffer {
    __u8 meta_data[IPU3_UAPI_AWB_FR_BAYER_TABLE_MAX_SIZE] ;
};
```

Members

meta_data Statistics output on the grid after convolving with 1D filter.

struct **ipu3_uapi_awb_fr_config_s**
AWB filter response config

Definition

```
struct ipu3_uapi_awb_fr_config_s {
    struct ipu3_uapi_grid_config grid_cfg;
    __u8 bayer_coeff[6];
    __u16 reserved1;
    __u32 bayer_sign;
    __u8 bayer_nf;
    __u8 reserved2[7];
};
```

Members

grid_cfg grid config, default 16x16.

bayer_coeff 1D Filter 1x11 center symmetry/anti-symmetry. coefficients defaults { 0, 0, 0, 0, 0, 128 }. Applied on whole image for each Bayer channel separately by a weighted sum of its 11x1 neighbors.

reserved1 reserved

bayer_sign sign of filter coefficients, default 0.

bayer_nf normalization factor for the convolution coeffs, to make sure total memory needed is within pre-determined range. NF should be the log2 of the sum of the abs values of the filter coeffs, range [7, 14], default 7.

reserved2 reserved

struct **ipu3_uapi_4a_config**
4A config

Definition

```
struct ipu3_uapi_4a_config {
    struct ipu3_uapi_awb_config_s awb_config ;
    struct ipu3_uapi_ae_grid_config ae_grd_config;
    __u8 padding[20];
    struct ipu3_uapi_af_config_s af_config;
    struct ipu3_uapi_awb_fr_config_s awb_fr_config ;
};
```

Members

awb_config *ipu3_uapi_awb_config_s*, default resolution 16x16

ae_grd_config auto exposure statistics *ipu3_uapi_ae_grid_config*

padding paddings

af_config auto focus config *ipu3_uapi_af_config_s*

awb_fr_config *ipu3_uapi_awb_fr_config_s*, default resolution 16x16

struct **ipu3_uapi_bubble_info**
Bubble info for host side debugging

Definition

```

struct ipu3_uapi_bubble_info {
    __u32 num_of_stripes ;
    __u8 padding[28];
    __u32 num_sets;
    __u8 padding1[28];
    __u32 size_of_set;
    __u8 padding2[28];
    __u32 bubble_size;
    __u8 padding3[28];
};

```

Members

num_of_stripes A single frame is divided into several parts called stripes due to limitation on line buffer memory. The separation between the stripes is vertical. Each such stripe is processed as a single frame by the ISP pipe.

padding padding bytes.

num_sets number of sets.

padding1 padding bytes.

size_of_set set size.

padding2 padding bytes.

bubble_size is the amount of padding in the bubble expressed in “sets”.

padding3 padding bytes.

struct **ipu3_uapi_ff_status**

Enable bits for each 3A fixed function

Definition

```

struct ipu3_uapi_ff_status {
    __u32 awb_en ;
    __u8 padding[28];
    __u32 ae_en;
    __u8 padding1[28];
    __u32 af_en;
    __u8 padding2[28];
    __u32 awb_fr_en;
    __u8 padding3[28];
};

```

Members

awb_en auto white balance enable

padding padding config

ae_en auto exposure enable

padding1 padding config

af_en auto focus enable

padding2 padding config

awb_fr_en awb filter response enable bit

padding3 padding config

struct **ipu3_uapi_stats_3a**
3A statistics

Definition

```
struct ipu3_uapi_stats_3a {
    struct ipu3_uapi_awb_raw_buffer awb_raw_buffer;
    struct ipu3_uapi_ae_raw_buffer_aligned ae_raw_buffer[IPU3_UAPI_MAX_STRIPES];
    struct ipu3_uapi_af_raw_buffer af_raw_buffer;
    struct ipu3_uapi_awb_fr_raw_buffer awb_fr_raw_buffer;
    struct ipu3_uapi_4a_config stats_4a_config;
    __u32 ae_join_buffers;
    __u8 padding[28];
    struct ipu3_uapi_stats_3a_bubble_info_per_stripe stats_3a_bubble_per_stripe;
    struct ipu3_uapi_ff_status stats_3a_status;
};
```

Members

awb_raw_buffer auto white balance meta data *ipu3_uapi_awb_raw_buffer*

ae_raw_buffer auto exposure raw data *ipu3_uapi_ae_raw_buffer_aligned* current Imgu does not output the auto exposure statistics to *ae_raw_buffer*, the user such as 3A algorithm can use the RGB table in *ipu3_uapi_awb_raw_buffer* to do auto-exposure.

af_raw_buffer *ipu3_uapi_af_raw_buffer* for auto focus meta data

awb_fr_raw_buffer value as specified by *ipu3_uapi_awb_fr_raw_buffer*

stats_4a_config 4a statistics config as defined by *ipu3_uapi_4a_config*.

ae_join_buffers 1 to use *ae_raw_buffer*.

padding padding config

stats_3a_bubble_per_stripe a *ipu3_uapi_stats_3a_bubble_info_per_stripe*

stats_3a_status 3a statistics status set in *ipu3_uapi_ff_status*

struct **ipu3_uapi_bnr_static_config_wb_gains_config**
White balance gains

Definition

```
struct ipu3_uapi_bnr_static_config_wb_gains_config {
    __u16 gr;
    __u16 r;
    __u16 b;
    __u16 gb;
};
```

Members

gr white balance gain for Gr channel.

r white balance gain for R channel.

b white balance gain for B channel.

gb white balance gain for Gb channel.

Description

Precision u3.13, range [0, 8). White balance correction is done by applying a multiplicative gain to each color channels prior to BNR.

struct **ipu3_uapi_bnr_static_config_wb_gains_thr_config**
Threshold config

Definition

```
struct ipu3_uapi_bnr_static_config_wb_gains_thr_config {
    __u8 gr;
    __u8 r;
    __u8 b;
    __u8 gb;
};
```

Members

gr white balance threshold gain for Gr channel.

r white balance threshold gain for R channel.

b white balance threshold gain for B channel.

gb white balance threshold gain for Gb channel.

Description

Defines the threshold that specifies how different a defect pixel can be from its neighbors.(used by dynamic defect pixel correction sub block) Precision u4.4 range [0, 8].

struct **ipu3_uapi_bnr_static_config_thr_coeffs_config**
Noise model coefficients that controls noise threshold

Definition

```
struct ipu3_uapi_bnr_static_config_thr_coeffs_config {
    __u32 cf:13;
    __u32 reserved0:3;
    __u32 cg:5;
    __u32 ci:5;
    __u32 reserved1:1;
    __u32 r_nf:5;
};
```

Members

cf Free coefficient for threshold calculation, range [0, 8191], default 0.

reserved0 reserved

cg Gain coefficient for threshold calculation, [0, 31], default 8.

ci Intensity coefficient for threshold calculation. range [0, 0x1f] default 6. format: u3.2 (3 most significant bits represent whole number, 2 least significant bits represent the fractional part with each count representing 0.25) e.g. 6 in binary format is 00110, that translates to 1.5

reserved1 reserved

r_nf Normalization shift value for r^2 calculation, range [12, 20] where r is a radius of pixel [row, col] from center of sensor. default 14.

Description

Threshold used to distinguish between noise and details.

struct **ipu3_uapi_bnr_static_config_thr_ctrl_shd_config**
Shading config

Definition

```
struct ipu3_uapi_bnr_static_config_thr_ctrl_shd_config {
    __u8 gr;
    __u8 r;
    __u8 b;
    __u8 gb;
};
```

Members

gr Coefficient defines lens shading gain approximation for gr channel

r Coefficient defines lens shading gain approximation for r channel

b Coefficient defines lens shading gain approximation for b channel

gb Coefficient defines lens shading gain approximation for gb channel

Description

Parameters for noise model (NM) adaptation of BNR due to shading correction. All above have precision of u3.3, default to 0.

struct **ipu3_uapi_bnr_static_config_opt_center_config**
Optical center config

Definition

```
struct ipu3_uapi_bnr_static_config_opt_center_config {
    __s32 x_reset:13;
    __u32 reserved0:3;
    __s32 y_reset:13;
    __u32 reserved2:3;
};
```

Members

x_reset Reset value of X (col start - X center). Precision s12.0.

reserved0 reserved

y_reset Reset value of Y (row start - Y center). Precision s12.0.

reserved2 reserved

Description

Distance from corner to optical center for NM adaptation due to shading correction (should be calculated based on shading tables)

struct **ipu3_uapi_bnr_static_config_lut_config**
BNR square root lookup table

Definition

```
struct ipu3_uapi_bnr_static_config_lut_config {
    __u8 values[IPU3_UAPI_BNR_LUT_SIZE];
};
```

Members

values pre-calculated values of square root function.

Description

LUT implementation of square root operation.

struct **ipu3_uapi_bnr_static_config_bp_ctrl_config**
Detect bad pixels (bp)

Definition

```
struct ipu3_uapi_bnr_static_config_bp_ctrl_config {
    __u32 bp_thr_gain:5;
    __u32 reserved0:2;
    __u32 defect_mode:1;
    __u32 bp_gain:6;
    __u32 reserved1:18;
    __u32 w0_coeff:4;
    __u32 reserved2:4;
    __u32 w1_coeff:4;
    __u32 reserved3:20;
};
```

Members

bp_thr_gain Defines the threshold that specifies how different a defect pixel can be from its neighbors. Threshold is dependent on de-noise threshold calculated by algorithm. Range [4, 31], default 4.

reserved0 reserved

defect_mode Mode of addressed defect pixels, 0 - single defect pixel is expected, 1 - 2 adjacent defect pixels are expected, default 1.

bp_gain Defines how 2nd derivation that passes through a defect pixel is different from 2nd derivations that pass through neighbor pixels. u4.2, range [0, 256], default 8.

reserved1 reserved

w0_coeff Blending coefficient of defect pixel correction. Precision u4, range [0, 8], default 8.

reserved2 reserved

w1_coeff Enable influence of incorrect defect pixel correction to be avoided. Precision u4, range [1, 8], default 8.

reserved3 reserved

struct **ipu3_uapi_bnr_static_config_dn_detect_ctrl_config**
Denoising config

Definition

```
struct ipu3_uapi_bnr_static_config_dn_detect_ctrl_config {
    __u32 alpha:4;
    __u32 beta:4;
    __u32 gamma:4;
    __u32 reserved0:4;
    __u32 max_inf:4;
    __u32 reserved1:7;
    __u32 gd_enable:1;
    __u32 bpc_enable:1;
    __u32 bnr_enable:1;
    __u32 ff_enable:1;
    __u32 reserved2:1;
};
```

Members

alpha Weight of central element of smoothing filter.

beta Weight of peripheral elements of smoothing filter, default 4.

gamma Weight of diagonal elements of smoothing filter, default 4.

reserved0 reserved

max_inf Maximum increase of peripheral or diagonal element influence relative to the pre-defined value range: [0x5, 0xa]

reserved1 reserved

gd_enable Green disparity enable control, 0 - disable, 1 - enable.

bpc_enable Bad pixel correction enable control, 0 - disable, 1 - enable.

bnr_enable Bayer noise removal enable control, 0 - disable, 1 - enable.

ff_enable Fixed function enable, 0 - disable, 1 - enable.

reserved2 reserved

Description

beta and gamma parameter define the strength of the noise removal filter. All above has precision u0.4, range [0, 0xf] format: u0.4 (no / zero bits represent whole number, 4 bits represent the fractional part with each count representing 0.0625) e.g. 0xf translates to $0.0625 \times 15 = 0.9375$

```
struct ipu3_uapi_bnr_static_config_opt_center_sqr_config
    BNR optical square
```

Definition

```
struct ipu3_uapi_bnr_static_config_opt_center_sqr_config {
    __u32 x_sqr_reset;
    __u32 y_sqr_reset;
};
```

Members

x_sqr_reset Reset value of X^2 .

y_sqr_reset Reset value of Y^2 .

Description

Please note:

1. X and Y ref to *ipu3_uapi_bnr_static_config_opt_center_config*
2. Both structs are used in threshold formula to calculate r^2 , where r is a radius of pixel [row, col] from center of sensor.

struct **ipu3_uapi_bnr_static_config**
BNR static config

Definition

```
struct ipu3_uapi_bnr_static_config {
    struct ipu3_uapi_bnr_static_config_wb_gains_config wb_gains;
    struct ipu3_uapi_bnr_static_config_wb_gains_thr_config wb_gains_thr;
    struct ipu3_uapi_bnr_static_config_thr_coeffs_config thr_coeffs;
    struct ipu3_uapi_bnr_static_config_thr_ctrl_shd_config thr_ctrl_shd;
    struct ipu3_uapi_bnr_static_config_opt_center_config opt_center;
    struct ipu3_uapi_bnr_static_config_lut_config lut;
    struct ipu3_uapi_bnr_static_config_bp_ctrl_config bp_ctrl;
    struct ipu3_uapi_bnr_static_config_dn_detect_ctrl_config dn_detect_ctrl;
    __u32 column_size;
    struct ipu3_uapi_bnr_static_config_opt_center_sqr_config opt_center_sqr;
};
```

Members

wb_gains white balance gains *ipu3_uapi_bnr_static_config_wb_gains_config*

wb_gains_thr white balance gains threshold as defined by *ipu3_uapi_bnr_static_config_wb_gains*

thr_coeffs coefficients of threshold *ipu3_uapi_bnr_static_config_thr_coeffs_config*

thr_ctrl_shd control of shading threshold *ipu3_uapi_bnr_static_config_thr_ctrl_shd_config*

opt_center optical center *ipu3_uapi_bnr_static_config_opt_center_config*

lut lookup table *ipu3_uapi_bnr_static_config_lut_config*

bp_ctrl detect and remove bad pixels as defined in struct *ipu3_uapi_bnr_static_config_bp_ctrl_config*

dn_detect_ctrl detect and remove noise. *ipu3_uapi_bnr_static_config_dn_detect_ctrl_config*

column_size The number of pixels in column.

opt_center_sqr Reset value of r^2 to optical center, see *ipu3_uapi_bnr_static_config_opt_center*

Description

Above parameters and `opt_center_sqr` are used for white balance and shading.

struct **ipu3_uapi_bnr_static_config_green_disparity**
Correct green disparity

Definition

```
struct ipu3_uapi_bnr_static_config_green_disparity {
    __u32 gd_red:6;
    __u32 reserved0:2;
    __u32 gd_green:6;
};
```

```
__u32 reserved1:2;
__u32 gd_blue:6;
__u32 reserved2:10;
__u32 gd_black:14;
__u32 reserved3:2;
__u32 gd_shading:7;
__u32 reserved4:1;
__u32 gd_support:2;
__u32 reserved5:1;
__u32 gd_clip:1;
__u32 gd_central_weight:4;
};
```

Members

gd_red Shading gain coeff for gr disparity level in bright red region. Precision u0.6, default 4(0.0625).

reserved0 reserved

gd_green Shading gain coeff for gr disparity level in bright green region. Precision u0.6, default 4(0.0625).

reserved1 reserved

gd_blue Shading gain coeff for gr disparity level in bright blue region. Precision u0.6, default 4(0.0625).

reserved2 reserved

gd_black Maximal green disparity level in dark region (stronger disparity assumed to be image detail). Precision u14, default 80.

reserved3 reserved

gd_shading Change maximal green disparity level according to square distance from image center.

reserved4 reserved

gd_support Lower bound for the number of second green color pixels in current pixel neighborhood with less than threshold difference from it.

reserved5 reserved

gd_clip Turn green disparity clip on/off, [0, 1], default 1.

gd_central_weight Central pixel weight in 9 pixels weighted sum.

Description

The shading gain coeff of red, green, blue and black are used to calculate threshold given a pixel's color value and its coordinates in the image.

struct **ipu3_uapi_dm_config**
De-mosaic parameters

Definition

```
struct ipu3_uapi_dm_config {
    __u32 dm_en:1;
```

```

__u32 ch_ar_en:1;
__u32 fcc_en:1;
__u32 reserved0:13;
__u32 frame_width:16;
__u32 gamma_sc:5;
__u32 reserved1:3;
__u32 lc_ctrl:5;
__u32 reserved2:3;
__u32 cr_param1:5;
__u32 reserved3:3;
__u32 cr_param2:5;
__u32 reserved4:3;
__u32 coring_param:5;
__u32 reserved5:27;
};

```

Members

dm_en de-mosaic enable.

ch_ar_en Checker artifacts removal enable flag. Default 0.

fcc_en False color correction (FCC) enable flag. Default 0.

reserved0 reserved

frame_width do not care

gamma_sc Sharpening coefficient (coefficient of 2-d derivation of complementary color in Hamilton-Adams interpolation). u5, range [0, 31], default 8.

reserved1 reserved

lc_ctrl Parameter that controls weights of Chroma Homogeneity metric in calculation of final homogeneity metric. u5, range [0, 31], default 7.

reserved2 reserved

cr_param1 First parameter that defines Checker artifact removal feature gain. Precision u5, range [0, 31], default 8.

reserved3 reserved

cr_param2 Second parameter that defines Checker artifact removal feature gain. Precision u5, range [0, 31], default 8.

reserved4 reserved

coring_param Defines power of false color correction operation. low for preserving edge colors, high for preserving gray edge artifacts. Precision u1.4, range [0, 1.9375], default 4 (0.25).

reserved5 reserved

Description

The demosaic fixed function block is responsible to covert Bayer(mosaiced) images into color images based on demosaicing algorithm.

```

struct ipu3_uapi_ccm_mat_config
    Color correction matrix

```

Definition

```
struct ipu3_uapi_ccm_mat_config {
    __s16 coeff_m11;
    __s16 coeff_m12;
    __s16 coeff_m13;
    __s16 coeff_o_r;
    __s16 coeff_m21;
    __s16 coeff_m22;
    __s16 coeff_m23;
    __s16 coeff_o_g;
    __s16 coeff_m31;
    __s16 coeff_m32;
    __s16 coeff_m33;
    __s16 coeff_o_b;
};
```

Members

coeff_m11 CCM 3x3 coefficient, range [-65536, 65535]

coeff_m12 CCM 3x3 coefficient, range [-8192, 8191]

coeff_m13 CCM 3x3 coefficient, range [-32768, 32767]

coeff_o_r Bias 3x1 coefficient, range [-8191, 8181]

coeff_m21 CCM 3x3 coefficient, range [-32767, 32767]

coeff_m22 CCM 3x3 coefficient, range [-8192, 8191]

coeff_m23 CCM 3x3 coefficient, range [-32768, 32767]

coeff_o_g Bias 3x1 coefficient, range [-8191, 8181]

coeff_m31 CCM 3x3 coefficient, range [-32768, 32767]

coeff_m32 CCM 3x3 coefficient, range [-8192, 8191]

coeff_m33 CCM 3x3 coefficient, range [-32768, 32767]

coeff_o_b Bias 3x1 coefficient, range [-8191, 8181]

Description

Transform sensor specific color space to standard sRGB by applying 3x3 matrix and adding a bias vector O. The transformation is basically a rotation and translation in the 3-dimensional color spaces. Here are the defaults:

9775, -2671, 1087, 0 -1071, 8303, 815, 0 -23, -7887, 16103, 0

struct **ipu3_uapi_gamma_corr_ctrl**
Gamma correction

Definition

```
struct ipu3_uapi_gamma_corr_ctrl {
    __u32 enable:1;
    __u32 reserved:31;
};
```

Members

enable gamma correction enable.

reserved reserved

struct **ipu3_uapi_gamma_corr_lut**
Per-pixel tone mapping implemented as LUT.

Definition

```
struct ipu3_uapi_gamma_corr_lut {
    __u16 lut[IPU3_UAPI_GAMMA_CORR_LUT_ENTRIES];
};
```

Members

lut 256 tabulated values of the gamma function. LUT[1].. LUT[256] format u13.0, range [0, 8191].

Description

The tone mapping operation is done by a Piece wise linear graph that is implemented as a lookup table(LUT). The pixel component input intensity is the X-axis of the graph which is the table entry.

struct **ipu3_uapi_gamma_config**
Gamma config

Definition

```
struct ipu3_uapi_gamma_config {
    struct ipu3_uapi_gamma_corr_ctrl gc_ctrl ;
    struct ipu3_uapi_gamma_corr_lut gc_lut ;
};
```

Members

gc_ctrl control of gamma correction *ipu3_uapi_gamma_corr_ctrl*

gc_lut lookup table of gamma correction *ipu3_uapi_gamma_corr_lut*

struct **ipu3_uapi_csc_mat_config**
Color space conversion matrix config

Definition

```
struct ipu3_uapi_csc_mat_config {
    __s16 coeff_c11;
    __s16 coeff_c12;
    __s16 coeff_c13;
    __s16 coeff_b1;
    __s16 coeff_c21;
    __s16 coeff_c22;
    __s16 coeff_c23;
    __s16 coeff_b2;
    __s16 coeff_c31;
    __s16 coeff_c32;
    __s16 coeff_c33;
    __s16 coeff_b3;
};
```

Members

- coeff_c11** Conversion matrix value, format s0.14, range [-16384, 16383].
- coeff_c12** Conversion matrix value, format s0.14, range [-8192, 8191].
- coeff_c13** Conversion matrix value, format s0.14, range [-16384, 16383].
- coeff_b1** Bias 3x1 coefficient, s13.0 range [-8192, 8191].
- coeff_c21** Conversion matrix value, format s0.14, range [-16384, 16383].
- coeff_c22** Conversion matrix value, format s0.14, range [-8192, 8191].
- coeff_c23** Conversion matrix value, format s0.14, range [-16384, 16383].
- coeff_b2** Bias 3x1 coefficient, s13.0 range [-8192, 8191].
- coeff_c31** Conversion matrix value, format s0.14, range [-16384, 16383].
- coeff_c32** Conversion matrix value, format s0.14, range [-8192, 8191].
- coeff_c33** Conversion matrix value, format s0.14, range [-16384, 16383].
- coeff_b3** Bias 3x1 coefficient, s13.0 range [-8192, 8191].

Description

To transform each pixel from RGB to YUV (Y - brightness/luminance, UV -chroma) by applying the pixel's values by a 3x3 matrix and adding an optional bias 3x1 vector. Here are the default values for the matrix:

4898, 9617, 1867, 0, -2410, -4732, 7143, 0, 10076, -8437, -1638, 0,
(i.e. for real number 0.299, $0.299 * 2^{14}$ becomes 4898.)

struct **ipu3_uapi_cds_params**
Chroma down-scaling

Definition

```
struct ipu3_uapi_cds_params {
    __u32 ds_c00:2;
    __u32 ds_c01:2;
    __u32 ds_c02:2;
    __u32 ds_c03:2;
    __u32 ds_c10:2;
    __u32 ds_c11:2;
    __u32 ds_c12:2;
    __u32 ds_c13:2;
    __u32 ds_nf:5;
    __u32 reserved0:3;
    __u32 csc_en:1;
    __u32 uv_bin_output:1;
    __u32 reserved1:6;
};
```

Members

- ds_c00** range [0, 3]
- ds_c01** range [0, 3]
- ds_c02** range [0, 3]

ds_c03 range [0, 3]

ds_c10 range [0, 3]

ds_c11 range [0, 3]

ds_c12 range [0, 3]

ds_c13 range [0, 3]

ds_nf Normalization factor for Chroma output downscaling filter, range 0,4, default 2.

reserved0 reserved

csc_en Color space conversion enable

uv_bin_output 0: output YUV 4.2.0, 1: output YUV 4.2.2(default).

reserved1 reserved

Description

In case user does not provide, above 4x2 filter will use following defaults: 1, 3, 3, 1, 1, 3, 3, 1,

struct **ipu3_uapi_shd_grid_config**
Bayer shading(darkening) correction

Definition

```

struct ipu3_uapi_shd_grid_config {
    __u8 width;
    __u8 height;
    __u8 block_width_log2:3;
    __u8 reserved0:1;
    __u8 block_height_log2:3;
    __u8 reserved1:1;
    __u8 grid_height_per_slice;
    __s16 x_start;
    __s16 y_start;
};

```

Members

width Grid horizontal dimensions, u8, [8, 128], default 73

height Grid vertical dimensions, u8, [8, 128], default 56

block_width_log2 Log2 of the width of the grid cell in pixel count u4, [0, 15], default value 5.

reserved0 reserved

block_height_log2 Log2 of the height of the grid cell in pixel count u4, [0, 15], default value 6.

reserved1 reserved

grid_height_per_slice SHD_MAX_CELLS_PER_SET/width. (with SHD_MAX_CELLS_PER_SET = 146).

x_start X value of top left corner of sensor relative to ROI s13, [-4096, 0], default 0, only negative values.

y_start Y value of top left corner of sensor relative to ROI s13, [-4096, 0], default 0, only negative values.

struct **ipu3_uapi_shd_general_config**
Shading general config

Definition

```
struct ipu3_uapi_shd_general_config {
    __u32 init_set_vrt_offst_ul:8;
    __u32 shd_enable:1;
    __u32 gain_factor:2;
    __u32 reserved:21;
};
```

Members

init_set_vrt_offst_ul set vertical offset, $y_start \gg block_height_log2 \% grid_height_per_slice$.

shd_enable shading enable.

gain_factor Gain factor. Shift calculated anti shading value. Precision u2. 0x0 - gain factor [1, 5], means no shift interpolated value. 0x1 - gain factor [1, 9], means shift interpolated by 1. 0x2 - gain factor [1, 17], means shift interpolated by 2.

reserved reserved

Description

Correction is performed by multiplying a gain factor for each of the 4 Bayer channels as a function of the pixel location in the sensor.

struct **ipu3_uapi_shd_black_level_config**
Black level correction

Definition

```
struct ipu3_uapi_shd_black_level_config {
    __s16 bl_r;
    __s16 bl_gr;
    __s16 bl_gb;
    __s16 bl_b;
};
```

Members

bl_r Bios values for green red. s11 range [-2048, 2047].

bl_gr Bios values for green blue. s11 range [-2048, 2047].

bl_gb Bios values for red. s11 range [-2048, 2047].

bl_b Bios values for blue. s11 range [-2048, 2047].

struct **ipu3_uapi_shd_config_static**
Shading config static

Definition


```

struct ipu3_uapi_shd_config_static {
    struct ipu3_uapi_shd_grid_config grid;
    struct ipu3_uapi_shd_general_config general;
    struct ipu3_uapi_shd_black_level_config black_level;
};

```

Members

grid shading grid config *ipu3_uapi_shd_grid_config*

general shading general config *ipu3_uapi_shd_general_config*

black_level black level config for shading correction as defined by *ipu3_uapi_shd_black_level_config*

struct **ipu3_uapi_shd_lut**

Shading gain factor lookup table.

Definition

```

struct ipu3_uapi_shd_lut {
    struct {
        struct {
            __u16 r;
            __u16 gr;
        } r_and_gr[IPU3_UAPI_SHD_MAX_CELLS_PER_SET];
        __u8 reserved1[24];
        struct {
            __u16 gb;
            __u16 b;
        } gb_and_b[IPU3_UAPI_SHD_MAX_CELLS_PER_SET];
        __u8 reserved2[24];
    } sets[IPU3_UAPI_SHD_MAX_CFG_SETS];
};

```

Members

sets array

sets.r_and_gr Red and GreenR Lookup table.

sets.r_and_gr.r Red shading factor.

sets.r_and_gr.gr GreenR shading factor.

sets.reserved1 reserved

sets.gb_and_b GreenB and Blue Lookup table.

sets.gb_and_b.gb GreenB shading factor.

sets.gb_and_b.b Blue shading factor.

sets.reserved2 reserved

Description

Map to shading correction LUT register set.

struct **ipu3_uapi_shd_config**

Shading config

Definition

```
struct ipu3_uapi_shd_config {
    struct ipu3_uapi_shd_config_static shd ;
    struct ipu3_uapi_shd_lut shd_lut ;
};
```

Members

shd shading static config, see *ipu3_uapi_shd_config_static*

shd_lut shading lookup table *ipu3_uapi_shd_lut*

struct **ipu3_uapi_iefd_cux2**
IEFd Config Unit 2 parameters

Definition

```
struct ipu3_uapi_iefd_cux2 {
    __u32 x0:9;
    __u32 x1:9;
    __u32 a01:9;
    __u32 b01:5;
};
```

Members

x0 X0 point of Config Unit, u9.0, default 0.

x1 X1 point of Config Unit, u9.0, default 0.

a01 Slope A of Config Unit, s4.4, default 0.

b01 Slope B, always 0.

Description

Calculate weight for blending directed and non-directed denoise elements

All CU inputs are unsigned, they will be converted to signed when written to register, i.e. `a01` will be written to 9 bit register in s4.4 format. The data precision s4.4 means 4 bits for integer parts and 4 bits for the fractional part, the first bit indicates positive or negative value. For userspace software (commonly the imaging library), the computation for the CU slope values should be based on the slope resolution 1/16 (binary 0.0001 - the minimal interval value), the slope value range is [-256, +255]. This applies to *ipu3_uapi_iefd_cux6_ed*, *ipu3_uapi_iefd_cux2_1*, *ipu3_uapi_iefd_cux2_1*, *ipu3_uapi_iefd_cux4* and *ipu3_uapi_iefd_cux6_rad*.

Note

Each instance of Config Unit needs X coordinate of n points and slope A factor between points calculated by driver based on calibration parameters.

struct **ipu3_uapi_iefd_cux6_ed**
Calculate power of non-directed sharpening element, Config Unit 6 for edge detail (ED).

Definition

```
struct ipu3_uapi_iefd_cux6_ed {
    __u32 x0:9;
    __u32 x1:9;
};
```

```

__u32 x2:9;
__u32 reserved0:5;
__u32 x3:9;
__u32 x4:9;
__u32 x5:9;
__u32 reserved1:5;
__u32 a01:9;
__u32 a12:9;
__u32 a23:9;
__u32 reserved2:5;
__u32 a34:9;
__u32 a45:9;
__u32 reserved3:14;
__u32 b01:9;
__u32 b12:9;
__u32 b23:9;
__u32 reserved4:5;
__u32 b34:9;
__u32 b45:9;
__u32 reserved5:14;
};

```

Members

x0 X coordinate of point 0, u9.0, default 0.

x1 X coordinate of point 1, u9.0, default 0.

x2 X coordinate of point 2, u9.0, default 0.

reserved0 reserved

x3 X coordinate of point 3, u9.0, default 0.

x4 X coordinate of point 4, u9.0, default 0.

x5 X coordinate of point 5, u9.0, default 0.

reserved1 reserved

a01 slope A points 01, s4.4, default 0.

a12 slope A points 12, s4.4, default 0.

a23 slope A points 23, s4.4, default 0.

reserved2 reserved

a34 slope A points 34, s4.4, default 0.

a45 slope A points 45, s4.4, default 0.

reserved3 reserved

b01 slope B points 01, s4.4, default 0.

b12 slope B points 12, s4.4, default 0.

b23 slope B points 23, s4.4, default 0.

reserved4 reserved

b34 slope B points 34, s4.4, default 0.

b45 slope B points 45, s4.4, default 0.

reserved5 reserved.

struct **ipu3_uapi_iefd_cux2_1**

Calculate power of non-directed denoise element apply.

Definition

```
struct ipu3_uapi_iefd_cux2_1 {
    __u32 x0:9;
    __u32 x1:9;
    __u32 a01:9;
    __u32 reserved1:5;
    __u32 b01:8;
    __u32 reserved2:24;
};
```

Members

x0 X0 point of Config Unit, u9.0, default 0.

x1 X1 point of Config Unit, u9.0, default 0.

a01 Slope A of Config Unit, s4.4, default 0.

reserved1 reserved

b01 offset B0 of Config Unit, u7.0, default 0.

reserved2 reserved

struct **ipu3_uapi_iefd_cux4**

Calculate power of non-directed sharpening element.

Definition

```
struct ipu3_uapi_iefd_cux4 {
    __u32 x0:9;
    __u32 x1:9;
    __u32 x2:9;
    __u32 reserved0:5;
    __u32 x3:9;
    __u32 a01:9;
    __u32 a12:9;
    __u32 reserved1:5;
    __u32 a23:9;
    __u32 b01:8;
    __u32 b12:8;
    __u32 reserved2:7;
    __u32 b23:8;
    __u32 reserved3:24;
};
```

Members

x0 X0 point of Config Unit, u9.0, default 0.

x1 X1 point of Config Unit, u9.0, default 0.

x2 X2 point of Config Unit, u9.0, default 0.

reserved0 reserved

x3 X3 point of Config Unit, u9.0, default 0.

a01 Slope A0 of Config Unit, s4.4, default 0.

a12 Slope A1 of Config Unit, s4.4, default 0.

reserved1 reserved

a23 Slope A2 of Config Unit, s4.4, default 0.

b01 Offset B0 of Config Unit, s7.0, default 0.

b12 Offset B1 of Config Unit, s7.0, default 0.

reserved2 reserved

b23 Offset B2 of Config Unit, s7.0, default 0.

reserved3 reserved

struct **ipu3_uapi_iefd_cux6_rad**
Radial Config Unit (CU)

Definition

```
struct ipu3_uapi_iefd_cux6_rad {
    __u32 x0:8;
    __u32 x1:8;
    __u32 x2:8;
    __u32 x3:8;
    __u32 x4:8;
    __u32 x5:8;
    __u32 reserved1:16;
    __u32 a01:16;
    __u32 a12:16;
    __u32 a23:16;
    __u32 a34:16;
    __u32 a45:16;
    __u32 reserved2:16;
    __u32 b01:10;
    __u32 b12:10;
    __u32 b23:10;
    __u32 reserved4:2;
    __u32 b34:10;
    __u32 b45:10;
    __u32 reserved5:12;
};
```

Members

x0 x0 points of Config Unit radial, u8.0

x1 x1 points of Config Unit radial, u8.0

x2 x2 points of Config Unit radial, u8.0

x3 x3 points of Config Unit radial, u8.0

x4 x4 points of Config Unit radial, u8.0

x5 x5 points of Config Unit radial, u8.0

reserved1 reserved

a01 Slope A of Config Unit radial, s7.8

a12 Slope A of Config Unit radial, s7.8

a23 Slope A of Config Unit radial, s7.8

a34 Slope A of Config Unit radial, s7.8

a45 Slope A of Config Unit radial, s7.8

reserved2 reserved

b01 Slope B of Config Unit radial, s9.0

b12 Slope B of Config Unit radial, s9.0

b23 Slope B of Config Unit radial, s9.0

reserved4 reserved

b34 Slope B of Config Unit radial, s9.0

b45 Slope B of Config Unit radial, s9.0

reserved5 reserved

struct **ipu3_uapi_yuvp1_iefd_cfg_units**
IEFd Config Units parameters

Definition

```
struct ipu3_uapi_yuvp1_iefd_cfg_units {
    struct ipu3_uapi_iefd_cux2 cu_1;
    struct ipu3_uapi_iefd_cux6_ed cu_ed;
    struct ipu3_uapi_iefd_cux2 cu_3;
    struct ipu3_uapi_iefd_cux2_1 cu_5;
    struct ipu3_uapi_iefd_cux4 cu_6;
    struct ipu3_uapi_iefd_cux2 cu_7;
    struct ipu3_uapi_iefd_cux4 cu_unsharp;
    struct ipu3_uapi_iefd_cux6_rad cu_radial;
    struct ipu3_uapi_iefd_cux2 cu_vssnlm;
};
```

Members

cu_1 calculate weight for blending directed and non-directed denoise elements. See [ipu3_uapi_iefd_cux2](#)

cu_ed calculate power of non-directed sharpening element, see [ipu3_uapi_iefd_cux6_ed](#)

cu_3 calculate weight for blending directed and non-directed denoise elements. A [ipu3_uapi_iefd_cux2](#)

cu_5 calculate power of non-directed denoise element apply, use [ipu3_uapi_iefd_cux2_1](#)

cu_6 calculate power of non-directed sharpening element. See [ipu3_uapi_iefd_cux4](#)

cu_7 calculate weight for blending directed and non-directed denoise elements. Use [ipu3_uapi_iefd_cux2](#)

cu_unsharp Config Unit of unsharp [ipu3_uapi_iefd_cux4](#)

cu_radial Config Unit of radial *ipu3_uapi_iefd_cux6_rad*

cu_vssnlm Config Unit of vssnlm *ipu3_uapi_iefd_cux2*

struct **ipu3_uapi_yuvp1_iefd_config_s**
IEFd config

Definition

```
struct ipu3_uapi_yuvp1_iefd_config_s {
    __u32 horver_diag_coeff:7;
    __u32 reserved0:1;
    __u32 clamp_stitch:6;
    __u32 reserved1:2;
    __u32 direct_metric_update:5;
    __u32 reserved2:3;
    __u32 ed_horver_diag_coeff:7;
    __u32 reserved3:1;
};
```

Members

horver_diag_coeff Gradient compensation. Compared with vertical / horizontal (0 / 90 degree), coefficient of diagonal (45 / 135 degree) direction should be corrected by approx. $1/\sqrt{2}$.

reserved0 reserved

clamp_stitch Slope to stitch between clamped and unclamped edge values

reserved1 reserved

direct_metric_update Update coeff for direction metric

reserved2 reserved

ed_horver_diag_coeff Radial Coefficient that compensates for different distance for vertical/horizontal and diagonal gradient calculation (approx. $1/\sqrt{2}$)

reserved3 reserved

struct **ipu3_uapi_yuvp1_iefd_control**
IEFd control

Definition

```
struct ipu3_uapi_yuvp1_iefd_control {
    __u32 iefd_en:1;
    __u32 denoise_en:1;
    __u32 direct_smooth_en:1;
    __u32 rad_en:1;
    __u32 vssnlm_en:1;
    __u32 reserved:27;
};
```

Members

iefd_en Enable IEFd

denoise_en Enable denoise

direct_smooth_en Enable directional smooth

rad_en Enable radial update

vssnlm_en Enable VSSNLM output filter

reserved reserved

struct **ipu3_uapi_sharp_cfg**
Sharpening config

Definition

```
struct ipu3_uapi_sharp_cfg {
    __u32 nega_lmt_txt:13;
    __u32 reserved0:19;
    __u32 posi_lmt_txt:13;
    __u32 reserved1:19;
    __u32 nega_lmt_dir:13;
    __u32 reserved2:19;
    __u32 posi_lmt_dir:13;
    __u32 reserved3:19;
};
```

Members

nega_lmt_txt Sharpening limit for negative overshoots for texture.

reserved0 reserved

posi_lmt_txt Sharpening limit for positive overshoots for texture.

reserved1 reserved

nega_lmt_dir Sharpening limit for negative overshoots for direction (edge).

reserved2 reserved

posi_lmt_dir Sharpening limit for positive overshoots for direction (edge).

reserved3 reserved

Description

Fixed point type u13.0, range [0, 8191].

struct **ipu3_uapi_far_w**
Sharpening config for far sub-group

Definition

```
struct ipu3_uapi_far_w {
    __u32 dir_shrp:7;
    __u32 reserved0:1;
    __u32 dir_dns:7;
    __u32 reserved1:1;
    __u32 ndir_dns_powr:7;
    __u32 reserved2:9;
};
```

Members

dir_shrp Weight of wide direct sharpening, u1.6, range [0, 64], default 64.

reserved0 reserved

dir_dns Weight of wide direct denoising, u1.6, range [0, 64], default 0.

reserved1 reserved

ndir_dns_powr Power of non-direct denoising, Precision u1.6, range [0, 64], default 64.

reserved2 reserved

struct **ipu3_uapi_unsharp_cfg**
Unsharp config

Definition

```
struct ipu3_uapi_unsharp_cfg {
    __u32 unsharp_weight:7;
    __u32 reserved0:1;
    __u32 unsharp_amount:9;
    __u32 reserved1:15;
};
```

Members

unsharp_weight Unsharp mask blending weight. u1.6, range [0, 64], default 16. 0 - disabled, 64 - use only unsharp.

reserved0 reserved

unsharp_amount Unsharp mask amount, u4.5, range [0, 511], default 0.

reserved1 reserved

struct **ipu3_uapi_yuvp1_iefd_shrp_cfg**
IEFd sharpness config

Definition

```
struct ipu3_uapi_yuvp1_iefd_shrp_cfg {
    struct ipu3_uapi_sharp_cfg cfg;
    struct ipu3_uapi_far_w far_w;
    struct ipu3_uapi_unsharp_cfg unshrp_cfg;
};
```

Members

cfg sharpness config [ipu3_uapi_sharp_cfg](#)

far_w wide range config, value as specified by [ipu3_uapi_far_w](#): The 5x5 environment is separated into 2 sub-groups, the 3x3 nearest neighbors (8 pixels called Near), and the second order neighborhood around them (16 pixels called Far).

unshrp_cfg unsharpness config. [ipu3_uapi_unsharp_cfg](#)

struct **ipu3_uapi_unsharp_coef0**
Unsharp mask coefficients

Definition

```
struct ipu3_uapi_unsharp_coef0 {
    __u32 c00:9;
    __u32 c01:9;
    __u32 c02:9;
};
```

```
__u32 reserved:5;
};
```

Members

c00 Coeff11, s0.8, range [-255, 255], default 1.

c01 Coeff12, s0.8, range [-255, 255], default 5.

c02 Coeff13, s0.8, range [-255, 255], default 9.

reserved reserved

Description

Configurable registers for common sharpening support.

struct **ipu3_uapi_unsharp_coef1**
Unsharp mask coefficients

Definition

```
struct ipu3_uapi_unsharp_coef1 {
    __u32 c11:9;
    __u32 c12:9;
    __u32 c22:9;
    __u32 reserved:5;
};
```

Members

c11 Coeff22, s0.8, range [-255, 255], default 29.

c12 Coeff23, s0.8, range [-255, 255], default 55.

c22 Coeff33, s0.8, range [-255, 255], default 96.

reserved reserved

struct **ipu3_uapi_yuvp1_iefd_unshrp_cfg**
Unsharp mask config

Definition

```
struct ipu3_uapi_yuvp1_iefd_unshrp_cfg {
    struct ipu3_uapi_unsharp_coef0 unsharp_coef0;
    struct ipu3_uapi_unsharp_coef1 unsharp_coef1;
};
```

Members

unsharp_coef0 unsharp coefficient 0 config. See [ipu3_uapi_unsharp_coef0](#)

unsharp_coef1 unsharp coefficient 1 config. See [ipu3_uapi_unsharp_coef1](#)

struct **ipu3_uapi_radial_reset_xy**
Radial coordinate reset

Definition

```

struct ipu3_uapi_radial_reset_xy {
    __s32 x:13;
    __u32 reserved0:3;
    __s32 y:13;
    __u32 reserved1:3;
};

```

Members

x Radial reset of x coordinate. Precision s12, [-4095, 4095], default 0.

reserved0 reserved

y Radial center y coordinate. Precision s12, [-4095, 4095], default 0.

reserved1 reserved

```

struct ipu3_uapi_radial_reset_x2
    Radial X^2 reset

```

Definition

```

struct ipu3_uapi_radial_reset_x2 {
    __u32 x2:24;
    __u32 reserved:8;
};

```

Members

x2 Radial reset of x² coordinate. Precision u24, default 0.

reserved reserved

```

struct ipu3_uapi_radial_reset_y2
    Radial Y^2 reset

```

Definition

```

struct ipu3_uapi_radial_reset_y2 {
    __u32 y2:24;
    __u32 reserved:8;
};

```

Members

y2 Radial reset of y² coordinate. Precision u24, default 0.

reserved reserved

```

struct ipu3_uapi_radial_cfg
    Radial config

```

Definition

```

struct ipu3_uapi_radial_cfg {
    __u32 rad_nf:4;
    __u32 reserved0:4;
    __u32 rad_inv_r2:7;
    __u32 reserved1:17;
};

```

Members

rad_nf Radial. R^2 normalization factor is scale down by $2^{-(15 + \text{scale})}$

reserved0 reserved

rad_inv_r2 Radial R^{-2} normalized to (0.5..1). Precision u7, range [0, 127].

reserved1 reserved

struct **ipu3_uapi_rad_far_w**
Radial FAR sub-group

Definition

```
struct ipu3_uapi_rad_far_w {
    __u32 rad_dir_far_sharp_w:8;
    __u32 rad_dir_far_dns_w:8;
    __u32 rad_ndir_far_dns_power:8;
    __u32 reserved:8;
};
```

Members

rad_dir_far_sharp_w Weight of wide direct sharpening, u1.6, range [0, 64], default 64.

rad_dir_far_dns_w Weight of wide direct denoising, u1.6, range [0, 64], default 0.

rad_ndir_far_dns_power power of non-direct sharpening, u1.6, range [0, 64], default 0.

reserved reserved

struct **ipu3_uapi_cu_cfg0**
Radius Config Unit cfg0 register

Definition

```
struct ipu3_uapi_cu_cfg0 {
    __u32 cu6_pow:7;
    __u32 reserved0:1;
    __u32 cu_unsharp_pow:7;
    __u32 reserved1:1;
    __u32 rad_cu6_pow:7;
    __u32 reserved2:1;
    __u32 rad_cu_unsharp_pow:6;
    __u32 reserved3:2;
};
```

Members

cu6_pow Power of CU6. Power of non-direct sharpening, u3.4.

reserved0 reserved

cu_unsharp_pow Power of unsharp mask, u2.4.

reserved1 reserved

rad_cu6_pow Radial/corner CU6. Directed sharpening power, u3.4.

reserved2 reserved

rad_cu_unsharp_pow Radial power of unsharp mask, u2.4.

reserved3 reserved

struct **ipu3_uapi_cu_cfg1**
Radius Config Unit cfg1 register

Definition

```
struct ipu3_uapi_cu_cfg1 {
    __u32 rad_cu6_x1:9;
    __u32 reserved0:1;
    __u32 rad_cu_unsharp_x1:9;
    __u32 reserved1:13;
};
```

Members

rad_cu6_x1 X1 point of Config Unit 6, precision u9.0.

reserved0 reserved

rad_cu_unsharp_x1 X1 point for Config Unit unsharp for radial/corner point precision u9.0.

reserved1 reserved

struct **ipu3_uapi_yuvp1_iefd_rad_cfg**
IEFd parameters changed radially over the picture plane.

Definition

```
struct ipu3_uapi_yuvp1_iefd_rad_cfg {
    struct ipu3_uapi_radial_reset_xy reset_xy;
    struct ipu3_uapi_radial_reset_x2 reset_x2;
    struct ipu3_uapi_radial_reset_y2 reset_y2;
    struct ipu3_uapi_radial_cfg cfg;
    struct ipu3_uapi_rad_far_w rad_far_w;
    struct ipu3_uapi_cu_cfg0 cu_cfg0;
    struct ipu3_uapi_cu_cfg1 cu_cfg1;
};
```

Members

reset_xy reset xy value in radial calculation. *ipu3_uapi_radial_reset_xy*

reset_x2 reset x square value in radial calculation. See struct *ipu3_uapi_radial_reset_x2*

reset_y2 reset y square value in radial calculation. See struct *ipu3_uapi_radial_reset_y2*

cfg radial config defined in *ipu3_uapi_radial_cfg*

rad_far_w weight for wide range radial. *ipu3_uapi_rad_far_w*

cu_cfg0 configuration unit 0. See *ipu3_uapi_cu_cfg0*

cu_cfg1 configuration unit 1. See *ipu3_uapi_cu_cfg1*

struct **ipu3_uapi_vss_lut_x**
Vssnlm LUT x0/x1/x2

Definition

```
struct ipu3_uapi_vss_lut_x {
    __u32 vs_x0:8;
```

```
__u32 vs_x1:8;
__u32 vs_x2:8;
__u32 reserved2:8;
};
```

Members

vs_x0 Vssnlm LUT x0, precision u8, range [0, 255], default 16.

vs_x1 Vssnlm LUT x1, precision u8, range [0, 255], default 32.

vs_x2 Vssnlm LUT x2, precision u8, range [0, 255], default 64.

reserved2 reserved

struct **ipu3_uapi_vss_lut_y**
Vssnlm LUT y0/y1/y2

Definition

```
struct ipu3_uapi_vss_lut_y {
    __u32 vs_y1:4;
    __u32 reserved0:4;
    __u32 vs_y2:4;
    __u32 reserved1:4;
    __u32 vs_y3:4;
    __u32 reserved2:12;
};
```

Members

vs_y1 Vssnlm LUT y1, precision u4, range [0, 8], default 1.

reserved0 reserved

vs_y2 Vssnlm LUT y2, precision u4, range [0, 8], default 3.

reserved1 reserved

vs_y3 Vssnlm LUT y3, precision u4, range [0, 8], default 8.

reserved2 reserved

struct **ipu3_uapi_yuvp1_iefd_vssnlm_cfg**
IEFd Vssnlm Lookup table

Definition

```
struct ipu3_uapi_yuvp1_iefd_vssnlm_cfg {
    struct ipu3_uapi_vss_lut_x vss_lut_x;
    struct ipu3_uapi_vss_lut_y vss_lut_y;
};
```

Members

vss_lut_x vss lookup table. See [ipu3_uapi_vss_lut_x](#) description

vss_lut_y vss lookup table. See [ipu3_uapi_vss_lut_y](#) description

struct **ipu3_uapi_yuvp1_iefd_config**
IEFd config

Definition

```

struct ipu3_uapi_yuvp1_iefd_config {
    struct ipu3_uapi_yuvp1_iefd_cfg_units units;
    struct ipu3_uapi_yuvp1_iefd_config_s config;
    struct ipu3_uapi_yuvp1_iefd_control control;
    struct ipu3_uapi_yuvp1_iefd_shrp_cfg sharp;
    struct ipu3_uapi_yuvp1_iefd_unshrp_cfg unsharp;
    struct ipu3_uapi_yuvp1_iefd_rad_cfg rad;
    struct ipu3_uapi_yuvp1_iefd_vssnlm_cfg vssnlm;
};

```

Members

units configuration unit setting, *ipu3_uapi_yuvp1_iefd_cfg_units*

config configuration, as defined by *ipu3_uapi_yuvp1_iefd_config_s*

control control setting, as defined by *ipu3_uapi_yuvp1_iefd_control*

sharp sharpness setting, as defined by *ipu3_uapi_yuvp1_iefd_shrp_cfg*

unsharp unsharpness setting, as defined by *ipu3_uapi_yuvp1_iefd_unshrp_cfg*

rad radial setting, as defined by *ipu3_uapi_yuvp1_iefd_rad_cfg*

vssnlm vssnlm setting, as defined by *ipu3_uapi_yuvp1_iefd_vssnlm_cfg*

struct **ipu3_uapi_yuvp1_yds_config**

Y Down-Sampling config

Definition

```

struct ipu3_uapi_yuvp1_yds_config {
    __u32 c00:2;
    __u32 c01:2;
    __u32 c02:2;
    __u32 c03:2;
    __u32 c10:2;
    __u32 c11:2;
    __u32 c12:2;
    __u32 c13:2;
    __u32 norm_factor:5;
    __u32 reserved0:4;
    __u32 bin_output:1;
    __u32 reserved1:6;
};

```

Members

c00 range [0, 3], default 0x0

c01 range [0, 3], default 0x1

c02 range [0, 3], default 0x1

c03 range [0, 3], default 0x0

c10 range [0, 3], default 0x0

c11 range [0, 3], default 0x1

c12 range [0, 3], default 0x1

c13 range [0, 3], default 0x0

norm_factor Normalization factor, range [0, 4], default 2 0 - divide by 1 1 - divide by 2 2 - divide by 4 3 - divide by 8 4 - divide by 16

reserved0 reserved

bin_output Down sampling on Luma channel in two optional modes 0 - Bin output 4.2.0 (default), 1 output 4.2.2.

reserved1 reserved

Description

Above are 4x2 filter coefficients for chroma output downscaling.

struct **ipu3_uapi_yuvp1_chnr_enable_config**
Chroma noise reduction enable

Definition

```
struct ipu3_uapi_yuvp1_chnr_enable_config {
    __u32 enable:1;
    __u32 yuv_mode:1;
    __u32 reserved0:14;
    __u32 col_size:12;
    __u32 reserved1:4;
};
```

Members

enable enable/disable chroma noise reduction

yuv_mode 0 - YUV420, 1 - YUV422

reserved0 reserved

col_size number of columns in the frame, max width is 2560

reserved1 reserved

struct **ipu3_uapi_yuvp1_chnr_coring_config**
Coring thresholds for UV

Definition

```
struct ipu3_uapi_yuvp1_chnr_coring_config {
    __u32 u:13;
    __u32 reserved0:3;
    __u32 v:13;
    __u32 reserved1:3;
};
```

Members

u U coring level, u0.13, range [0.0, 1.0], default 0.0

reserved0 reserved

v V coring level, u0.13, range [0.0, 1.0], default 0.0

reserved1 reserved

struct **ipu3_uapi_yuvp1_chnr_sense_gain_config**
Chroma noise reduction gains

Definition

```
struct ipu3_uapi_yuvp1_chnr_sense_gain_config {
    __u32 vy:8;
    __u32 vu:8;
    __u32 vv:8;
    __u32 reserved0:8;
    __u32 hy:8;
    __u32 hu:8;
    __u32 hv:8;
    __u32 reserved1:8;
};
```

Members

vy Sensitivity of horizontal edge of Y, default 100

vu Sensitivity of horizontal edge of U, default 100

vv Sensitivity of horizontal edge of V, default 100

reserved0 reserved

hy Sensitivity of vertical edge of Y, default 50

hu Sensitivity of vertical edge of U, default 50

hv Sensitivity of vertical edge of V, default 50

reserved1 reserved

Description

All sensitivity gain parameters have precision u13.0, range [0, 8191].

struct **ipu3_uapi_yuvp1_chnr_iir_fir_config**
Chroma IIR/FIR filter config

Definition

```
struct ipu3_uapi_yuvp1_chnr_iir_fir_config {
    __u32 fir_0h:6;
    __u32 reserved0:2;
    __u32 fir_1h:6;
    __u32 reserved1:2;
    __u32 fir_2h:6;
    __u32 dalpha_clip_val:9;
    __u32 reserved2:1;
};
```

Members

fir_0h Value of center tap in horizontal FIR, range [0, 32], default 8.

reserved0 reserved

fir_1h Value of distance 1 in horizontal FIR, range [0, 32], default 12.

reserved1 reserved

fir_2h Value of distance 2 tap in horizontal FIR, range [0, 32], default 0.

dalphi_clip_val weight for previous row in IIR, range [1, 256], default 0.

reserved2 reserved

struct **ipu3_uapi_yuvp1_chnr_config**
Chroma noise reduction config

Definition

```
struct ipu3_uapi_yuvp1_chnr_config {
    struct ipu3_uapi_yuvp1_chnr_enable_config enable;
    struct ipu3_uapi_yuvp1_chnr_coring_config coring;
    struct ipu3_uapi_yuvp1_chnr_sense_gain_config sense_gain;
    struct ipu3_uapi_yuvp1_chnr_iir_fir_config iir_fir;
};
```

Members

enable chroma noise reduction enable, see [ipu3_uapi_yuvp1_chnr_enable_config](#)

coring coring config for chroma noise reduction, see [ipu3_uapi_yuvp1_chnr_coring_config](#)

sense_gain sensitivity config for chroma noise reduction, see [ipu3_uapi_yuvp1_chnr_sense_gain_config](#)

iir_fir iir and fir config for chroma noise reduction, see [ipu3_uapi_yuvp1_chnr_iir_fir_config](#)

struct **ipu3_uapi_yuvp1_y_ee_nr_lpf_config**
Luma(Y) edge enhancement low-pass filter coefficients

Definition

```
struct ipu3_uapi_yuvp1_y_ee_nr_lpf_config {
    __u32 a_diag:5;
    __u32 reserved0:3;
    __u32 a_periph:5;
    __u32 reserved1:3;
    __u32 a_cent:5;
    __u32 reserved2:9;
    __u32 enable:1;
};
```

Members

a_diag Smoothing diagonal coefficient, u5.0.

reserved0 reserved

a_periph Image smoothing peripheral, u5.0.

reserved1 reserved

a_cent Image Smoothing center coefficient, u5.0.

reserved2 reserved

enable 0: Y_EE_NR disabled, output = input; 1: Y_EE_NR enabled.

struct **ipu3_uapi_yuvp1_y_ee_nr_sense_config**
 Luma(Y) edge enhancement noise reduction sensitivity gains

Definition

```
struct ipu3_uapi_yuvp1_y_ee_nr_sense_config {
    __u32 edge_sense_0:13;
    __u32 reserved0:3;
    __u32 delta_edge_sense:13;
    __u32 reserved1:3;
    __u32 corner_sense_0:13;
    __u32 reserved2:3;
    __u32 delta_corner_sense:13;
    __u32 reserved3:3;
};
```

Members

edge_sense_0 Sensitivity of edge in dark area. u13.0, default 8191.

reserved0 reserved

delta_edge_sense Difference in the sensitivity of edges between the bright and dark areas. u13.0, default 0.

reserved1 reserved

corner_sense_0 Sensitivity of corner in dark area. u13.0, default 0.

reserved2 reserved

delta_corner_sense Difference in the sensitivity of corners between the bright and dark areas. u13.0, default 8191.

reserved3 reserved

struct **ipu3_uapi_yuvp1_y_ee_nr_gain_config**
 Luma(Y) edge enhancement noise reduction gain config

Definition

```
struct ipu3_uapi_yuvp1_y_ee_nr_gain_config {
    __u32 gain_pos_0:5;
    __u32 reserved0:3;
    __u32 delta_gain_posi:5;
    __u32 reserved1:3;
    __u32 gain_neg_0:5;
    __u32 reserved2:3;
    __u32 delta_gain_neg:5;
    __u32 reserved3:3;
};
```

Members

gain_pos_0 Gain for positive edge in dark area. u5.0, [0, 16], default 2.

reserved0 reserved

delta_gain_posi Difference in the gain of edges between the bright and dark areas for positive edges. u5.0, [0, 16], default 0.

reserved1 reserved

gain_neg_0 Gain for negative edge in dark area. u5.0, [0, 16], default 8.

reserved2 reserved

delta_gain_neg Difference in the gain of edges between the bright and dark areas for negative edges. u5.0, [0, 16], default 0.

reserved3 reserved

struct **ipu3_uapi_yuvp1_y_ee_nr_clip_config**
Luma(Y) edge enhancement noise reduction clipping config

Definition

```
struct ipu3_uapi_yuvp1_y_ee_nr_clip_config {
    __u32 clip_pos_0:5;
    __u32 reserved0:3;
    __u32 delta_clip_posi:5;
    __u32 reserved1:3;
    __u32 clip_neg_0:5;
    __u32 reserved2:3;
    __u32 delta_clip_neg:5;
    __u32 reserved3:3;
};
```

Members

clip_pos_0 Limit of positive edge in dark area u5, value [0, 16], default 8.

reserved0 reserved

delta_clip_posi Difference in the limit of edges between the bright and dark areas for positive edges. u5, value [0, 16], default 8.

reserved1 reserved

clip_neg_0 Limit of negative edge in dark area u5, value [0, 16], default 8.

reserved2 reserved

delta_clip_neg Difference in the limit of edges between the bright and dark areas for negative edges. u5, value [0, 16], default 8.

reserved3 reserved

struct **ipu3_uapi_yuvp1_y_ee_nr_frng_config**
Luma(Y) edge enhancement noise reduction fringe config

Definition

```
struct ipu3_uapi_yuvp1_y_ee_nr_frng_config {
    __u32 gain_exp:4;
    __u32 reserved0:28;
    __u32 min_edge:13;
    __u32 reserved1:3;
    __u32 lin_seg_param:4;
    __u32 reserved2:4;
    __u32 t1:1;
    __u32 t2:1;
    __u32 reserved3:6;
};
```

Members

gain_exp Common exponent of gains, u4, [0, 8], default 2.

reserved0 reserved

min_edge Threshold for edge and smooth stitching, u13.

reserved1 reserved

lin_seg_param Power of LinSeg, u4.

reserved2 reserved

t1 Parameter for enabling/disabling the edge enhancement, u1.0, [0, 1], default 1.

t2 Parameter for enabling/disabling the smoothing, u1.0, [0, 1], default 1.

reserved3 reserved

struct **ipu3_uapi_yuvp1_y_ee_nr_diag_config**

Luma(Y) edge enhancement noise reduction diagonal config

Definition

```
struct ipu3_uapi_yuvp1_y_ee_nr_diag_config {
    __u32 diag_disc_g:4;
    __u32 reserved0:4;
    __u32 hvw_hor:4;
    __u32 dw_hor:4;
    __u32 hvw_diag:4;
    __u32 dw_diag:4;
    __u32 reserved1:8;
};
```

Members

diag_disc_g Coefficient that prioritize diagonal edge direction on horizontal or vertical for final enhancement. u4.0, [1, 15], default 1.

reserved0 reserved

hvw_hor Weight of horizontal/vertical edge enhancement for hv edge. u2.2, [1, 15], default 4.

dw_hor Weight of diagonal edge enhancement for hv edge. u2.2, [1, 15], default 1.

hvw_diag Weight of horizontal/vertical edge enhancement for diagonal edge. u2.2, [1, 15], default 1.

dw_diag Weight of diagonal edge enhancement for diagonal edge. u2.2, [1, 15], default 4.

reserved1 reserved

struct **ipu3_uapi_yuvp1_y_ee_nr_fc_coring_config**

Luma(Y) edge enhancement noise reduction false color correction (FCC) coring config

Definition

```
struct ipu3_uapi_yuvp1_y_ee_nr_fc_coring_config {
    __u32 pos_0:13;
    __u32 reserved0:3;
    __u32 pos_delta:13;
    __u32 reserved1:3;
};
```

```
__u32 neg_0:13;
__u32 reserved2:3;
__u32 neg_delta:13;
__u32 reserved3:3;
};
```

Members

pos_0 Gain for positive edge in dark, u13.0, [0, 16], default 0.

reserved0 reserved

pos_delta Gain for positive edge in bright, value: $\text{pos_0} + \text{pos_delta} \leq 16$ u13.0, default 0.

reserved1 reserved

neg_0 Gain for negative edge in dark area, u13.0, range [0, 16], default 0.

reserved2 reserved

neg_delta Gain for negative edge in bright area. $\text{neg_0} + \text{neg_delta} \leq 16$ u13.0, default 0.

reserved3 reserved

Description

Coring is a simple soft thresholding technique.

struct **ipu3_uapi_yuvp1_y_ee_nr_config**
Edge enhancement and noise reduction

Definition

```
struct ipu3_uapi_yuvp1_y_ee_nr_config {
    struct ipu3_uapi_yuvp1_y_ee_nr_lpf_config lpf;
    struct ipu3_uapi_yuvp1_y_ee_nr_sense_config sense;
    struct ipu3_uapi_yuvp1_y_ee_nr_gain_config gain;
    struct ipu3_uapi_yuvp1_y_ee_nr_clip_config clip;
    struct ipu3_uapi_yuvp1_y_ee_nr_frng_config frng;
    struct ipu3_uapi_yuvp1_y_ee_nr_diag_config diag;
    struct ipu3_uapi_yuvp1_y_ee_nr_fc_coring_config fc_coring;
};
```

Members

lpf low-pass filter config. See [ipu3_uapi_yuvp1_y_ee_nr_lpf_config](#)

sense sensitivity config. See [ipu3_uapi_yuvp1_y_ee_nr_sense_config](#)

gain gain config as defined in [ipu3_uapi_yuvp1_y_ee_nr_gain_config](#)

clip clip config as defined in [ipu3_uapi_yuvp1_y_ee_nr_clip_config](#)

frng fringe config as defined in [ipu3_uapi_yuvp1_y_ee_nr_frng_config](#)

diag diagonal edge config. See [ipu3_uapi_yuvp1_y_ee_nr_diag_config](#)

fc_coring coring config for fringe control. See [ipu3_uapi_yuvp1_y_ee_nr_fc_coring_config](#)

struct **ipu3_uapi_yuvp2_tcc_gen_control_static_config**
Total color correction general control config

Definition

```

struct ipu3_uapi_yuvp2_tcc_gen_control_static_config {
    __u32 en:1;
    __u32 blend_shift:3;
    __u32 gain_according_to_y_only:1;
    __u32 reserved0:11;
    __s32 gamma:5;
    __u32 reserved1:3;
    __s32 delta:5;
    __u32 reserved2:3;
};

```

Members

en 0 - TCC disabled. Output = input 1 - TCC enabled.

blend_shift blend shift, Range[3, 4], default NA.

gain_according_to_y_only 0: Gain is calculated according to YUV, 1: Gain is calculated according to Y only

reserved0 reserved

gamma Final blending coefficients. Values[-16, 16], default NA.

reserved1 reserved

delta Final blending coefficients. Values[-16, 16], default NA.

reserved2 reserved

struct **ipu3_uapi_yuvp2_tcc_macc_elem_static_config**
 Total color correction multi-axis color control (MACC) config

Definition

```

struct ipu3_uapi_yuvp2_tcc_macc_elem_static_config {
    __s32 a:12;
    __u32 reserved0:4;
    __s32 b:12;
    __u32 reserved1:4;
    __s32 c:12;
    __u32 reserved2:4;
    __s32 d:12;
    __u32 reserved3:4;
};

```

Members

a a coefficient for 2x2 MACC conversion matrix.

reserved0 reserved

b b coefficient 2x2 MACC conversion matrix.

reserved1 reserved

c c coefficient for 2x2 MACC conversion matrix.

reserved2 reserved

d d coefficient for 2x2 MACC conversion matrix.

reserved3 reserved

struct **ipu3_uapi_yuvp2_tcc_macc_table_static_config**
Total color correction multi-axis color control (MACC) table array

Definition

```
struct ipu3_uapi_yuvp2_tcc_macc_table_static_config {
    struct ipu3_uapi_yuvp2_tcc_macc_elem_static_config entries[IPU3_UAPI_YUVP2_TCC_MACC_
↪TABLE_ELEMENTS];
};
```

Members

entries config for multi axis color correction, as specified by
ipu3_uapi_yuvp2_tcc_macc_elem_static_config

struct **ipu3_uapi_yuvp2_tcc_inv_y_lut_static_config**
Total color correction inverse y lookup table

Definition

```
struct ipu3_uapi_yuvp2_tcc_inv_y_lut_static_config {
    __u16 entries[IPU3_UAPI_YUVP2_TCC_INV_Y_LUT_ELEMENTS];
};
```

Members

entries lookup table for inverse y estimation, and use it to estimate the ratio between luma and chroma. Chroma by approximate the absolute value of the radius on the chroma plane ($R = \sqrt{u^2+v^2}$) and luma by approximate by $1/Y$.

struct **ipu3_uapi_yuvp2_tcc_gain_pcwl_lut_static_config**
Total color correction lookup table for PCWL

Definition

```
struct ipu3_uapi_yuvp2_tcc_gain_pcwl_lut_static_config {
    __u16 entries[IPU3_UAPI_YUVP2_TCC_GAIN_PCWL_LUT_ELEMENTS];
};
```

Members

entries lookup table for gain piece wise linear transformation (PCWL)

struct **ipu3_uapi_yuvp2_tcc_r_sqr_lut_static_config**
Total color correction lookup table for r square root

Definition

```
struct ipu3_uapi_yuvp2_tcc_r_sqr_lut_static_config {
    __s16 entries[IPU3_UAPI_YUVP2_TCC_R_SQR_LUT_ELEMENTS];
};
```

Members

entries lookup table for r square root estimation

struct **ipu3_uapi_yuvp2_tcc_static_config**
Total color correction static

Definition

```

struct ipu3_uapi_yuvp2_tcc_static_config {
    struct ipu3_uapi_yuvp2_tcc_gen_control_static_config gen_control;
    struct ipu3_uapi_yuvp2_tcc_macc_table_static_config macc_table;
    struct ipu3_uapi_yuvp2_tcc_inv_y_lut_static_config inv_y_lut;
    struct ipu3_uapi_yuvp2_tcc_gain_pcwl_lut_static_config gain_pcwl;
    struct ipu3_uapi_yuvp2_tcc_r_sqr_lut_static_config r_sqr_lut;
};

```

Members

gen_control general config for Total Color Correction

macc_table config for multi axis color correction

inv_y_lut lookup table for inverse y estimation

gain_pcwl lookup table for gain PCWL

r_sqr_lut lookup table for r square root estimation.

struct **ipu3_uapi_anr_transform_config**
Advanced noise reduction transform

Definition

```

struct ipu3_uapi_anr_transform_config {
    __u32 enable:1;
    __u32 adaptive_treshhold_en:1;
    __u32 reserved1:30;
    __u8 reserved2[44];
    struct ipu3_uapi_anr_alpha alpha[3];
    struct ipu3_uapi_anr_beta beta[3];
    struct ipu3_uapi_anr_plane_color color[3];
    __u16 sqrt_lut[IPU3_UAPI_ANR_LUT_SIZE];
    __s16 xreset:13;
    __u16 reserved3:3;
    __s16 yreset:13;
    __u16 reserved4:3;
    __u32 x_sqr_reset:24;
    __u32 r_normfactor:5;
    __u32 reserved5:3;
    __u32 y_sqr_reset:24;
    __u32 gain_scale:8;
};

```

Members

enable advanced noise reduction enabled.

adaptive_treshhold_en On IPU3, adaptive threshold is always enabled.

reserved1 reserved

reserved2 reserved

alpha using following defaults: 13, 13, 13, 13, 0, 0, 0, 0 11, 11, 11, 11, 0, 0, 0, 0 14, 14, 14, 14, 0, 0, 0, 0

beta use following defaults: 24, 24, 24, 24 21, 20, 20, 21 25, 25, 25, 25

color use defaults defined in driver/media/pci/intel/ipu3-tables.c

sqrt_lut 11 bits per element, values = [724 768 810 849 887 923 958 991 1024 1056 1116 1145 1173 1201 1086 1228 1254 1280 1305 1330 1355 1379 1402 1425 1448]

xreset Reset value of X for r^2 calculation Value: col_start-X_center Constraint: Xreset + FrameWidth=4095 Xreset= -4095, default -1632.

reserved3 reserved

yreset Reset value of Y for r^2 calculation Value: row_start-Y_center Constraint: Yreset + FrameHeight=4095 Yreset= -4095, default -1224.

reserved4 reserved

x_sqr_reset Reset value of X^2 for r^2 calculation Value = (Xreset)²

r_normfactor Normalization factor for R. Default 14.

reserved5 reserved

y_sqr_reset Reset value of Y^2 for r^2 calculation Value = (Yreset)²

gain_scale Parameter describing shading gain as a function of distance from the image center. A single value per frame, loaded by the driver. Default 115.

struct **ipu3_uapi_anr_stitch_pyramid**
ANR stitch pyramid

Definition

```
struct ipu3_uapi_anr_stitch_pyramid {
    __u32 entry0:6;
    __u32 entry1:6;
    __u32 entry2:6;
    __u32 reserved:14;
};
```

Members

entry0 pyramid LUT entry0, range [0x0, 0x3f]

entry1 pyramid LUT entry1, range [0x0, 0x3f]

entry2 pyramid LUT entry2, range [0x0, 0x3f]

reserved reserved

struct **ipu3_uapi_anr_stitch_config**
ANR stitch config

Definition

```
struct ipu3_uapi_anr_stitch_config {
    __u32 anr_stitch_en;
    __u8 reserved[44];
    struct ipu3_uapi_anr_stitch_pyramid pyramid[IPU3_UAPI_ANR_PYRAMID_SIZE];
};
```

Members

anr_stitch_en enable stitch. Enabled with 1.

reserved reserved

pyramid pyramid table as defined by *ipu3_uapi_anr_stitch_pyramid* default values: { 1, 3, 5 }, { 7, 7, 5 }, { 3, 1, 3 }, { 9, 15, 21 }, { 21, 15, 9 }, { 3, 5, 15 }, { 25, 35, 35 }, { 25, 15, 5 }, { 7, 21, 35 }, { 49, 49, 35 }, { 21, 7, 7 }, { 21, 35, 49 }, { 49, 35, 21 }, { 7, 5, 15 }, { 25, 35, 35 }, { 25, 15, 5 }, { 3, 9, 15 }, { 21, 21, 15 }, { 9, 3, 1 }, { 3, 5, 7 }, { 7, 5, 3 }, { 1 }

struct **ipu3_uapi_anr_config**
ANR config

Definition

```
struct ipu3_uapi_anr_config {
    struct ipu3_uapi_anr_transform_config transform ;
    struct ipu3_uapi_anr_stitch_config stitch ;
};
```

Members

transform advanced noise reduction transform config as specified by *ipu3_uapi_anr_transform_config*

stitch create 4x4 patch from 4 surrounding 8x8 patches.

struct **ipu3_uapi_acc_param**
Accelerator cluster parameters

Definition

```
struct ipu3_uapi_acc_param {
    struct ipu3_uapi_bnr_static_config bnr;
    struct ipu3_uapi_bnr_static_config_green_disparity green_disparity ;
    struct ipu3_uapi_dm_config dm ;
    struct ipu3_uapi_ccm_mat_config ccm ;
    struct ipu3_uapi_gamma_config gamma ;
    struct ipu3_uapi_csc_mat_config csc ;
    struct ipu3_uapi_cds_params cds ;
    struct ipu3_uapi_shd_config shd ;
    struct ipu3_uapi_yuvp1_iefd_config iefd ;
    struct ipu3_uapi_yuvp1_yds_config yds_c0 ;
    struct ipu3_uapi_yuvp1_chnr_config chnr_c0 ;
    struct ipu3_uapi_yuvp1_y_ee_nr_config y_ee_nr ;
    struct ipu3_uapi_yuvp1_yds_config yds ;
    struct ipu3_uapi_yuvp1_chnr_config chnr ;
    struct ipu3_uapi_yuvp1_yds_config yds2 ;
    struct ipu3_uapi_yuvp2_tcc_static_config tcc ;
    struct ipu3_uapi_anr_config anr;
    struct ipu3_uapi_awb_fr_config_s awb_fr;
    struct ipu3_uapi_ae_config ae;
    struct ipu3_uapi_af_config_s af;
    struct ipu3_uapi_awb_config awb;
};
```

Members

bnr parameters for bayer noise reduction static config. See *ipu3_uapi_bnr_static_config*

green_disparity disparity static config between gr and gb channel. See *ipu3_uapi_bnr_static_config_green_disparity*

dm de-mosaic config. See *ipu3_uapi_dm_config*

ccm color correction matrix. See *ipu3_uapi_ccm_mat_config*

gamma gamma correction config. See *ipu3_uapi_gamma_config*

csc color space conversion matrix. See *ipu3_uapi_csc_mat_config*

cds color down sample config. See *ipu3_uapi_cds_params*

shd lens shading correction config. See *ipu3_uapi_shd_config*

iefd Image enhancement filter and denoise config. *ipu3_uapi_yuvp1_iefd_config*

yds_c0 y down scaler config. *ipu3_uapi_yuvp1_yds_config*

chnr_c0 chroma noise reduction config. *ipu3_uapi_yuvp1_chnr_config*

y_ee_nr y edge enhancement and noise reduction config. *ipu3_uapi_yuvp1_y_ee_nr_config*

yds y down scaler config. See *ipu3_uapi_yuvp1_yds_config*

chnr chroma noise reduction config. See *ipu3_uapi_yuvp1_chnr_config*

yds2 y channel down scaler config. See *ipu3_uapi_yuvp1_yds_config*

tcc total color correction config as defined in struct *ipu3_uapi_yuvp2_tcc_static_config*

anr advanced noise reduction config. See *ipu3_uapi_anr_config*

awb_fr AWB filter response config. See *ipu3_uapi_awb_fr_config*

ae auto exposure config As specified by *ipu3_uapi_ae_config*

af auto focus config. As specified by *ipu3_uapi_af_config*

awb auto white balance config. As specified by *ipu3_uapi_awb_config*

Description

ACC refers to the HW cluster containing all Fixed Functions (FFs). Each FF implements a specific algorithm.

struct **ipu3_uapi_isp_lin_vmem_params**
Linearization parameters

Definition

```
struct ipu3_uapi_isp_lin_vmem_params {
    __s16 lin_lutlow_gr[IPU3_UAPI_LIN_LUT_SIZE];
    __s16 lin_lutlow_r[IPU3_UAPI_LIN_LUT_SIZE];
    __s16 lin_lutlow_b[IPU3_UAPI_LIN_LUT_SIZE];
    __s16 lin_lutlow_gb[IPU3_UAPI_LIN_LUT_SIZE];
    __s16 lin_lutdif_gr[IPU3_UAPI_LIN_LUT_SIZE];
    __s16 lin_lutdif_r[IPU3_UAPI_LIN_LUT_SIZE];
    __s16 lin_lutdif_b[IPU3_UAPI_LIN_LUT_SIZE];
    __s16 lin_lutdif_gb[IPU3_UAPI_LIN_LUT_SIZE];
};
```

Members

lin_lutlow_gr linearization look-up table for GR channel interpolation.

lin_lutlow_r linearization look-up table for R channel interpolation.

lin_lutlow_b linearization look-up table for B channel interpolation.

lin_lutlow_gb linearization look-up table for GB channel interpolation. $\text{lin_lutlow_gr} / \text{lin_lutlow_r} / \text{lin_lutlow_b} / \text{lin_lutlow_gb} \leq \text{LIN_MAX_VALUE} - 1$.

lin_lutdif_gr $\text{lin_lutlow_gr}[i+1] - \text{lin_lutlow_gr}[i]$.

lin_lutdif_r $\text{lin_lutlow_r}[i+1] - \text{lin_lutlow_r}[i]$.

lin_lutdif_b $\text{lin_lutlow_b}[i+1] - \text{lin_lutlow_b}[i]$.

lin_lutdif_gb $\text{lin_lutlow_gb}[i+1] - \text{lin_lutlow_gb}[i]$.

struct **ipu3_uapi_isp_tnr3_vmem_params**

Temporal noise reduction vector memory parameters

Definition

```
struct ipu3_uapi_isp_tnr3_vmem_params {
    __u16 slope[IPU3_UAPI_ISP_TNR3_VMEM_LEN];
    __u16 reserved1[IPU3_UAPI_ISP_VEC_ELEMS - IPU3_UAPI_ISP_TNR3_VMEM_LEN];
    __u16 sigma[IPU3_UAPI_ISP_TNR3_VMEM_LEN];
    __u16 reserved2[IPU3_UAPI_ISP_VEC_ELEMS - IPU3_UAPI_ISP_TNR3_VMEM_LEN];
};
```

Members

slope slope setting in interpolation curve for temporal noise reduction.

reserved1 reserved

sigma knee point setting in interpolation curve for temporal noise reduction.

reserved2 reserved

struct **ipu3_uapi_isp_tnr3_params**

Temporal noise reduction v3 parameters

Definition

```
struct ipu3_uapi_isp_tnr3_params {
    __u32 knee_y1;
    __u32 knee_y2;
    __u32 maxfb_y;
    __u32 maxfb_u;
    __u32 maxfb_v;
    __u32 round_adj_y;
    __u32 round_adj_u;
    __u32 round_adj_v;
    __u32 ref_buf_select;
};
```

Members

knee_y1 Knee point TNR3 assumes standard deviation of Y,U and V at Y1 are TnrY1_Sigma_Y, U and V.

knee_y2 Knee point TNR3 assumes standard deviation of Y,U and V at Y2 are TnrY2_Sigma_Y, U and V.

maxfb_y Max feedback gain for Y

maxfb_u Max feedback gain for U

maxfb_v Max feedback gain for V

round_adj_y rounding Adjust for Y

round_adj_u rounding Adjust for U

round_adj_v rounding Adjust for V

ref_buf_select selection of the reference frame buffer to be used.

struct **ipu3_uapi_isp_xnr3_vmem_params**

Extreme noise reduction v3 vector memory parameters

Definition

```
struct ipu3_uapi_isp_xnr3_vmem_params {
    __u16 x[IPU3_UAPI_ISP_VEC_ELEMS];
    __u16 a[IPU3_UAPI_ISP_VEC_ELEMS];
    __u16 b[IPU3_UAPI_ISP_VEC_ELEMS];
    __u16 c[IPU3_UAPI_ISP_VEC_ELEMS];
};
```

Members

x xnr3 parameters.

a xnr3 parameters.

b xnr3 parameters.

c xnr3 parameters.

struct **ipu3_uapi_xnr3_alpha_params**

Extreme noise reduction v3 alpha tuning parameters

Definition

```
struct ipu3_uapi_xnr3_alpha_params {
    __u32 y0;
    __u32 u0;
    __u32 v0;
    __u32 ydiff;
    __u32 udiff;
    __u32 vdiff;
};
```

Members

y0 Sigma for Y range similarity in dark area.

u0 Sigma for U range similarity in dark area.

v0 Sigma for V range similarity in dark area.

ydiff Sigma difference for Y between bright area and dark area.

udiff Sigma difference for U between bright area and dark area.

vdiff Sigma difference for V between bright area and dark area.

struct **ipu3_uapi_xnr3_coring_params**
 Extreme noise reduction v3 coring parameters

Definition

```
struct ipu3_uapi_xnr3_coring_params {
    __u32 u0;
    __u32 v0;
    __u32 udiff;
    __u32 vdiff;
};
```

Members

u0 Coring Threshold of U channel in dark area.

v0 Coring Threshold of V channel in dark area.

udiff Threshold difference of U channel between bright and dark area.

vdiff Threshold difference of V channel between bright and dark area.

struct **ipu3_uapi_xnr3_blending_params**
 Blending factor

Definition

```
struct ipu3_uapi_xnr3_blending_params {
    __u32 strength;
};
```

Members

strength The factor for blending output with input. This is tuning parameter Higher values lead to more aggressive XNR operation.

struct **ipu3_uapi_isp_xnr3_params**
 Extreme noise reduction v3 parameters

Definition

```
struct ipu3_uapi_isp_xnr3_params {
    struct ipu3_uapi_xnr3_alpha_params alpha;
    struct ipu3_uapi_xnr3_coring_params coring;
    struct ipu3_uapi_xnr3_blending_params blending;
};
```

Members

alpha parameters for xnr3 alpha. See [ipu3_uapi_xnr3_alpha_params](#)

coring parameters for xnr3 coring. See [ipu3_uapi_xnr3_coring_params](#)

blending parameters for xnr3 blending. See [ipu3_uapi_xnr3_blending_params](#)

struct **ipu3_uapi_obgrid_param**
 Optical black level compensation parameters

Definition

```
struct ipu3_uapi_obgrid_param {
    __u16 gr;
    __u16 r;
    __u16 b;
    __u16 gb;
};
```

Members

gr Grid table values for color GR

r Grid table values for color R

b Grid table values for color B

gb Grid table values for color GB

Description

Black level is different for red, green, and blue channels. So black level compensation is different per channel.

struct **ipu3_uapi_flags**

bits to indicate which pipeline needs update

Definition

```
struct ipu3_uapi_flags {
    __u32 gdc:1;
    __u32 obgrid:1;
    __u32 reserved1:30;
    __u32 acc_bnr:1;
    __u32 acc_green_disparity:1;
    __u32 acc_dm:1;
    __u32 acc_ccm:1;
    __u32 acc_gamma:1;
    __u32 acc_csc:1;
    __u32 acc_cds:1;
    __u32 acc_shd:1;
    __u32 reserved2:2;
    __u32 acc_iefd:1;
    __u32 acc_yds_c0:1;
    __u32 acc_chnr_c0:1;
    __u32 acc_y_ee_nr:1;
    __u32 acc_yds:1;
    __u32 acc_chnr:1;
    __u32 acc_ytm:1;
    __u32 acc_yds2:1;
    __u32 acc_tcc:1;
    __u32 acc_dpc:1;
    __u32 acc_bds:1;
    __u32 acc_anr:1;
    __u32 acc_awb_fr:1;
    __u32 acc_ae:1;
    __u32 acc_af:1;
    __u32 acc_awb:1;
    __u32 reserved3:4;
    __u32 lin_vmem_params:1;
    __u32 tnr3_vmem_params:1;
};
```



```

__u32 xnr3_vmem_params:1;
__u32 tnr3_dmem_params:1;
__u32 xnr3_dmem_params:1;
__u32 reserved4:1;
__u32 obgrid_param:1;
__u32 reserved5:25;
};

```

Members

gdc 0 = no update, 1 = update.

obgrid 0 = no update, 1 = update.

reserved1 Not used.

acc_bnr 0 = no update, 1 = update.

acc_green_disparity 0 = no update, 1 = update.

acc_dm 0 = no update, 1 = update.

acc_ccm 0 = no update, 1 = update.

acc_gamma 0 = no update, 1 = update.

acc_csc 0 = no update, 1 = update.

acc_cds 0 = no update, 1 = update.

acc_shd 0 = no update, 1 = update.

reserved2 Not used.

acc_iefd 0 = no update, 1 = update.

acc_yds_c0 0 = no update, 1 = update.

acc_chnr_c0 0 = no update, 1 = update.

acc_y_ee_nr 0 = no update, 1 = update.

acc_yds 0 = no update, 1 = update.

acc_chnr 0 = no update, 1 = update.

acc_ytm 0 = no update, 1 = update.

acc_yds2 0 = no update, 1 = update.

acc_tcc 0 = no update, 1 = update.

acc_dpc 0 = no update, 1 = update.

acc_bds 0 = no update, 1 = update.

acc_anr 0 = no update, 1 = update.

acc_awb_fr 0 = no update, 1 = update.

acc_ae 0 = no update, 1 = update.

acc_af 0 = no update, 1 = update.

acc_awb 0 = no update, 1 = update.

reserved3 Not used.

lin_vmem_params 0 = no update, 1 = update.

tnr3_vmem_params 0 = no update, 1 = update.

xnr3_vmem_params 0 = no update, 1 = update.

tnr3_dmem_params 0 = no update, 1 = update.

xnr3_dmem_params 0 = no update, 1 = update.

reserved4 Not used.

obgrid_param 0 = no update, 1 = update.

reserved5 Not used.

```
struct ipu3_uapi_params
    V4L2_META_FMT_IPU3_PARAMS
```

Definition

```
struct ipu3_uapi_params {
    struct ipu3_uapi_flags use ;
    struct ipu3_uapi_acc_param acc_param;
    struct ipu3_uapi_isp_lin_vmem_params lin_vmem_params;
    struct ipu3_uapi_isp_tnr3_vmem_params tnr3_vmem_params;
    struct ipu3_uapi_isp_xnr3_vmem_params xnr3_vmem_params;
    struct ipu3_uapi_isp_tnr3_params tnr3_dmem_params;
    struct ipu3_uapi_isp_xnr3_params xnr3_dmem_params;
    struct ipu3_uapi_obgrid_param obgrid_param;
};
```

Members

use select which parameters to apply, see [ipu3_uapi_flags](#)

acc_param ACC parameters, as specified by [ipu3_uapi_acc_param](#)

lin_vmem_params linearization VMEM, as specified by [ipu3_uapi_isp_lin_vmem_params](#)

tnr3_vmem_params tnr3 VMEM as specified by [ipu3_uapi_isp_tnr3_vmem_params](#)

xnr3_vmem_params xnr3 VMEM as specified by [ipu3_uapi_isp_xnr3_vmem_params](#)

tnr3_dmem_params tnr3 DMEM as specified by [ipu3_uapi_isp_tnr3_params](#)

xnr3_dmem_params xnr3 DMEM as specified by [ipu3_uapi_isp_xnr3_params](#)

obgrid_param obgrid parameters as specified by [ipu3_uapi_obgrid_param](#)

Description

The video queue “parameters” is of format V4L2_META_FMT_IPU3_PARAMS. This is a “single plane” v4l2_meta_format using V4L2_BUF_TYPE_META_OUTPUT.

struct ipu3_uapi_params as defined below contains a lot of parameters and *ipu3_uapi_flags* selects which parameters to apply.

V4L2_META_FMT_RK_ISP1_PARAMS ('rk1p'), **V4L2_META_FMT_RK_ISP1_STAT_3A** ('rk1s')

Configuration parameters

The configuration parameters are passed to the *rkisp1_params* metadata output video node, using the *v4l2_meta_format* interface. The buffer contains a single instance of the C structure *rkisp1_params_cfg* defined in *rkisp1-config.h*. So the structure can be obtained from the buffer by:

```
struct rkisp1_params_cfg *params = (struct rkisp1_params_cfg*) buffer;
```

3A and histogram statistics

The ISP1 device collects different statistics over an input Bayer frame. Those statistics are obtained from the *rkisp1_stats* metadata capture video node, using the *v4l2_meta_format* interface. The buffer contains a single instance of the C structure *rkisp1_stat_buffer* defined in *rkisp1-config.h*. So the structure can be obtained from the buffer by:

```
struct rkisp1_stat_buffer *stats = (struct rkisp1_stat_buffer*) buffer;
```

The statistics collected are Exposure, AWB (Auto-white balance), Histogram and AF (Auto-focus). See *rkisp1_stat_buffer* for details of the statistics.

The 3A statistics and configuration parameters described here are usually consumed and produced by dedicated user space libraries that comprise the important tuning tools using software control loop.

rkisp1 uAPI data types

enum **rkisp1_cif_isp_version**
ISP variants

Constants

RKISP1_V10 used at least in rk3288 and rk3399

RKISP1_V11 declared in the original vendor code, but not used

RKISP1_V12 used at least in rk3326 and px30

RKISP1_V13 used at least in rk1808

enum **rkisp1_cif_isp_exp_ctrl_autostop**
stop modes

Constants

RKISP1_CIF_ISP_EXP_CTRL_AUTOSTOP_0 continuous measurement

RKISP1_CIF_ISP_EXP_CTRL_AUTOSTOP_1 stop measuring after a complete frame

enum **rkisp1_cif_isp_exp_meas_mode**
Exposure measure mode

Constants

RKISP1_CIF_ISP_EXP_MEASURING_MODE_0 $Y = 16 + 0.25R + 0.5G + 0.1094B$

RKISP1_CIF_ISP_EXP_MEASURING_MODE_1 $Y = (R + G + B) \times (85/256)$

struct **rkisp1_cif_isp_window**
measurement window.

Definition

```
struct rkisp1_cif_isp_window {
    __u16 h_offs;
    __u16 v_offs;
    __u16 h_size;
    __u16 v_size;
};
```

Members

h_offs the horizontal offset of the window from the left of the frame in pixels.

v_offs the vertical offset of the window from the top of the frame in pixels.

h_size the horizontal size of the window in pixels

v_size the vertical size of the window in pixels.

Description

Measurements are calculated per window inside the frame. This struct represents a window for a measurement.

struct **rkisp1_cif_isp_bls_fixed_val**
BLS fixed subtraction values

Definition

```
struct rkisp1_cif_isp_bls_fixed_val {
    __s16 r;
    __s16 gr;
    __s16 gb;
    __s16 b;
};
```

Members

r Fixed (signed!) subtraction value for Bayer pattern R

gr Fixed (signed!) subtraction value for Bayer pattern Gr

gb Fixed (signed!) subtraction value for Bayer pattern Gb

b Fixed (signed!) subtraction value for Bayer pattern B

Description

The values will be subtracted from the sensor values. Therefore a negative value means addition instead of subtraction!

struct **rkisp1_cif_isp_bls_config**
Configuration used by black level subtraction

Definition

```

struct rkisp1_cif_isp_bls_config {
    __u8 enable_auto;
    __u8 en_windows;
    struct rkisp1_cif_isp_window bls_window1;
    struct rkisp1_cif_isp_window bls_window2;
    __u8 bls_samples;
    struct rkisp1_cif_isp_bls_fixed_val fixed_val;
};

```

Members

enable_auto Automatic mode activated means that the measured values are subtracted. Otherwise the fixed subtraction values will be subtracted.

en_windows enabled window

bls_window1 Measurement window 1 size

bls_window2 Measurement window 2 size

bls_samples Set amount of measured pixels for each Bayer position (A, B,C and D) to $2^{\text{bls_samples}}$.

fixed_val Fixed subtraction values

struct **rkisp1_cif_isp_dpcc_methods_config**
Methods Configuration used by DPCC

Definition

```

struct rkisp1_cif_isp_dpcc_methods_config {
    __u32 method;
    __u32 line_thresh;
    __u32 line_mad_fac;
    __u32 pg_fac;
    __u32 rnd_thresh;
    __u32 rg_fac;
};

```

Members

method Method enable bits

line_thresh Line threshold

line_mad_fac Line MAD factor

pg_fac Peak gradient factor

rnd_thresh Rank Neighbor Difference threshold

rg_fac Rank gradient factor

Description

Methods Configuration used by Defect Pixel Cluster Correction

struct **rkisp1_cif_isp_dpcc_config**
Configuration used by DPCC

Definition

```
struct rkisp1_cif_isp_dpcc_config {
    __u32 mode;
    __u32 output_mode;
    __u32 set_use;
    struct rkisp1_cif_isp_dpcc_methods_config methods[RKISP1_CIF_ISP_DPCC_METHODS_MAX];
    __u32 ro_limits;
    __u32 rnd_offs;
};
```

Members

mode dpcc output mode

output_mode whether use hard coded methods

set_use stage1 methods set

methods methods config

ro_limits rank order limits

rnd_offs differential rank offsets for rank neighbor difference

Description

Configuration used by Defect Pixel Cluster Correction

struct **rkisp1_cif_isp_gamma_corr_curve**
gamma curve point definition y-axis (output).

Definition

```
struct rkisp1_cif_isp_gamma_corr_curve {
    __u16 gamma_y[RKISP1_CIF_ISP_DEGAMMA_CURVE_SIZE];
};
```

Members

gamma_y the values for the y-axis of gamma curve points. Each value is 12 bit.

Description

The reset values define a linear curve which has the same effect as bypass. Reset values are: gamma_y[0] = 0x0000, gamma_y[1] = 0x0100, ... gamma_y[15] = 0xf00, gamma_y[16] = 0xffff

struct **rkisp1_cif_isp_gamma_curve_x_axis_pnts**

De-Gamma Curve definition x increments (sampling points). gamma_dx0 is for the lower samples (1-8), gamma_dx1 is for the higher samples (9-16). The reset values for both fields is 0x44444444. This means that each sample is 4 units away from the previous one on the x-axis.

Definition

```
struct rkisp1_cif_isp_gamma_curve_x_axis_pnts {
    __u32 gamma_dx0;
    __u32 gamma_dx1;
};
```

Members

gamma_dx0 gamma curve sample points definitions. Bits 0:2 for sample 1. Bit 3 unused. Bits 4:6 for sample 2. bit 7 unused ... Bits 28:30 for sample 8. Bit 31 unused

gamma_dx1 gamma curve sample points definitions. Bits 0:2 for sample 9. Bit 3 unused. Bits 4:6 for sample 10. bit 7 unused ... Bits 28:30 for sample 16. Bit 31 unused

struct **rkisp1_cif_isp_sdg_config**
Configuration used by sensor degamma

Definition

```
struct rkisp1_cif_isp_sdg_config {
    struct rkisp1_cif_isp_gamma_corr_curve curve_r;
    struct rkisp1_cif_isp_gamma_corr_curve curve_g;
    struct rkisp1_cif_isp_gamma_corr_curve curve_b;
    struct rkisp1_cif_isp_gamma_curve_x_axis_pnts xa_pnts;
};
```

Members

curve_r gamma curve point definition axis for red

curve_g gamma curve point definition axis for green

curve_b gamma curve point definition axis for blue

xa_pnts x axis increments

struct **rkisp1_cif_isp_lsc_config**
Configuration used by Lens shading correction

Definition

```
struct rkisp1_cif_isp_lsc_config {
    __u16 r_data_tbl[RKISP1_CIF_ISP_LSC_SAMPLES_MAX][RKISP1_CIF_ISP_LSC_SAMPLES_MAX];
    __u16 gr_data_tbl[RKISP1_CIF_ISP_LSC_SAMPLES_MAX][RKISP1_CIF_ISP_LSC_SAMPLES_MAX];
    __u16 gb_data_tbl[RKISP1_CIF_ISP_LSC_SAMPLES_MAX][RKISP1_CIF_ISP_LSC_SAMPLES_MAX];
    __u16 b_data_tbl[RKISP1_CIF_ISP_LSC_SAMPLES_MAX][RKISP1_CIF_ISP_LSC_SAMPLES_MAX];
    __u16 x_grad_tbl[RKISP1_CIF_ISP_LSC_SECTORS_TBL_SIZE];
    __u16 y_grad_tbl[RKISP1_CIF_ISP_LSC_SECTORS_TBL_SIZE];
    __u16 x_size_tbl[RKISP1_CIF_ISP_LSC_SECTORS_TBL_SIZE];
    __u16 y_size_tbl[RKISP1_CIF_ISP_LSC_SECTORS_TBL_SIZE];
    __u16 config_width;
    __u16 config_height;
};
```

Members

r_data_tbl sample table red

gr_data_tbl sample table green (red)

gb_data_tbl sample table green (blue)

b_data_tbl sample table blue

x_grad_tbl gradient table x

y_grad_tbl gradient table y

x_size_tbl size table x

y_size_tbl size table y

config_width not used at the moment

config_height not used at the moment

struct **rkisp1_cif_isp_ie_config**
Configuration used by image effects

Definition

```
struct rkisp1_cif_isp_ie_config {
    __u16 effect;
    __u16 color_sel;
    __u16 eff_mat_1;
    __u16 eff_mat_2;
    __u16 eff_mat_3;
    __u16 eff_mat_4;
    __u16 eff_mat_5;
    __u16 eff_tint;
};
```

Members

effect values from 'enum v4l2_colorfx'. Possible values are: V4L2_COLORFX_SEPIA, V4L2_COLORFX_SET_CBCR, V4L2_COLORFX_AQUA, V4L2_COLORFX_EMOSS, V4L2_COLORFX_SKETCH, V4L2_COLORFX_BW, V4L2_COLORFX_NEGATIVE

color_sel bits 0:2 - colors bitmask (001 - blue, 010 - green, 100 - red). bits 8:15 - Threshold value of the RGB colors for the color selection effect.

eff_mat_1 3x3 Matrix Coefficients for Emboss Effect 1

eff_mat_2 3x3 Matrix Coefficients for Emboss Effect 2

eff_mat_3 3x3 Matrix Coefficients for Emboss 3/Sketch 1

eff_mat_4 3x3 Matrix Coefficients for Sketch Effect 2

eff_mat_5 3x3 Matrix Coefficients for Sketch Effect 3

eff_tint Chrominance increment values of tint (used for sepia effect)

struct **rkisp1_cif_isp_cproc_config**
Configuration used by Color Processing

Definition

```
struct rkisp1_cif_isp_cproc_config {
    __u8 c_out_range;
    __u8 y_in_range;
    __u8 y_out_range;
    __u8 contrast;
    __u8 brightness;
    __u8 sat;
    __u8 hue;
};
```

Members

c_out_range Chrominance pixel clipping range at output. (0 for limit, 1 for full)

y_in_range Luminance pixel clipping range at output.

y_out_range Luminance pixel clipping range at output.

contrast 00~ff, 0.0~1.992

brightness 80~7F, -128~+127

sat saturation, 00~FF, 0.0~1.992

hue 80~7F, -90~+87.188

struct **rkisp1_cif_isp_awb_meas_config**
Configuration for the AWB statistics

Definition

```
struct rkisp1_cif_isp_awb_meas_config {
    struct rkisp1_cif_isp_window awb_wnd;
    __u32 awb_mode;
    __u8 max_y;
    __u8 min_y;
    __u8 max_csum;
    __u8 min_c;
    __u8 frames;
    __u8 awb_ref_cr;
    __u8 awb_ref_cb;
    __u8 enable_ymax_cmp;
};
```

Members

awb_wnd white balance measurement window (in pixels)

awb_mode the awb meas mode. From enum `rkisp1_cif_isp_awb_mode_type`.

max_y only pixels values < max_y contribute to awb measurement, set to 0 to disable this feature

min_y only pixels values > min_y contribute to awb measurement

max_csum Chrominance sum maximum value, only consider pixels with Cb+Cr, smaller than threshold for awb measurements

min_c Chrominance minimum value, only consider pixels with Cb/Cr each greater than threshold value for awb measurements

frames number of frames - 1 used for mean value calculation (ucFrames=0 means 1 Frame)

awb_ref_cr reference Cr value for AWB regulation, target for AWB

awb_ref_cb reference Cb value for AWB regulation, target for AWB

enable_ymax_cmp enable Y_MAX compare (Not valid in RGB measurement mode.)

struct **rkisp1_cif_isp_awb_gain_config**
Configuration used by auto white balance gain

Definition

```
struct rkisp1_cif_isp_awb_gain_config {
    __u16 gain_red;
    __u16 gain_green_r;
```

```
    __u16 gain_blue;
    __u16 gain_green_b;
};
```

Members

gain_red gain value for red component.

gain_green_r gain value for green component in red line.

gain_blue gain value for blue component.

gain_green_b gain value for green component in blue line.

Description

All fields in this struct are 10 bit, where: 0x100h = 1, unsigned integer value, range 0 to 4 with 8 bit fractional part.

$$\text{out_data_x} = (\text{AWB_GAIN_X} * \text{in_data} + 128) >> 8$$

struct **rkisp1_cif_ispflt_config**
Configuration used by ISP filtering

Definition

```
struct rkisp1_cif_ispflt_config {
    __u32 mode;
    __u8 grn_stage1;
    __u8 chr_h_mode;
    __u8 chr_v_mode;
    __u32 thresh_bl0;
    __u32 thresh_bl1;
    __u32 thresh_sh0;
    __u32 thresh_sh1;
    __u32 lum_weight;
    __u32 fac_sh1;
    __u32 fac_sh0;
    __u32 fac_mid;
    __u32 fac_bl0;
    __u32 fac_bl1;
};
```

Members

mode ISP_FILTER_MODE register fields (from enum rkisp1_cif_ispflt_mode)

grn_stage1 Green filter stage 1 select (range 0x0...0x8)

chr_h_mode Chroma filter horizontal mode

chr_v_mode Chroma filter vertical mode

thresh_bl0 If $\text{thresh_bl1} < \text{sum_grad} < \text{thresh_bl0}$ then **fac_bl0** is selected (blurring th)

thresh_bl1 If $\text{sum_grad} < \text{thresh_bl1}$ then **fac_bl1** is selected (blurring th)

thresh_sh0 If $\text{thresh_sh0} < \text{sum_grad} < \text{thresh_sh1}$ then **thresh_sh0** is selected (sharpening th)

thresh_sh1 If $\text{thresh_sh1} < \text{sum_grad}$ then **thresh_sh1** is selected (sharpening th)

lum_weight Parameters for luminance weight function.

fac_sh1 filter factor for sharp1 level

fac_sh0 filter factor for sharp0 level

fac_mid filter factor for mid level and for static filter mode

fac_bl0 filter factor for blur 0 level

fac_bl1 filter factor for blur 1 level (max blur)

Description

All 4 threshold fields (thresh_*) are 10 bits. All 6 factor fields (fac_*) are 6 bits.

struct **rkisp1_cif_isp_bdm_config**
Configuration used by Bayer DeMosaic

Definition

```
struct rkisp1_cif_isp_bdm_config {
    __u8 demosaic_th;
};
```

Members

demosaic_th threshold for bayer demosaicing texture detection

struct **rkisp1_cif_isp_ctk_config**
Configuration used by Cross Talk correction

Definition

```
struct rkisp1_cif_isp_ctk_config {
    __u16 coeff[3][3];
    __u16 ct_offset[3];
};
```

Members

coeff color correction matrix. Values are 11-bit signed fixed-point numbers with 4 bit integer and 7 bit fractional part, ranging from -8 (0x400) to +7.992 (0x3FF). 0 is represented by 0x000 and a coefficient value of 1 as 0x080.

ct_offset Red, Green, Blue offsets for the crosstalk correction matrix

struct **rkisp1_cif_isp_goc_config**
Configuration used by Gamma Out correction

Definition

```
struct rkisp1_cif_isp_goc_config {
    __u32 mode;
    __u16 gamma_y[RKISP1_CIF_ISP_GAMMA_OUT_MAX_SAMPLES];
};
```

Members

mode goc mode (from enum rkisp1_cif_isp_goc_mode)

gamma_y gamma out curve y-axis for all color components

Description

The number of entries of **gamma_y** depends on the hardware revision as is reported by the `hw_revision` field of the struct `media_device_info` that is returned by `ioctl MEDIA_IOC_DEVICE_INFO`.

Versions `<= V11` have `RKISP1_CIF_ISP_GAMMA_OUT_MAX_SAMPLES_V10` entries, versions `>= V12` have `RKISP1_CIF_ISP_GAMMA_OUT_MAX_SAMPLES_V12` entries. `RKISP1_CIF_ISP_GAMMA_OUT_MAX_SAMPLES` is equal to the maximum of the two.

struct **rkisp1_cif_isp_hst_config**
Configuration for Histogram statistics

Definition

```
struct rkisp1_cif_isp_hst_config {
    __u32 mode;
    __u8 histogram_predivider;
    struct rkisp1_cif_isp_window meas_window;
    __u8 hist_weight[RKISP1_CIF_ISP_HISTOGRAM_WEIGHT_GRIDS_SIZE];
};
```

Members

mode histogram mode (from enum `rkisp1_cif_isp_histogram_mode`)

histogram_predivider process every stepsize pixel, all other pixels are skipped

meas_window coordinates of the measure window

hist_weight weighting factor for sub-windows

Description

The number of entries of **hist_weight** depends on the hardware revision as is reported by the `hw_revision` field of the struct `media_device_info` that is returned by `ioctl MEDIA_IOC_DEVICE_INFO`.

Versions `<= V11` have `RKISP1_CIF_ISP_HISTOGRAM_WEIGHT_GRIDS_SIZE_V10` entries, versions `>= V12` have `RKISP1_CIF_ISP_HISTOGRAM_WEIGHT_GRIDS_SIZE_V12` entries. `RKISP1_CIF_ISP_HISTOGRAM_WEIGHT_GRIDS_SIZE` is equal to the maximum of the two.

struct **rkisp1_cif_isp_aec_config**
Configuration for Auto Exposure statistics

Definition

```
struct rkisp1_cif_isp_aec_config {
    __u32 mode;
    __u32 autostop;
    struct rkisp1_cif_isp_window meas_window;
};
```

Members

mode Exposure measure mode (from enum `rkisp1_cif_isp_exp_meas_mode`)

autostop stop mode (from enum `rkisp1_cif_isp_exp_ctrl_autostop`)

meas_window coordinates of the measure window

struct **rkisp1_cif_isp_afc_config**
 Configuration for the Auto Focus statistics

Definition

```
struct rkisp1_cif_isp_afc_config {
    __u8 num_afm_win;
    struct rkisp1_cif_isp_window afm_win[RKISP1_CIF_ISP_AFM_MAX_WINDOWS];
    __u32 thres;
    __u32 var_shift;
};
```

Members

num_afm_win max RKISP1_CIF_ISP_AFM_MAX_WINDOWS

afm_win coordinates of the meas window

thres threshold used for minimizing the influence of noise

var_shift the number of bits for the shift operation at the end of the calculation chain.

enum **rkisp1_cif_isp_dpf_gain_usage**
 dpf gain usage

Constants

RKISP1_CIF_ISP_DPF_GAIN_USAGE_DISABLED don't use any gains in preprocessing stage

RKISP1_CIF_ISP_DPF_GAIN_USAGE_NF_GAINS use only the noise function gains from registers
 DPF_NF_GAIN_R, ...

RKISP1_CIF_ISP_DPF_GAIN_USAGE_LSC_GAINS use only the gains from LSC module

RKISP1_CIF_ISP_DPF_GAIN_USAGE_NF_LSC_GAINS use the noise function gains and the gains
 from LSC module

RKISP1_CIF_ISP_DPF_GAIN_USAGE_AWB_GAINS use only the gains from AWB module

RKISP1_CIF_ISP_DPF_GAIN_USAGE_AWB_LSC_GAINS use the gains from AWB and LSC module

RKISP1_CIF_ISP_DPF_GAIN_USAGE_MAX upper border (only for an internal evaluation)

enum **rkisp1_cif_isp_dpf_rb_filtersize**
 Red and blue filter sizes

Constants

RKISP1_CIF_ISP_DPF_RB_FILTERSIZE_13x9 red and blue filter kernel size 13x9 (means 7x5
 active pixel)

RKISP1_CIF_ISP_DPF_RB_FILTERSIZE_9x9 red and blue filter kernel size 9x9 (means 5x5 active
 pixel)

enum **rkisp1_cif_isp_dpf_nll_scale_mode**
 dpf noise level scale mode

Constants

RKISP1_CIF_ISP_NLL_SCALE_LINEAR use a linear scaling

RKISP1_CIF_ISP_NLL_SCALE_LOGARITHMIC use a logarithmic scaling

struct **rkisp1_cif_isp_dpf_nll**
Noise level lookup

Definition

```
struct rkisp1_cif_isp_dpf_nll {
    __u16 coeff[RKISP1_CIF_ISP_DPF_MAX_NLF_COEFFS];
    __u32 scale_mode;
};
```

Members

coeff Noise level Lookup coefficient

scale_mode dpf noise level scale mode (from *enum rkisp1_cif_isp_dpf_nll_scale_mode*)

struct **rkisp1_cif_isp_dpf_rbflt**
Red blue filter config

Definition

```
struct rkisp1_cif_isp_dpf_rbflt {
    __u32 fltsize;
    __u8 spatial_coeff[RKISP1_CIF_ISP_DPF_MAX_SPATIAL_COEFFS];
    __u8 r_enable;
    __u8 b_enable;
};
```

Members

fltsize The filter size for the red and blue pixels (from *enum rkisp1_cif_isp_dpf_rb_filtersize*)

spatial_coeff Spatial weights

r_enable enable filter processing for red pixels

b_enable enable filter processing for blue pixels

struct **rkisp1_cif_isp_dpf_gflt**
Green filter Configuration

Definition

```
struct rkisp1_cif_isp_dpf_gflt {
    __u8 spatial_coeff[RKISP1_CIF_ISP_DPF_MAX_SPATIAL_COEFFS];
    __u8 gr_enable;
    __u8 gb_enable;
};
```

Members

spatial_coeff Spatial weights

gr_enable enable filter processing for green pixels in green/red lines

gb_enable enable filter processing for green pixels in green/blue lines

struct **rkisp1_cif_isp_dpf_gain**
Noise function Configuration

Definition

```

struct rkisp1_cif_isp_dpf_gain {
    __u32 mode;
    __u16 nf_r_gain;
    __u16 nf_b_gain;
    __u16 nf_gr_gain;
    __u16 nf_gb_gain;
};

```

Members

mode dpf gain usage (from *enum rkisp1_cif_isp_dpf_gain_usage*)

nf_r_gain Noise function Gain that replaces the AWB gain for red pixels

nf_b_gain Noise function Gain that replaces the AWB gain for blue pixels

nf_gr_gain Noise function Gain that replaces the AWB gain for green pixels in a red line

nf_gb_gain Noise function Gain that replaces the AWB gain for green pixels in a blue line

struct **rkisp1_cif_isp_dpf_config**
 Configuration used by De-noising pre-filter

Definition

```

struct rkisp1_cif_isp_dpf_config {
    struct rkisp1_cif_isp_dpf_gain gain;
    struct rkisp1_cif_isp_dpf_gflt gflt;
    struct rkisp1_cif_isp_dpf_rbflt rbflt;
    struct rkisp1_cif_isp_dpf_nll nll;
};

```

Members

gain noise function gain

gflt green filter config

rbflt red blue filter config

nll noise level lookup

struct **rkisp1_cif_isp_dpf_strength_config**
 strength of the filter

Definition

```

struct rkisp1_cif_isp_dpf_strength_config {
    __u8 r;
    __u8 g;
    __u8 b;
};

```

Members

r filter strength of the RED filter

g filter strength of the GREEN filter

b filter strength of the BLUE filter

struct `rkisp1_cif_isp_isp_other_cfg`

Parameters for some blocks in rockchip isp1

Definition

```
struct rkisp1_cif_isp_isp_other_cfg {
    struct rkisp1_cif_isp_dpcc_config dpcc_config;
    struct rkisp1_cif_isp_bls_config bls_config;
    struct rkisp1_cif_isp_sdg_config sdg_config;
    struct rkisp1_cif_isp_lsc_config lsc_config;
    struct rkisp1_cif_isp_awb_gain_config awb_gain_config;
    struct rkisp1_cif_ispflt_config flt_config;
    struct rkisp1_cif_isp_bdm_config bdmm_config;
    struct rkisp1_cif_isp_ctk_config ctk_config;
    struct rkisp1_cif_isp_goc_config goc_config;
    struct rkisp1_cif_isp_dpf_config dpf_config;
    struct rkisp1_cif_isp_dpf_strength_config dpf_strength_config;
    struct rkisp1_cif_isp_cproc_config cproc_config;
    struct rkisp1_cif_isp_ie_config ie_config;
};
```

Members

dpcc_config Defect Pixel Cluster Correction config

bls_config black level subtraction config

sdg_config sensor degamma config

lsc_config Lens Shade config

awb_gain_config Auto White balance gain config

flt_config filter config

bdm_config demosaic config

ctk_config cross talk config

goc_config gamma out config

dpf_config De-noising pre-filter config

dpf_strength_config dpf strength config

cproc_config color process config

ie_config image effects config

struct `rkisp1_cif_isp_isp_meas_cfg`

Rockchip ISP1 Measure Parameters

Definition

```
struct rkisp1_cif_isp_isp_meas_cfg {
    struct rkisp1_cif_isp_awb_meas_config awb_meas_config;
    struct rkisp1_cif_isp_hst_config hst_config;
    struct rkisp1_cif_isp_aec_config aec_config;
    struct rkisp1_cif_isp_afc_config afc_config;
};
```

Members

awb_meas_config auto white balance config

hst_config histogram config

aec_config auto exposure config

afc_config auto focus config

struct **rkisp1_params_cfg**
Rockchip ISP1 Input Parameters Meta Data

Definition

```
struct rkisp1_params_cfg {
    __u32 module_en_update;
    __u32 module_ens;
    __u32 module_cfg_update;
    struct rkisp1_cif_isp_isp_meas_cfg meas;
    struct rkisp1_cif_isp_isp_other_cfg others;
};
```

Members

module_en_update mask the enable bits of which module should be updated

module_ens mask the enable value of each module, only update the module which correspond bit was set in module_en_update

module_cfg_update mask the config bits of which module should be updated

meas measurement config

others other config

struct **rkisp1_cif_isp_awb_meas**
AWB measured values

Definition

```
struct rkisp1_cif_isp_awb_meas {
    __u32 cnt;
    __u8 mean_y_or_g;
    __u8 mean_cb_or_b;
    __u8 mean_cr_or_r;
};
```

Members

cnt White pixel count, number of “white pixels” found during last measurement

mean_y_or_g Mean value of Y within window and frames, Green if RGB is selected.

mean_cb_or_b Mean value of Cb within window and frames, Blue if RGB is selected.

mean_cr_or_r Mean value of Cr within window and frames, Red if RGB is selected.

struct **rkisp1_cif_isp_awb_stat**
statistics automatic white balance data

Definition

```
struct rkisp1_cif_isp_awb_stat {
    struct rkisp1_cif_isp_awb_meas awb_mean[RKISP1_CIF_ISP_AWB_MAX_GRID];
};
```

Members

awb_mean Mean measured data

struct **rkisp1_cif_isp_bls_meas_val**
BLS measured values

Definition

```
struct rkisp1_cif_isp_bls_meas_val {
    __u16 meas_r;
    __u16 meas_gr;
    __u16 meas_gb;
    __u16 meas_b;
};
```

Members

meas_r Mean measured value for Bayer pattern R

meas_gr Mean measured value for Bayer pattern Gr

meas_gb Mean measured value for Bayer pattern Gb

meas_b Mean measured value for Bayer pattern B

struct **rkisp1_cif_isp_ae_stat**
statistics auto exposure data

Definition

```
struct rkisp1_cif_isp_ae_stat {
    __u8 exp_mean[RKISP1_CIF_ISP_AE_MEAN_MAX];
    struct rkisp1_cif_isp_bls_meas_val bls_val;
};
```

Members

exp_mean Mean luminance value of block xx

bls_val BLS measured values

Description

The number of entries of **exp_mean** depends on the hardware revision as is reported by the `hw_revision` field of the struct `media_device_info` that is returned by `ioctl MEDIA_IOC_DEVICE_INFO`.

Versions `<= V11` have `RKISP1_CIF_ISP_AE_MEAN_MAX_V10` entries, versions `>= V12` have `RKISP1_CIF_ISP_AE_MEAN_MAX_V12` entries. `RKISP1_CIF_ISP_AE_MEAN_MAX` is equal to the maximum of the two.

Image is divided into 5x5 blocks on V10 and 9x9 blocks on V12.

struct **rkisp1_cif_isp_af_meas_val**
AF measured values

Definition

```
struct rkisp1_cif_isp_af_meas_val {
    __u32 sum;
    __u32 lum;
};
```

Members

sum sharpness value

lum luminance value

struct **rkisp1_cif_isp_af_stat**
statistics auto focus data

Definition

```
struct rkisp1_cif_isp_af_stat {
    struct rkisp1_cif_isp_af_meas_val window[RKISP1_CIF_ISP_AFM_MAX_WINDOWS];
};
```

Members

window AF measured value of window x

Description

The module measures the sharpness in 3 windows of selectable size via register settings(ISP_AFM_*_A/B/C)

struct **rkisp1_cif_isp_hist_stat**
statistics histogram data

Definition

```
struct rkisp1_cif_isp_hist_stat {
    __u32 hist_bins[RKISP1_CIF_ISP_HIST_BIN_N_MAX];
};
```

Members

hist_bins measured bin counters. Each bin is a 20 bits unsigned fixed point type. Bits 0-4 are the fractional part and bits 5-19 are the integer part.

Description

The window of the measurements area is divided to 5x5 sub-windows for V10/V11 and to 9x9 sub-windows for V12. The histogram is then computed for each sub-window independently and the final result is a weighted average of the histogram measurements on all sub-windows. The window of the measurements area and the weight of each sub-window are configurable using struct **rkisp1_cif_isp_hst_config**.

The histogram contains 16 bins in V10/V11 and 32 bins in V12/V13.

The number of entries of **hist_bins** depends on the hardware revision as is reported by the `hw_revision` field of the struct `media_device_info` that is returned by `ioctl MEDIA_IOC_DEVICE_INFO`.

Versions \leq V11 have `RKISP1_CIF_ISP_HIST_BIN_N_MAX_V10` entries, versions \geq V12 have `RKISP1_CIF_ISP_HIST_BIN_N_MAX_V12` entries. `RKISP1_CIF_ISP_HIST_BIN_N_MAX` is equal to the maximum of the two.

struct **rkisp1_cif_isp_stat**
Rockchip ISP1 Statistics Data

Definition

```
struct rkisp1_cif_isp_stat {
    struct rkisp1_cif_isp_awb_stat awb;
    struct rkisp1_cif_isp_ae_stat ae;
    struct rkisp1_cif_isp_af_stat af;
    struct rkisp1_cif_isp_hist_stat hist;
};
```

Members

awb statistics data for automatic white balance

ae statistics data for auto exposure

af statistics data for auto focus

hist statistics histogram data

struct **rkisp1_stat_buffer**
Rockchip ISP1 Statistics Meta Data

Definition

```
struct rkisp1_stat_buffer {
    __u32 meas_type;
    __u32 frame_id;
    struct rkisp1_cif_isp_stat params;
};
```

Members

meas_type measurement types (`RKISP1_CIF_ISP_STAT_*` definitions)

frame_id frame ID for sync

params statistics data

V4L2_META_FMT_UVC ('UVCH')

UVC Payload Header Data

Description

This format describes standard UVC metadata, extracted from UVC packet headers and provided by the UVC driver through metadata video nodes. That data includes exact copies of the standard part of UVC Payload Header contents and auxiliary timing information, required for precise interpretation of timestamps, contained in those headers. See section “2.4.3.3 Video and Still Image Payload Headers” of the “UVC 1.5 Class specification” for details.

Each UVC payload header can be between 2 and 12 bytes large. Buffers can contain multiple headers, if multiple such headers have been transmitted by the camera for the respective frame. However, the driver may drop headers when the buffer is full, when they contain no useful information (e.g. those without the SCR field or with that field identical to the previous header), or generally to perform rate limiting when the device sends a large number of headers.

Each individual block contains the following fields:

Table 79: UVC Metadata Block

Field	Description
__u64 ts;	system timestamp in host byte order, measured by the driver upon reception of the payload
__u16 sof;	USB Frame Number in host byte order, also obtained by the driver as close as possible to the above timestamp to enable correlation between them
<i>The rest is an exact copy of the UVC payload header:</i>	
__u8 length;	length of the rest of the block, including this field
__u8 flags;	Flags, indicating presence of other standard UVC fields
__u8 buf[];	The rest of the header, possibly including UVC PTS and SCR fields

V4L2_META_FMT_VSP1_HGO ('VSPH')

Renesas R-Car VSP1 1-D Histogram Data

Description

This format describes histogram data generated by the Renesas R-Car VSP1 1-D Histogram (HGO) engine.

The VSP1 HGO is a histogram computation engine that can operate on RGB, YCrCb or HSV data. It operates on a possibly cropped and subsampled input image and computes the minimum, maximum and sum of all pixels as well as per-channel histograms.

The HGO can compute histograms independently per channel, on the maximum of the three channels (RGB data only) or on the Y channel only (YCbCr only). It can additionally output the histogram with 64 or 256 bins, resulting in four possible modes of operation.

- In *64 bins normal mode*, the HGO operates on the three channels independently to compute three 64-bins histograms. RGB, YCbCr and HSV image formats are supported.
- In *64 bins maximum mode*, the HGO operates on the maximum of the (R, G, B) channels to compute a single 64-bins histogram. Only the RGB image format is supported.

- In *256 bins normal mode*, the HGO operates on the Y channel to compute a single 256-bins histogram. Only the YCbCr image format is supported.
- In *256 bins maximum mode*, the HGO operates on the maximum of the (R, G, B) channels to compute a single 256-bins histogram. Only the RGB image format is supported.

Byte Order. All data is stored in memory in little endian format. Each cell in the tables contains one byte.

Table 80: VSP1 HGO Data - 64 Bins, Normal Mode (792 bytes)

Offset	Memory			
	[31:24]	[23:16]	[15:8]	[7:0]
0		R/Cr/H max [7:0]		R/Cr/H min [7:0]
4		G/Y/S max [7:0]		G/Y/S min [7:0]
8		B/Cb/V max [7:0]		B/Cb/V min [7:0]
12	R/Cr/H sum [31:0]			
16	G/Y/S sum [31:0]			
20	B/Cb/V sum [31:0]			
24	R/Cr/H bin 0 [31:0]			
	...			
276	R/Cr/H bin 63 [31:0]			
280	G/Y/S bin 0 [31:0]			
	...			
532	G/Y/S bin 63 [31:0]			
536	B/Cb/V bin 0 [31:0]			
	...			
788	B/Cb/V bin 63 [31:0]			

Table 81: VSP1 HGO Data - 64 Bins, Max Mode (264 bytes)

Offset	Memory			
	[31:24]	[23:16]	[15:8]	[7:0]
0		max(R,G,B) max [7:0]		max(R,G,B) min [7:0]
4	max(R,G,B) sum [31:0]			
8	max(R,G,B) bin 0 [31:0]			
	...			
260	max(R,G,B) bin 63 [31:0]			

Table 82: VSP1 HGO Data - 256 Bins, Normal Mode (1032 bytes)

Offset	Memory			
	[31:24]	[23:16]	[15:8]	[7:0]
0		Y max [7:0]		Y min [7:0]
4	Y sum [31:0]			
8	Y bin 0 [31:0]			
	...			
1028	Y bin 255 [31:0]			

Table 83: VSP1 HGO Data - 256 Bins, Max Mode (1032 bytes)

Offset	Memory			
	[31:24]	[23:16]	[15:8]	[7:0]
0		max(R,G,B) max [7:0]		max(R,G,B) min [7:0]
4	max(R,G,B) sum [31:0]			
8	max(R,G,B) bin 0 [31:0]			
	...			
1028	max(R,G,B) bin 255 [31:0]			

V4L2_META_FMT_VSP1_HGT ('VSPT')

Renesas R-Car VSP1 2-D Histogram Data

Description

This format describes histogram data generated by the Renesas R-Car VSP1 2-D Histogram (HGT) engine.

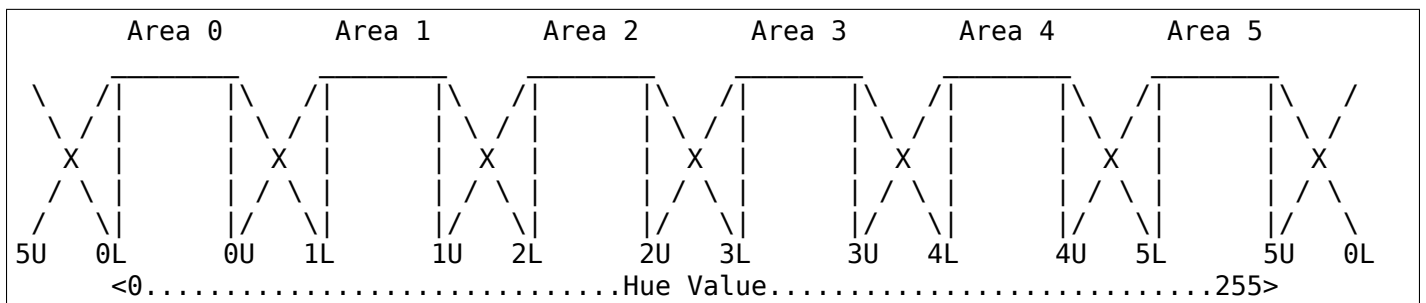
The VSP1 HGT is a histogram computation engine that operates on HSV data. It operates on a possibly cropped and subsampled input image and computes the sum, maximum and minimum of the S component as well as a weighted frequency histogram based on the H and S components.

The histogram is a matrix of 6 Hue and 32 Saturation buckets, 192 in total. Each HSV value is added to one or more buckets with a weight between 1 and 16 depending on the Hue areas configuration. Finding the corresponding buckets is done by inspecting the H and S value independently.

The Saturation position **n** (0 - 31) of the bucket in the matrix is found by the expression:

$$n = S / 8$$

The Hue position **m** (0 - 5) of the bucket in the matrix depends on how the HGT Hue areas are configured. There are 6 user configurable Hue Areas which can be configured to cover overlapping Hue values:



When two consecutive areas don't overlap ($n+1L$ is equal to nU) the boundary value is considered as part of the lower area.

Pixels with a hue value included in the centre of an area (between nL and nU included) are attributed to that single area and given a weight of 16. Pixels with a hue value included in

the overlapping region between two areas (between $n+1L$ and nU excluded) are attributed to both areas and given a weight for each of these areas proportional to their position along the diagonal lines (rounded down).

The Hue area setup must match one of the following constrains:

$$0L \leq 0U \leq 1L \leq 1U \leq 2L \leq 2U \leq 3L \leq 3U \leq 4L \leq 4U \leq 5L \leq 5U$$

$$0U \leq 1L \leq 1U \leq 2L \leq 2U \leq 3L \leq 3U \leq 4L \leq 4U \leq 5L \leq 5U \leq 0L$$

Byte Order. All data is stored in memory in little endian format. Each cell in the tables contains one byte.

Table 84: VSP1 HGT Data - (776 bytes)

Offset	Memory			
	[31:24]	[23:16]	[15:8]	[7:0]
0	.	S max [7:0]	.	S min [7:0]
4	S sum [31:0]			
8	Histogram bucket (m=0, n=0) [31:0]			
12	Histogram bucket (m=0, n=1) [31:0]			
	...			
132	Histogram bucket (m=0, n=31) [31:0]			
136	Histogram bucket (m=1, n=0) [31:0]			
	...			
264	Histogram bucket (m=2, n=0) [31:0]			
	...			
392	Histogram bucket (m=3, n=0) [31:0]			
	...			
520	Histogram bucket (m=4, n=0) [31:0]			
	...			
648	Histogram bucket (m=5, n=0) [31:0]			
	...			
772	Histogram bucket (m=5, n=31) [31:0]			

V4L2_META_FMT_VIVID ('VIVID')

VIVID Metadata Format

Description

This describes metadata format used by the vivid driver.

It sets Brightness, Saturation, Contrast and Hue, each of which maps to corresponding controls of the vivid driver with respect to the range and default values.

It contains the following fields:

Table 85: VIVID Metadata

Field	Description
u16 brightness;	Image brightness, the value is in the range 0 to 255, with the default value as 128.
u16 contrast;	Image contrast, the value is in the range 0 to 255, with the default value as 128.
u16 saturation;	Image color saturation, the value is in the range 0 to 255, with the default value as 128.
s16 hue;	Image color balance, the value is in the range -128 to 128, with the default value as 0.

3.2.2.14 Reserved Format Identifiers

These formats are not defined by this specification, they are just listed for reference and to avoid naming conflicts. If you want to register your own format, send an e-mail to the linux-media mailing list <https://linuxtv.org/lists.php> for inclusion in the videodev2.h file. If you want to share your format with other developers add a link to your documentation and send a copy to the linux-media mailing list for inclusion in this section. If you think your format should be listed in a standard format section please make a proposal on the linux-media mailing list.

Table 86: Reserved Image Formats

Identifier	Code	Details
V4L2_PIX_FMT_DV	'dvsd'	unknown
V4L2_PIX_FMT_ET61X251	'E625'	Compressed format of the ET61X251 driver.
V4L2_PIX_FMT_HI240	'HI24'	8 bit RGB format used by the BTTV driver.
V4L2_PIX_FMT_CPIA1	'CPIA'	YUV format used by the gspca cpia1 driver.
V4L2_PIX_FMT_JPGL	'JPGL'	JPEG-Light format (Pegasus Lossless JPEG) used in Divio webcams NW 80x.
V4L2_PIX_FMT_SPCA501	'S501'	YUYV per line used by the gspca driver.
V4L2_PIX_FMT_SPCA505	'S505'	YYUV per line used by the gspca driver.
V4L2_PIX_FMT_SPCA508	'S508'	YUVY per line used by the gspca driver.
V4L2_PIX_FMT_SPCA561	'S561'	Compressed GBRG Bayer format used by the gspca driver.
V4L2_PIX_FMT_PAC207	'P207'	Compressed BGGR Bayer format used by the gspca driver.
V4L2_PIX_FMT_MR97310A	'M310'	Compressed BGGR Bayer format used by the gspca driver.
V4L2_PIX_FMT_JL2005BCD	'JL20'	JPEG compressed RGGB Bayer format used by the gspca driver.
V4L2_PIX_FMT_OV511	'O511'	OV511 JPEG format used by the gspca driver.
V4L2_PIX_FMT_OV518	'O518'	OV518 JPEG format used by the gspca driver.
V4L2_PIX_FMT_PJPG	'PJPG'	Pixart 73xx JPEG format used by the gspca driver.
V4L2_PIX_FMT_SE401	'S401'	Compressed RGB format used by the gspca se401 driver
V4L2_PIX_FMT_SQ905C	'905C'	Compressed RGGB bayer format used by the gspca driver.
V4L2_PIX_FMT_MJPEG	'MJPG'	Compressed format used by the Zoran driver
V4L2_PIX_FMT_PWC1	'PWC1'	Compressed format of the PWC driver.
V4L2_PIX_FMT_PWC2	'PWC2'	Compressed format of the PWC driver.
V4L2_PIX_FMT_SN9C10X	'S910'	Compressed format of the SN9C102 driver.

Continued on next page

Table 86 - continued from previous page

Identifier	Code	Details
V4L2_PIX_FMT_SN9C20X_I420	'S920'	YUV 4:2:0 format of the gspca sn9c20x driver.
V4L2_PIX_FMT_SN9C2028	'SONX'	Compressed GBRG bayer format of the gspca sn9c2028 driver.
V4L2_PIX_FMT_STV0680	'S680'	Bayer format of the gspca stv0680 driver.
V4L2_PIX_FMT_WNVA	'WNVA'	Used by the Winnov Videum driver, http://www.thedirks.org/winnov/
V4L2_PIX_FMT_TM6000	'TM60'	Used by Trident tm6000
V4L2_PIX_FMT_CIT_YYVYUY	'CITV'	Used by xirlink CIT, found at IBM webcams. Uses one line of Y then 1 line of VYUY
V4L2_PIX_FMT_KONICA420	'KONI'	Used by Konica webcams. YUV420 planar in blocks of 256 pixels.
V4L2_PIX_FMT_YYUV	'YYUV'	unknown
V4L2_PIX_FMT_Y4	'Y04 '	Old 4-bit greyscale format. Only the most significant 4 bits of each byte are used, the other bits are set to 0.
V4L2_PIX_FMT_Y6	'Y06 '	Old 6-bit greyscale format. Only the most significant 6 bits of each byte are used, the other bits are set to 0.
V4L2_PIX_FMT_S5C_UYVY_JPG	'S5CI'	<p>Two-planar format used by Samsung S5C73MX cameras. The first plane contains interleaved JPEG and UYVY image data, followed by meta data in form of an array of offsets to the UYVY data blocks. The actual pointer array follows immediately the interleaved JPEG/UYVY data, the number of entries in this array equals the height of the UYVY image. Each entry is a 4-byte unsigned integer in big endian order and it's an offset to a single pixel line of the UYVY image. The first plane can start either with JPEG or UYVY data chunk. The size of a single UYVY block equals the UYVY image's width multiplied by 2. The size of a JPEG chunk depends on the image and can vary with each line.</p> <p>The second plane, at an offset of 4084 bytes, contains a 4-byte offset to the pointer array in the first plane. This offset is followed by a 4-byte value indicating size of the pointer array. All numbers in the second plane are also in big endian order. Remaining data in the second plane is undefined. The information in the second plane allows to easily find location of the pointer array, which can be different for each frame. The size of the pointer array is constant for given UYVY image height.</p> <p>In order to extract UYVY and JPEG frames an application can initially set a data pointer to the start of first plane and then add an offset from the first entry of the pointers table. Such a pointer indicates start of an UYVY image pixel line. Whole UYVY line can be copied to a separate buffer. These steps should be repeated for each line, i.e. the number of entries in the pointer array. Anything what's in between the UYVY lines is JPEG data and should be concatenated to form the JPEG stream.</p>

Continued on next page

Table 86 - continued from previous page

Identifier	Code	Details
V4L2_PIX_FMT_MT21C	'MT21'	Compressed two-planar YVU420 format used by Mediatek MT8173. The compression is lossless. It is an opaque intermediate format and the MDP hardware must be used to convert V4L2_PIX_FMT_MT21C to V4L2_PIX_FMT_NV12M, V4L2_PIX_FMT_YUV420M or V4L2_PIX_FMT_YVU420.
V4L2_PIX_FMT_MM21	'MM21'	Non-compressed, tiled two-planar format used by Mediatek MT8183. This is an opaque intermediate format and the MDP3 hardware can be used to convert it to other formats.

3.2.2.15 Colorspaces

'Color' is a very complex concept and depends on physics, chemistry and biology. Just because you have three numbers that describe the 'red', 'green' and 'blue' components of the color of a pixel does not mean that you can accurately display that color. A colorspace defines what it actually *means* to have an RGB value of e.g. (255, 0, 0). That is, which color should be reproduced on the screen in a perfectly calibrated environment.

In order to do that we first need to have a good definition of color, i.e. some way to uniquely and unambiguously define a color so that someone else can reproduce it. Human color vision is trichromatic since the human eye has color receptors that are sensitive to three different wavelengths of light. Hence the need to use three numbers to describe color. Be glad you are not a mantis shrimp as those are sensitive to 12 different wavelengths, so instead of RGB we would be using the ABCDEFGHIJKL colorspace...

Color exists only in the eye and brain and is the result of how strongly color receptors are stimulated. This is based on the Spectral Power Distribution (SPD) which is a graph showing the intensity (radiant power) of the light at wavelengths covering the visible spectrum as it enters the eye. The science of colorimetry is about the relationship between the SPD and color as perceived by the human brain.

Since the human eye has only three color receptors it is perfectly possible that different SPDs will result in the same stimulation of those receptors and are perceived as the same color, even though the SPD of the light is different.

In the 1920s experiments were devised to determine the relationship between SPDs and the perceived color and that resulted in the CIE 1931 standard that defines spectral weighting functions that model the perception of color. Specifically that standard defines functions that can take an SPD and calculate the stimulus for each color receptor. After some further mathematical transforms these stimuli are known as the *CIE XYZ tristimulus* values and these X, Y and Z values describe a color as perceived by a human unambiguously. These X, Y and Z values are all in the range [0...1].

The Y value in the CIE XYZ colorspace corresponds to luminance. Often the CIE XYZ colorspace is transformed to the normalized CIE xyY colorspace:

$$x = X / (X + Y + Z)$$

$$y = Y / (X + Y + Z)$$

The x and y values are the chromaticity coordinates and can be used to define a color without the luminance component Y. It is very confusing to have such similar names for these colorspace.

Just be aware that if colors are specified with lower case 'x' and 'y', then the CIE xyY colorspace is used. Upper case 'X' and 'Y' refer to the CIE XYZ colorspace. Also, y has nothing to do with luminance. Together x and y specify a color, and Y the luminance. That is really all you need to remember from a practical point of view. At the end of this section you will find reading resources that go into much more detail if you are interested.

A monitor or TV will reproduce colors by emitting light at three different wavelengths, the combination of which will stimulate the color receptors in the eye and thus cause the perception of color. Historically these wavelengths were defined by the red, green and blue phosphors used in the displays. These *color primaries* are part of what defines a colorspace.

Different display devices will have different primaries and some primaries are more suitable for some display technologies than others. This has resulted in a variety of colorspace that are used for different display technologies or uses. To define a colorspace you need to define the three color primaries (these are typically defined as x, y chromaticity coordinates from the CIE xyY colorspace) but also the white reference: that is the color obtained when all three primaries are at maximum power. This determines the relative power or energy of the primaries. This is usually chosen to be close to daylight which has been defined as the CIE D65 Illuminant.

To recapitulate: the CIE XYZ colorspace uniquely identifies colors. Other colorspace are defined by three chromaticity coordinates defined in the CIE xyY colorspace. Based on those a 3x3 matrix can be constructed that transforms CIE XYZ colors to colors in the new colorspace.

Both the CIE XYZ and the RGB colorspace that are derived from the specific chromaticity primaries are linear colorspace. But neither the eye, nor display technology is linear. Doubling the values of all components in the linear colorspace will not be perceived as twice the intensity of the color. So each colorspace also defines a transfer function that takes a linear color component value and transforms it to the non-linear component value, which is a closer match to the non-linear performance of both the eye and displays. Linear component values are denoted RGB, non-linear are denoted as R'G'B'. In general colors used in graphics are all R'G'B', except in OpenGL which uses linear RGB. Special care should be taken when dealing with OpenGL to provide linear RGB colors or to use the built-in OpenGL support to apply the inverse transfer function.

The final piece that defines a colorspace is a function that transforms non-linear R'G'B' to non-linear Y'CbCr. This function is determined by the so-called luma coefficients. There may be multiple possible Y'CbCr encodings allowed for the same colorspace. Many encodings of color prefer to use luma (Y') and chroma (CbCr) instead of R'G'B'. Since the human eye is more sensitive to differences in luminance than in color this encoding allows one to reduce the amount of color information compared to the luma data. Note that the luma (Y') is unrelated to the Y in the CIE XYZ colorspace. Also note that Y'CbCr is often called YCbCr or YUV even though these are strictly speaking wrong.

Sometimes people confuse Y'CbCr as being a colorspace. This is not correct, it is just an encoding of an R'G'B' color into luma and chroma values. The underlying colorspace that is associated with the R'G'B' color is also associated with the Y'CbCr color.

The final step is how the RGB, R'G'B' or Y'CbCr values are quantized. The CIE XYZ colorspace where X, Y and Z are in the range [0...1] describes all colors that humans can perceive, but the transform to another colorspace will produce colors that are outside the [0...1] range. Once clamped to the [0...1] range those colors can no longer be reproduced in that colorspace. This clamping is what reduces the extent or gamut of the colorspace. How the range of [0...1] is translated to integer values in the range of [0...255] (or higher, depending on the color depth) is called the quantization. This is *not* part of the colorspace definition. In practice RGB or R'G'B' values are full range, i.e. they use the full [0...255] range. Y'CbCr values on the other

hand are limited range with Y' using [16...235] and Cb and Cr using [16...240].

Unfortunately, in some cases limited range RGB is also used where the components use the range [16...235]. And full range Y'CbCr also exists using the [0...255] range.

In order to correctly interpret a color you need to know the quantization range, whether it is R'G'B' or Y'CbCr, the used Y'CbCr encoding and the colorspace. From that information you can calculate the corresponding CIE XYZ color and map that again to whatever colorspace your display device uses.

The colorspace definition itself consists of the three chromaticity primaries, the white reference chromaticity, a transfer function and the luma coefficients needed to transform R'G'B' to Y'CbCr. While some colorspace standards correctly define all four, quite often the colorspace standard only defines some, and you have to rely on other standards for the missing pieces. The fact that colorspace standards are often a mix of different standards also led to very confusing naming conventions where the name of a standard was used to name a colorspace when in fact that standard was part of various other colorspace standards as well.

If you want to read more about colors and colorspace standards, then the following resources are useful: *poynnton* is a good practical book for video engineers, *colimg* has a much broader scope and describes many more aspects of color (physics, chemistry, biology, etc.). The <http://www.brucelindbloom.com> website is an excellent resource, especially with respect to the mathematics behind colorspace conversions. The wikipedia [CIE 1931 colorspace](#) article is also very useful.

3.2.2.16 Defining Colorspaces in V4L2

In V4L2 colorspace standards are defined by four values. The first is the colorspace identifier (enum *v4l2_colorspace*) which defines the chromaticities, the default transfer function, the default Y'CbCr encoding and the default quantization method. The second is the transfer function identifier (enum *v4l2_xfer_func*) to specify non-standard transfer functions. The third is the Y'CbCr encoding identifier (enum *v4l2_ycbcr_encoding*) to specify non-standard Y'CbCr encodings and the fourth is the quantization identifier (enum *v4l2_quantization*) to specify non-standard quantization methods. Most of the time only the colorspace field of struct *v4l2_pix_format* or struct *v4l2_pix_format_plane* needs to be filled in.

On *HSV formats* the *Hue* is defined as the angle on the cylindrical color representation. Usually this angle is measured in degrees, i.e. 0-360. When we map this angle value into 8 bits, there are two basic ways to do it: Divide the angular value by 2 (0-179), or use the whole range, 0-255, dividing the angular value by 1.41. The enum *v4l2_hsv_encoding* specifies which encoding is used.

Note: The default R'G'B' quantization is full range for all colorspace standards. HSV formats are always full range.

v4l2_colorspace

Table 87: V4L2 Colorspaces

Identifier	Details
V4L2_COLORSPACE_DEFAULT	The default colorspace. This can be used by applications to let the driver fill in the colorspace.
V4L2_COLORSPACE_SMPTE170M	See <i>Colorspace SMPTE 170M</i> (V4L2_COLORSPACE_SMPTE170M).
V4L2_COLORSPACE_REC709	See <i>Colorspace Rec. 709</i> (V4L2_COLORSPACE_REC709).
V4L2_COLORSPACE_SRGB	See <i>Colorspace sRGB</i> (V4L2_COLORSPACE_SRGB).
V4L2_COLORSPACE_OPRGB	See <i>Colorspace opRGB</i> (V4L2_COLORSPACE_OPRGB).
V4L2_COLORSPACE_BT2020	See <i>Colorspace BT.2020</i> (V4L2_COLORSPACE_BT2020).
V4L2_COLORSPACE_DCI_P3	See <i>Colorspace DCI-P3</i> (V4L2_COLORSPACE_DCI_P3).
V4L2_COLORSPACE_SMPTE240M	See <i>Colorspace SMPTE 240M</i> (V4L2_COLORSPACE_SMPTE240M).
V4L2_COLORSPACE_470_SYSTEM_M	See <i>Colorspace NTSC 1953</i> (V4L2_COLORSPACE_470_SYSTEM_M).
V4L2_COLORSPACE_470_SYSTEM_BG	See <i>Colorspace EBU Tech. 3213</i> (V4L2_COLORSPACE_470_SYSTEM_BG).
V4L2_COLORSPACE_JPEG	See <i>Colorspace JPEG</i> (V4L2_COLORSPACE_JPEG).
V4L2_COLORSPACE_RAW	The raw colorspace. This is used for raw image capture where the image is minimally processed and is using the internal colorspace of the device. The software that processes an image using this 'colorspace' will have to know the internals of the capture device.

v4l2_xfer_func

Table 88: V4L2 Transfer Function

Identifier	Details
V4L2_XFER_FUNC_DEFAULT	Use the default transfer function as defined by the colorspace.
V4L2_XFER_FUNC_709	Use the Rec. 709 transfer function.
V4L2_XFER_FUNC_SRGB	Use the sRGB transfer function.
V4L2_XFER_FUNC_OPRGB	Use the opRGB transfer function.
V4L2_XFER_FUNC_SMPTE240M	Use the SMPTE 240M transfer function.
V4L2_XFER_FUNC_NONE	Do not use a transfer function (i.e. use linear RGB values).
V4L2_XFER_FUNC_DCI_P3	Use the DCI-P3 transfer function.
V4L2_XFER_FUNC_SMPTE2084	Use the SMPTE 2084 transfer function. See <i>Transfer Function SMPTE 2084</i> (V4L2_XFER_FUNC_SMPTE2084).

v4l2_ycbcr_encoding

Table 89: V4L2 Y'CbCr Encodings

Identifier	Details
V4L2_YCBCR_ENC_DEFAULT	Use the default Y'CbCr encoding as defined by the colorspace.
V4L2_YCBCR_ENC_601	Use the BT.601 Y'CbCr encoding.
V4L2_YCBCR_ENC_709	Use the Rec. 709 Y'CbCr encoding.
V4L2_YCBCR_ENC_XV601	Use the extended gamut xvYCC BT.601 encoding.
V4L2_YCBCR_ENC_XV709	Use the extended gamut xvYCC Rec. 709 encoding.
V4L2_YCBCR_ENC_BT2020	Use the default non-constant luminance BT.2020 Y'CbCr encoding.
V4L2_YCBCR_ENC_BT2020_CONST_LUM	Use the constant luminance BT.2020 Yc'CbCrCrc encoding.
V4L2_YCBCR_ENC_SMPTE_240M	Use the SMPTE 240M Y'CbCr encoding.

v4l2_hsv_encoding

Table 90: V4L2 HSV Encodings

Identifier	Details
V4L2_HSV_ENC_180	For the Hue, each LSB is two degrees.
V4L2_HSV_ENC_256	For the Hue, the 360 degrees are mapped into 8 bits, i.e. each LSB is roughly 1.41 degrees.

v4l2_quantization

Table 91: V4L2 Quantization Methods

Identifier	Details
V4L2_QUANTIZATION_DEFAULT	Use the default quantization encoding as defined by the colorspace. This is always full range for R'G'B' and HSV. It is usually limited range for Y'CbCr.
V4L2_QUANTIZATION_FULL_RANGE	Use the full range quantization encoding. I.e. the range [0...1] is mapped to [0...255] (with possible clipping to [1...254] to avoid the 0x00 and 0xff values). Cb and Cr are mapped from [-0.5...0.5] to [0...255] (with possible clipping to [1...254] to avoid the 0x00 and 0xff values).
V4L2_QUANTIZATION_LIM_RANGE	Use the limited range quantization encoding. I.e. the range [0...1] is mapped to [16...235]. Cb and Cr are mapped from [-0.5...0.5] to [16...240]. Limited Range cannot be used with HSV.

3.2.2.17 Detailed Colorspace Descriptions

Colorspace SMPTE 170M (V4L2_COLORSPACE_SMPTE170M)

The *SMPTE 170M* standard defines the colorspace used by NTSC and PAL and by SDTV in general. The default transfer function is `V4L2_XFER_FUNC_709`. The default Y'CbCr encoding is `V4L2_YCBCR_ENC_601`. The default Y'CbCr quantization is limited range. The chromaticities of the primary colors and the white reference are:

Table 92: SMPTE 170M Chromaticities

Color	x	y
Red	0.630	0.340
Green	0.310	0.595
Blue	0.155	0.070
White Reference (D65)	0.3127	0.3290

The red, green and blue chromaticities are also often referred to as the SMPTE C set, so this colorspace is sometimes called SMPTE C as well.

The transfer function defined for SMPTE 170M is the same as the one defined in Rec. 709.

$$L' = -1.099(-L)^{0.45} + 0.099, \text{ for } L \leq -0.018$$

$$L' = 4.5L, \text{ for } -0.018 < L < 0.018$$

$$L' = 1.099L^{0.45} - 0.099, \text{ for } L \geq 0.018$$

Inverse Transfer function:

$$L = -\left(\frac{L' - 0.099}{-1.099}\right)^{\frac{1}{0.45}}, \text{ for } L' \leq -0.081$$

$$L = \frac{L'}{4.5}, \text{ for } -0.081 < L' < 0.081$$

$$L = \left(\frac{L' + 0.099}{1.099}\right)^{\frac{1}{0.45}}, \text{ for } L' \geq 0.081$$

The luminance (Y') and color difference (Cb and Cr) are obtained with the following `V4L2_YCBCR_ENC_601` encoding:

$$Y' = 0.2990R' + 0.5870G' + 0.1140B'$$

$$Cb = -0.1687R' - 0.3313G' + 0.5B'$$

$$Cr = 0.5R' - 0.4187G' - 0.0813B'$$

Y' is clamped to the range [0...1] and Cb and Cr are clamped to the range [-0.5...0.5]. This conversion to Y'CbCr is identical to the one defined in the *ITU BT.601* standard and this colorspace is sometimes called BT.601 as well, even though BT.601 does not mention any color primaries.

The default quantization is limited range, but full range is possible although rarely seen.

Colorspace Rec. 709 (V4L2_COLORSPACE_REC709)

The *ITU BT.709* standard defines the colorspace used by HDTV in general. The default transfer function is `V4L2_XFER_FUNC_709`. The default Y'CbCr encoding is `V4L2_YCBCR_ENC_709`. The default Y'CbCr quantization is limited range. The chromaticities of the primary colors and the white reference are:

Table 93: Rec. 709 Chromaticities

Color	x	y
Red	0.640	0.330
Green	0.300	0.600
Blue	0.150	0.060
White Reference (D65)	0.3127	0.3290

The full name of this standard is Rec. ITU-R BT.709-5.

Transfer function. Normally L is in the range [0...1], but for the extended gamut xvYCC encoding values outside that range are allowed.

$$L' = -1.099(-L)^{0.45} + 0.099, \text{ for } L \leq -0.018$$

$$L' = 4.5L, \text{ for } -0.018 < L < 0.018$$

$$L' = 1.099L^{0.45} - 0.099, \text{ for } L \geq 0.018$$

Inverse Transfer function:

$$L = -\left(\frac{L' - 0.099}{-1.099}\right)^{\frac{1}{0.45}}, \text{ for } L' \leq -0.081$$

$$L = \frac{L'}{4.5}, \text{ for } -0.081 < L' < 0.081$$

$$L = \left(\frac{L' + 0.099}{1.099}\right)^{\frac{1}{0.45}}, \text{ for } L' \geq 0.081$$

The luminance (Y') and color difference (Cb and Cr) are obtained with the following `V4L2_YCBCR_ENC_709` encoding:

$$Y' = 0.2126R' + 0.7152G' + 0.0722B'$$

$$Cb = -0.1146R' - 0.3854G' + 0.5B'$$

$$Cr = 0.5R' - 0.4542G' - 0.0458B'$$

Y' is clamped to the range [0...1] and Cb and Cr are clamped to the range [-0.5...0.5].

The default quantization is limited range, but full range is possible although rarely seen.

The `V4L2_YCBCR_ENC_709` encoding described above is the default for this colorspace, but it can be overridden with `V4L2_YCBCR_ENC_601`, in which case the BT.601 Y'CbCr encoding is used.

Two additional extended gamut Y'CbCr encodings are also possible with this colorspace:

The xvYCC 709 encoding (`V4L2_YCBCR_ENC_XV709`, *xvYCC*) is similar to the Rec. 709 encoding, but it allows for R', G' and B' values that are outside the range [0...1]. The resulting Y', Cb and

Cr values are scaled and offset according to the limited range formula:

$$Y' = \frac{219}{256} * (0.2126R' + 0.7152G' + 0.0722B') + \frac{16}{256}$$

$$Cb = \frac{224}{256} * (-0.1146R' - 0.3854G' + 0.5B')$$

$$Cr = \frac{224}{256} * (0.5R' - 0.4542G' - 0.0458B')$$

The xvYCC 601 encoding (V4L2_YCBCR_ENC_XV601, *xvYCC*) is similar to the BT.601 encoding, but it allows for R', G' and B' values that are outside the range [0...1]. The resulting Y', Cb and Cr values are scaled and offset according to the limited range formula:

$$Y' = \frac{219}{256} * (0.2990R' + 0.5870G' + 0.1140B') + \frac{16}{256}$$

$$Cb = \frac{224}{256} * (-0.1687R' - 0.3313G' + 0.5B')$$

$$Cr = \frac{224}{256} * (0.5R' - 0.4187G' - 0.0813B')$$

Y' is clamped to the range [0...1] and Cb and Cr are clamped to the range [-0.5...0.5] and quantized without further scaling or offsets. The non-standard xvYCC 709 or xvYCC 601 encodings can be used by selecting V4L2_YCBCR_ENC_XV709 or V4L2_YCBCR_ENC_XV601. As seen by the xvYCC formulas these encodings always use limited range quantization, there is no full range variant. The whole point of these extended gamut encodings is that values outside the limited range are still valid, although they map to R', G' and B' values outside the [0...1] range and are therefore outside the Rec. 709 colorspace gamut.

Colorspace sRGB (V4L2_COLORSPACE_SRGB)

The *sRGB* standard defines the colorspace used by most webcams and computer graphics. The default transfer function is V4L2_XFER_FUNC_SRGB. The default Y'CbCr encoding is V4L2_YCBCR_ENC_601. The default Y'CbCr quantization is limited range.

Note that the *sYCC* standard specifies full range quantization, however all current capture hardware supported by the kernel convert R'G'B' to limited range Y'CbCr. So choosing full range as the default would break how applications interpret the quantization range.

The chromaticities of the primary colors and the white reference are:

Table 94: sRGB Chromaticities

Color	x	y
Red	0.640	0.330
Green	0.300	0.600
Blue	0.150	0.060
White Reference (D65)	0.3127	0.3290

These chromaticities are identical to the Rec. 709 colorspace.

Transfer function. Note that negative values for L are only used by the Y'CbCr conversion.

$$L' = -1.055(-L)^{\frac{1}{2.4}} + 0.055, \text{ for } L < -0.0031308$$

$$L' = 12.92L, \text{ for } -0.0031308 \leq L \leq 0.0031308$$

$$L' = 1.055L^{\frac{1}{2.4}} - 0.055, \text{ for } 0.0031308 < L \leq 1$$

Inverse Transfer function:

$$L = -((-L' + 0.055)/1.055)^{2.4}, \text{ for } L' < -0.04045$$

$$L = L'/12.92, \text{ for } -0.04045 \leq L' \leq 0.04045$$

$$L = ((L' + 0.055)/1.055)^{2.4}, \text{ for } L' > 0.04045$$

The luminance (Y') and color difference (Cb and Cr) are obtained with the following V4L2_YCBCR_ENC_601 encoding as defined by *sYCC*:

$$Y' = 0.2990R' + 0.5870G' + 0.1140B'$$

$$Cb = -0.1687R' - 0.3313G' + 0.5B'$$

$$Cr = 0.5R' - 0.4187G' - 0.0813B'$$

Y' is clamped to the range [0...1] and Cb and Cr are clamped to the range [-0.5...0.5]. This transform is identical to one defined in SMPTE 170M/BT.601. The Y'CbCr quantization is limited range.

Colorspace opRGB (V4L2_COLORSPACE_OPRGB)

The *opRGB* standard defines the colorspace used by computer graphics that use the opRGB colorspace. The default transfer function is V4L2_XFER_FUNC_OPRGB. The default Y'CbCr encoding is V4L2_YCBCR_ENC_601. The default Y'CbCr quantization is limited range.

Note that the *opRGB* standard specifies full range quantization, however all current capture hardware supported by the kernel convert R'G'B' to limited range Y'CbCr. So choosing full range as the default would break how applications interpret the quantization range.

The chromaticities of the primary colors and the white reference are:

Table 95: opRGB Chromaticities

Color	x	y
Red	0.6400	0.3300
Green	0.2100	0.7100
Blue	0.1500	0.0600
White Reference (D65)	0.3127	0.3290

Transfer function:

$$L' = L^{\frac{1}{2.19921875}}$$

Inverse Transfer function:

$$L = L'^{(2.19921875)}$$

The luminance (Y') and color difference (Cb and Cr) are obtained with the following

V4L2_YCBCR_ENC_601 encoding:

$$\begin{aligned} Y' &= 0.2990R' + 0.5870G' + 0.1140B' \\ Cb &= -0.1687R' - 0.3313G' + 0.5B' \\ Cr &= 0.5R' - 0.4187G' - 0.0813B' \end{aligned}$$

Y' is clamped to the range [0...1] and Cb and Cr are clamped to the range [-0.5...0.5]. This transform is identical to one defined in SMPTE 170M/BT.601. The $Y'CbCr$ quantization is limited range.

Colorspace BT.2020 (V4L2_COLORSPACE_BT2020)

The *ITU BT.2020* standard defines the colorspace used by Ultra-high definition television (UHDTV). The default transfer function is V4L2_XFER_FUNC_709. The default $Y'CbCr$ encoding is V4L2_YCBCR_ENC_BT2020. The default $Y'CbCr$ quantization is limited range. The chromaticities of the primary colors and the white reference are:

Table 96: BT.2020 Chromaticities

Color	x	y
Red	0.708	0.292
Green	0.170	0.797
Blue	0.131	0.046
White Reference (D65)	0.3127	0.3290

Transfer function (same as Rec. 709):

$$\begin{aligned} L' &= 4.5L, \text{ for } 0 \leq L < 0.018 \\ L' &= 1.099L^{0.45} - 0.099, \text{ for } 0.018 \leq L \leq 1 \end{aligned}$$

Inverse Transfer function:

$$\begin{aligned} L &= L'/4.5, \text{ for } L' < 0.081 \\ L &= \left(\frac{L' + 0.099}{1.099} \right)^{\frac{1}{0.45}}, \text{ for } L' \geq 0.081 \end{aligned}$$

Please note that while Rec. 709 is defined as the default transfer function by the *ITU BT.2020* standard, in practice this colorspace is often used with the *Transfer Function SMPTE 2084 (V4L2_XFER_FUNC_SMPTE2084)*. In particular Ultra HD Blu-ray discs use this combination.

The luminance (Y') and color difference (Cb and Cr) are obtained with the following V4L2_YCBCR_ENC_BT2020 encoding:

$$\begin{aligned} Y' &= 0.2627R' + 0.6780G' + 0.0593B' \\ Cb &= -0.1396R' - 0.3604G' + 0.5B' \\ Cr &= 0.5R' - 0.4598G' - 0.0402B' \end{aligned}$$

Y' is clamped to the range [0...1] and Cb and Cr are clamped to the range [-0.5...0.5]. The $Y'CbCr$ quantization is limited range.

There is also an alternate constant luminance R'G'B' to Yc'CbCrCrc (V4L2_YCBCR_ENC_BT2020_CONST_LUM) encoding:

Luma:

$$Yc' = (0.2627R + 0.6780G + 0.0593B)'$$

$$B' - Yc' \leq 0 : Cbc = (B' - Yc')/1.9404$$

$$B' - Yc' > 0 : Cbc = (B' - Yc')/1.5816$$

$$R' - Yc' \leq 0 : Crc = (R' - Yc')/1.7184$$

$$R' - Yc' > 0 : Crc = (R' - Yc')/0.9936$$

Yc' is clamped to the range [0...1] and Cbc and Crc are clamped to the range [-0.5...0.5]. The $Yc'CbcCrc$ quantization is limited range.

Colorspace DCI-P3 (V4L2_COLORSPACE_DCI_P3)

The *SMPTE RP 431-2* standard defines the colorspace used by cinema projectors that use the DCI-P3 colorspace. The default transfer function is `V4L2_XFER_FUNC_DCI_P3`. The default $Y'CbCr$ encoding is `V4L2_YCBCR_ENC_709`. The default $Y'CbCr$ quantization is limited range.

Note: Note that this colorspace standard does not specify a $Y'CbCr$ encoding since it is not meant to be encoded to $Y'CbCr$. So this default $Y'CbCr$ encoding was picked because it is the HDTV encoding.

The chromaticities of the primary colors and the white reference are:

Table 97: DCI-P3 Chromaticities

Color	x	y
Red	0.6800	0.3200
Green	0.2650	0.6900
Blue	0.1500	0.0600
White Reference	0.3140	0.3510

Transfer function:

$$L' = L^{\frac{1}{2.6}}$$

Inverse Transfer function:

$$L = L'^{(2.6)}$$

$Y'CbCr$ encoding is not specified. V4L2 defaults to Rec. 709.

Colorspace SMPTE 240M (V4L2_COLORSPACE_SMPTE240M)

The *SMPTE 240M* standard was an interim standard used during the early days of HDTV (1988-1998). It has been superseded by Rec. 709. The default transfer function is `V4L2_XFER_FUNC_SMPTE240M`. The default Y'CbCr encoding is `V4L2_YCBCR_ENC_SMPTE240M`. The default Y'CbCr quantization is limited range. The chromaticities of the primary colors and the white reference are:

Table 98: SMPTE 240M Chromaticities

Color	x	y
Red	0.630	0.340
Green	0.310	0.595
Blue	0.155	0.070
White Reference (D65)	0.3127	0.3290

These chromaticities are identical to the SMPTE 170M colorspace.

Transfer function:

$$L' = 4L, \text{ for } 0 \leq L < 0.0228$$

$$L' = 1.1115L^{0.45} - 0.1115, \text{ for } 0.0228 \leq L \leq 1$$

Inverse Transfer function:

$$L = \frac{L'}{4}, \text{ for } 0 \leq L' < 0.0913$$

$$L = \left(\frac{L' + 0.1115}{1.1115} \right)^{\frac{1}{0.45}}, \text{ for } L' \geq 0.0913$$

The luminance (Y') and color difference (Cb and Cr) are obtained with the following `V4L2_YCBCR_ENC_SMPTE240M` encoding:

$$Y' = 0.2122R' + 0.7013G' + 0.0865B'$$

$$Cb = -0.1161R' - 0.3839G' + 0.5B'$$

$$Cr = 0.5R' - 0.4451G' - 0.0549B'$$

Y' is clamped to the range [0...1] and Cb and Cr are clamped to the range [-0.5...0.5]. The Y'CbCr quantization is limited range.

Colorspace NTSC 1953 (V4L2_COLORSPACE_470_SYSTEM_M)

This standard defines the colorspace used by NTSC in 1953. In practice this colorspace is obsolete and SMPTE 170M should be used instead. The default transfer function is `V4L2_XFER_FUNC_709`. The default Y'CbCr encoding is `V4L2_YCBCR_ENC_601`. The default Y'CbCr quantization is limited range. The chromaticities of the primary colors and the white reference are:

Table 99: NTSC 1953 Chromaticities

Color	x	y
Red	0.67	0.33
Green	0.21	0.71
Blue	0.14	0.08
White Reference (C)	0.310	0.316

Note: This colorspace uses Illuminant C instead of D65 as the white reference. To correctly convert an image in this colorspace to another that uses D65 you need to apply a chromatic adaptation algorithm such as the Bradford method.

The transfer function was never properly defined for NTSC 1953. The Rec. 709 transfer function is recommended in the literature:

$$L' = 4.5L, \text{ for } 0 \leq L < 0.018$$

$$L' = 1.099L^{0.45} - 0.099, \text{ for } 0.018 \leq L \leq 1$$

Inverse Transfer function:

$$L = \frac{L'}{4.5}, \text{ for } L' < 0.081$$

$$L = \left(\frac{L' + 0.099}{1.099} \right)^{\frac{1}{0.45}}, \text{ for } L' \geq 0.081$$

The luminance (Y') and color difference (Cb and Cr) are obtained with the following V4L2_YCBCR_ENC_601 encoding:

$$Y' = 0.2990R' + 0.5870G' + 0.1140B'$$

$$Cb = -0.1687R' - 0.3313G' + 0.5B'$$

$$Cr = 0.5R' - 0.4187G' - 0.0813B'$$

Y' is clamped to the range $[0...1]$ and Cb and Cr are clamped to the range $[-0.5...0.5]$. The $Y'CbCr$ quantization is limited range. This transform is identical to one defined in SMPTE 170M/BT.601.

Colorspace EBU Tech. 3213 (V4L2_COLORSPACE_470_SYSTEM_BG)

The *EBU Tech 3213* standard defines the colorspace used by PAL/SECAM in 1975. Note that this colorspace is not supported by the HDMI interface. Instead *EBU Tech 3321* recommends that Rec. 709 is used instead for HDMI. The default transfer function is V4L2_XFER_FUNC_709. The default $Y'CbCr$ encoding is V4L2_YCBCR_ENC_601. The default $Y'CbCr$ quantization is limited range. The chromaticities of the primary colors and the white reference are:

Table 100: EBU Tech. 3213 Chromaticities

Color	x	y
Red	0.64	0.33
Green	0.29	0.60
Blue	0.15	0.06
White Reference (D65)	0.3127	0.3290

The transfer function was never properly defined for this colorspace. The Rec. 709 transfer function is recommended in the literature:

$$L' = 4.5L, \text{ for } 0 \leq L < 0.018$$
$$L' = 1.099L^{0.45} - 0.099, \text{ for } 0.018 \leq L \leq 1$$

Inverse Transfer function:

$$L = \frac{L'}{4.5}, \text{ for } L' < 0.081$$
$$L = \left(\frac{L' + 0.099}{1.099} \right)^{\frac{1}{0.45}}, \text{ for } L' \geq 0.081$$

The luminance (Y') and color difference (Cb and Cr) are obtained with the following V4L2_YCBCR_ENC_601 encoding:

$$Y' = 0.2990R' + 0.5870G' + 0.1140B'$$
$$Cb = -0.1687R' - 0.3313G' + 0.5B'$$
$$Cr = 0.5R' - 0.4187G' - 0.0813B'$$

Y' is clamped to the range [0...1] and Cb and Cr are clamped to the range [-0.5...0.5]. The Y' CbCr quantization is limited range. This transform is identical to one defined in SMPTE 170M/BT.601.

Colorspace JPEG (V4L2_COLORSPACE_JPEG)

This colorspace defines the colorspace used by most (Motion-)JPEG formats. The chromaticities of the primary colors and the white reference are identical to sRGB. The transfer function use is V4L2_XFER_FUNC_SRGB. The Y' CbCr encoding is V4L2_YCBCR_ENC_601 with full range quantization where Y' is scaled to [0...255] and Cb/Cr are scaled to [-128...128] and then clipped to [-128...127].

Note: The JPEG standard does not actually store colorspace information. So if something other than sRGB is used, then the driver will have to set that information explicitly. Effectively V4L2_COLORSPACE_JPEG can be considered to be an abbreviation for V4L2_COLORSPACE_SRGB, V4L2_XFER_FUNC_SRGB, V4L2_YCBCR_ENC_601 and V4L2_QUANTIZATION_FULL_RANGE.

3.2.2.18 Detailed Transfer Function Descriptions

Transfer Function SMPTE 2084 (V4L2_XFER_FUNC_SMPTE2084)

The *SMPTE ST 2084* standard defines the transfer function used by High Dynamic Range content.

Constants: $m1 = (2610 / 4096) / 4$

$$m2 = (2523 / 4096) * 128$$

$$c1 = 3424 / 4096$$

$$c2 = (2413 / 4096) * 32$$

$$c3 = (2392 / 4096) * 32$$

Transfer function: $L' = ((c1 + c2 * L^{m1}) / (1 + c3 * L^{m1}))^{m2}$

Inverse Transfer function: $L = (\max(L'^{1/m2} - c1, 0) / (c2 - c3 * L'^{1/m2}))^{1/m1}$

Take care when converting between this transfer function and non-HDR transfer functions: the linear RGB values [0...1] of HDR content map to a luminance range of 0 to 10000 cd/m² whereas the linear RGB values of non-HDR (aka Standard Dynamic Range or SDR) map to a luminance range of 0 to 100 cd/m².

To go from SDR to HDR you will have to divide L by 100 first. To go in the other direction you will have to multiply L by 100. Of course, this clamps all luminance values over 100 cd/m² to 100 cd/m².

There are better methods, see e.g. *colimg* for more in-depth information about this.

3.2.3 Input/Output

The V4L2 API defines several different methods to read from or write to a device. All drivers exchanging data with applications must support at least one of them.

The classic I/O method using the `read()` and `write()` function is automatically selected after opening a V4L2 device. When the driver does not support this method attempts to read or write will fail at any time.

Other methods must be negotiated. To select the streaming I/O method with memory mapped or user buffers applications call the *ioctl VIDIOC_REQBUFS* ioctl. The asynchronous I/O method is not defined yet.

Video overlay can be considered another I/O method, although the application does not directly receive the image data. It is selected by initiating video overlay with the *VIDIOC_S_FMT* ioctl. For more information see *Video Overlay Interface*.

Generally exactly one I/O method, including overlay, is associated with each file descriptor. The only exceptions are applications not exchanging data with a driver (“panel applications”, see *Opening and Closing Devices*) and drivers permitting simultaneous video capturing and overlay using the same file descriptor, for compatibility with V4L and earlier versions of V4L2.

VIDIOC_S_FMT and *ioctl VIDIOC_REQBUFS* would permit this to some degree, but for simplicity drivers need not support switching the I/O method (after first switching away from read/write) other than by closing and reopening the device.

The following sections describe the various I/O methods in more detail.

3.2.3.1 Read/Write

Input and output devices support the `read()` and `write()` function, respectively, when the `V4L2_CAP_READWRITE` flag in the capabilities field of struct `v4l2_capability` returned by the *ioctl VIDIOC_QUERYCAP* ioctl is set.

Drivers may need the CPU to copy the data, but they may also support DMA to or from user memory, so this I/O method is not necessarily less efficient than other methods merely exchanging buffer pointers. It is considered inferior though because no meta-information like frame counters or timestamps are passed. This information is necessary to recognize frame dropping and to synchronize with other data streams. However this is also the simplest I/O

method, requiring little or no setup to exchange data. It permits command line stunts like this (the `vidctrl` tool is fictitious):

```
$ vidctrl /dev/video --input=0 --format=YUYV --size=352x288
$ dd if=/dev/video of=myimage.422 bs=202752 count=1
```

To read from the device applications use the `read()` function, to write the `write()` function. Drivers must implement one I/O method if they exchange data with applications, but it need not be this.¹ When reading or writing is supported, the driver must also support the `select()` and `poll()` function.²

3.2.3.2 Streaming I/O (Memory Mapping)

Input and output devices support this I/O method when the `V4L2_CAP_STREAMING` flag in the `capabilities` field of `struct v4l2_capability` returned by the `ioctl VIDIOC_QUERYCAP` `ioctl` is set. There are two streaming methods, to determine if the memory mapping flavor is supported applications must call the `ioctl VIDIOC_REQBUFS` `ioctl` with the memory type set to `V4L2_MEMORY_MMAP`.

Streaming is an I/O method where only pointers to buffers are exchanged between application and driver, the data itself is not copied. Memory mapping is primarily intended to map buffers in device memory into the application's address space. Device memory can be for example the video memory on a graphics card with a video capture add-on. However, being the most efficient I/O method available for a long time, many other drivers support streaming as well, allocating buffers in DMA-able main memory.

A driver can support many sets of buffers. Each set is identified by a unique buffer type value. The sets are independent and each set can hold a different type of data. To access different sets at the same time different file descriptors must be used.¹

To allocate device buffers applications call the `ioctl VIDIOC_REQBUFS` `ioctl` with the desired number of buffers and buffer type, for example `V4L2_BUF_TYPE_VIDEO_CAPTURE`. This `ioctl` can also be used to change the number of buffers or to free the allocated memory, provided none of the buffers are still mapped.

Before applications can access the buffers they must map them into their address space with the `mmap()` function. The location of the buffers in device memory can be determined with the `ioctl VIDIOC_QUERYBUF` `ioctl`. In the single-planar API case, the `m.offset` and `length` returned in a `struct v4l2_buffer` are passed as sixth and second parameter to the `mmap()` function. When using the multi-planar API, `struct v4l2_buffer` contains an array of `struct v4l2_plane` structures, each containing its own `m.offset` and `length`. When using the multi-planar API, every plane of every buffer has to be mapped separately, so the number of calls to `mmap()` should be equal to number of buffers times number of planes in each buffer. The offset and length values must not be modified. Remember, the buffers are allocated in physical memory, as opposed to virtual memory, which can be swapped out to disk. Applications should free the buffers as soon as possible with the `munmap()` function.

¹ It would be desirable if applications could depend on drivers supporting all I/O interfaces, but as much as the complex memory mapping I/O can be inadequate for some devices we have no reason to require this interface, which is most useful for simple applications capturing still images.

² At the driver level `select()` and `poll()` are the same, and `select()` is too important to be optional.

¹ One could use one file descriptor and set the buffer type field accordingly when calling `ioctl VIDIOC_QBUF`, `VIDIOC_DQBUF` etc., but it makes the `select()` function ambiguous. We also like the clean approach of one file descriptor per logical stream. Video overlay for example is also a logical stream, although the CPU is not needed for continuous operation.

Example: Mapping buffers in the single-planar API

```

struct v4l2_requestbuffers reqbuf;
struct {
    void *start;
    size_t length;
} *buffers;
unsigned int i;

memset(&reqbuf, 0, sizeof(reqbuf));
reqbuf.type = V4L2_BUF_TYPE_VIDEO_CAPTURE;
reqbuf.memory = V4L2_MEMORY_MMAP;
reqbuf.count = 20;

if (-1 == ioctl (fd, VIDIOC_REQBUFS, &reqbuf)) {
    if (errno == EINVAL)
        printf("Video capturing or mmap-streaming is not supported\n");
    else
        perror("VIDIOC_REQBUFS");

    exit(EXIT_FAILURE);
}

/* We want at least five buffers. */

if (reqbuf.count < 5) {
    /* You may need to free the buffers here. */
    printf("Not enough buffer memory\n");
    exit(EXIT_FAILURE);
}

buffers = calloc(reqbuf.count, sizeof(*buffers));
assert(buffers != NULL);

for (i = 0; i < reqbuf.count; i++) {
    struct v4l2_buffer buffer;

    memset(&buffer, 0, sizeof(buffer));
    buffer.type = reqbuf.type;
    buffer.memory = V4L2_MEMORY_MMAP;
    buffer.index = i;

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

    buffers[i].length = buffer.length; /* remember for munmap() */

    buffers[i].start = mmap(NULL, buffer.length,
        PROT_READ | PROT_WRITE, /* recommended */
        MAP_SHARED, /* recommended */
        fd, buffer.m.offset);

    if (MAP_FAILED == buffers[i].start) {
        /* If you do not exit here you should unmap() and free()
           the buffers mapped so far. */
    }
}

```

```

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

/* Cleanup. */

for (i = 0; i < reqbuf.count; i++)
    munmap(buffer[i].start, buffer[i].length);

```

Example: Mapping buffers in the multi-planar API

```

struct v4l2_requestbuffers reqbuf;
/* Our current format uses 3 planes per buffer */
#define FMT_NUM_PLANES = 3

struct {
    void *start[FMT_NUM_PLANES];
    size_t length[FMT_NUM_PLANES];
} *buffers;
unsigned int i, j;

memset(&reqbuf, 0, sizeof(reqbuf));
reqbuf.type = V4L2_BUF_TYPE_VIDEO_CAPTURE_MPLANE;
reqbuf.memory = V4L2_MEMORY_MMAP;
reqbuf.count = 20;

if (ioctl(fd, VIDIOC_REQBUFS, &reqbuf) < 0) {
    if (errno == EINVAL)
        printf("Video capturing or mmap-streaming is not supported\n");
    else
        perror("VIDIOC_REQBUFS");

    exit(EXIT_FAILURE);
}

/* We want at least five buffers. */

if (reqbuf.count < 5) {
    /* You may need to free the buffers here. */
    printf("Not enough buffer memory\n");
    exit(EXIT_FAILURE);
}

buffers = calloc(reqbuf.count, sizeof(*buffers));
assert(buffers != NULL);

for (i = 0; i < reqbuf.count; i++) {
    struct v4l2_buffer buffer;
    struct v4l2_plane planes[FMT_NUM_PLANES];

    memset(&buffer, 0, sizeof(buffer));
    buffer.type = reqbuf.type;
    buffer.memory = V4L2_MEMORY_MMAP;
    buffer.index = i;

```

```

/* length in struct v4l2_buffer in multi-planar API stores the size
 * of planes array. */
buffer.length = FMT_NUM_PLANES;
buffer.m.planes = p_planes;

if (ioctl(fd, VIDIOC_QUERYBUF, &buffer) < 0) {
    perror("VIDIOC_QUERYBUF");
    exit(EXIT_FAILURE);
}

/* Every plane has to be mapped separately */
for (j = 0; j < FMT_NUM_PLANES; j++) {
    buffers[i].length[j] = buffer.m.planes[j].length; /* remember for munmap() */

    buffers[i].start[j] = mmap(NULL, buffer.m.planes[j].length,
                              PROT_READ | PROT_WRITE, /* recommended */
                              MAP_SHARED, /* recommended */
                              fd, buffer.m.planes[j].m.offset);

    if (MAP_FAILED == buffers[i].start[j]) {
        /* If you do not exit here you should unmap() and free()
         * the buffers and planes mapped so far. */
        perror("mmap");
        exit(EXIT_FAILURE);
    }
}
}

/* Cleanup. */
for (i = 0; i < reqbuf.count; i++)
    for (j = 0; j < FMT_NUM_PLANES; j++)
        munmap(buffers[i].start[j], buffers[i].length[j]);

```

Conceptually streaming drivers maintain two buffer queues, an incoming and an outgoing queue. They separate the synchronous capture or output operation locked to a video clock from the application which is subject to random disk or network delays and preemption by other processes, thereby reducing the probability of data loss. The queues are organized as FIFOs, buffers will be output in the order enqueued in the incoming FIFO, and were captured in the order dequeued from the outgoing FIFO.

The driver may require a minimum number of buffers enqueued at all times to function, apart of this no limit exists on the number of buffers applications can enqueue in advance, or dequeue and process. They can also enqueue in a different order than buffers have been dequeued, and the driver can *fill* enqueued *empty* buffers in any order.² The index number of a buffer (struct `v4l2_buffer` index) plays no role here, it only identifies the buffer.

Initially all mapped buffers are in dequeued state, inaccessible by the driver. For capturing applications it is customary to first enqueue all mapped buffers, then to start capturing and enter the read loop. Here the application waits until a filled buffer can be dequeued, and re-enqueues the buffer when the data is no longer needed. Output applications fill and enqueue buffers, when enough buffers are stacked up the output is started with `VIDIOC_STREAMON`. In

² Random enqueue order permits applications processing images out of order (such as video codecs) to return buffers earlier, reducing the probability of data loss. Random fill order allows drivers to reuse buffers on a LIFO-basis, taking advantage of caches holding scatter-gather lists and the like.

the write loop, when the application runs out of free buffers, it must wait until an empty buffer can be dequeued and reused.

To enqueue and dequeue a buffer applications use the `VIVIOC_QBUF` and `VIDIOC_DQBUF` ioctl. The status of a buffer being mapped, enqueued, full or empty can be determined at any time using the `ioctl VIDIOC_QUERYBUF` ioctl. Two methods exist to suspend execution of the application until one or more buffers can be dequeued. By default `VIDIOC_DQBUF` blocks when no buffer is in the outgoing queue. When the `O_NONBLOCK` flag was given to the `open()` function, `VIDIOC_DQBUF` returns immediately with an `EAGAIN` error code when no buffer is available. The `select()` or `poll()` functions are always available.

To start and stop capturing or output applications call the `VIDIOC_STREAMON` and `VIDIOC_STREAMOFF` ioctl.

Drivers implementing memory mapping I/O must support the `VIDIOC_REQBUFS`, `VIDIOC_QUERYBUF`, `VIDIOC_QBUF`, `VIDIOC_DQBUF`, `VIDIOC_STREAMON` and `VIDIOC_STREAMOFF` ioctls, the `mmap()`, `munmap()`, `select()` and `poll()` function.³

[capture example]

3.2.3.3 Streaming I/O (User Pointers)

Input and output devices support this I/O method when the `V4L2_CAP_STREAMING` flag in the `capabilities` field of struct `v4l2_capability` returned by the `ioctl VIDIOC_QUERYCAP` ioctl is set. If the particular user pointer method (not only memory mapping) is supported must be determined by calling the `ioctl VIDIOC_REQBUFS` ioctl with the memory type set to `V4L2_MEMORY_USERPTR`.

This I/O method combines advantages of the read/write and memory mapping methods. Buffers (planes) are allocated by the application itself, and can reside for example in virtual or shared memory. Only pointers to data are exchanged, these pointers and meta-information are passed in struct `v4l2_buffer` (or in struct `v4l2_plane` in the multi-planar API case). The driver must be switched into user pointer I/O mode by calling the `ioctl VIDIOC_REQBUFS` with the desired buffer type. No buffers (planes) are allocated beforehand, consequently they are not indexed and cannot be queried like mapped buffers with the `VIDIOC_QUERYBUF` ioctl.

Example: Initiating streaming I/O with user pointers

```
struct v4l2_requestbuffers reqbuf;

memset (&reqbuf, 0, sizeof (reqbuf));
reqbuf.type = V4L2_BUF_TYPE_VIDEO_CAPTURE;
reqbuf.memory = V4L2_MEMORY_USERPTR;

if (ioctl (fd, VIDIOC_REQBUFS, &reqbuf) == -1) {
    if (errno == EINVAL)
        printf ("Video capturing or user pointer streaming is not supported\n");
    else
        perror ("VIDIOC_REQBUFS");
}
```

³ At the driver level `select()` and `poll()` are the same, and `select()` is too important to be optional. The rest should be evident.

```
    exit (EXIT_FAILURE);  
}
```

Buffer (plane) addresses and sizes are passed on the fly with the `VIDIOC_QBUF` ioctl. Although buffers are commonly cycled, applications can pass different addresses and sizes at each `VIDIOC_QBUF` call. If required by the hardware the driver swaps memory pages within physical memory to create a continuous area of memory. This happens transparently to the application in the virtual memory subsystem of the kernel. When buffer pages have been swapped out to disk they are brought back and finally locked in physical memory for DMA.¹

Filled or displayed buffers are dequeued with the `VIDIOC_DQBUF` ioctl. The driver can unlock the memory pages at any time between the completion of the DMA and this ioctl. The memory is also unlocked when `VIDIOC_STREAMOFF` is called, `ioctl VIDIOC_REQBUFS`, or when the device is closed. Applications must take care not to free buffers without dequeuing. Firstly, the buffers remain locked for longer, wasting physical memory. Secondly the driver will not be notified when the memory is returned to the application's free list and subsequently reused for other purposes, possibly completing the requested DMA and overwriting valuable data.

For capturing applications it is customary to enqueue a number of empty buffers, to start capturing and enter the read loop. Here the application waits until a filled buffer can be dequeued, and re-enqueues the buffer when the data is no longer needed. Output applications fill and enqueue buffers, when enough buffers are stacked up output is started. In the write loop, when the application runs out of free buffers it must wait until an empty buffer can be dequeued and reused. Two methods exist to suspend execution of the application until one or more buffers can be dequeued. By default `VIDIOC_DQBUF` blocks when no buffer is in the outgoing queue. When the `O_NONBLOCK` flag was given to the `open()` function, `VIDIOC_DQBUF` returns immediately with an `EAGAIN` error code when no buffer is available. The `select()` or `poll()` function are always available.

To start and stop capturing or output applications call the `VIDIOC_STREAMON` and `VIDIOC_STREAMOFF` ioctl.

Note: `VIDIOC_STREAMOFF` removes all buffers from both queues and unlocks all buffers as a side effect. Since there is no notion of doing anything "now" on a multitasking system, if an application needs to synchronize with another event it should examine the `struct v4l2_buffer` timestamp of captured or outputted buffers.

Drivers implementing user pointer I/O must support the `VIDIOC_REQBUFS`, `VIDIOC_QBUF`, `VIDIOC_DQBUF`, `VIDIOC_STREAMON` and `VIDIOC_STREAMOFF` ioctls, the `select()` and `poll()` function.²

¹ We expect that frequently used buffers are typically not swapped out. Anyway, the process of swapping, locking or generating scatter-gather lists may be time consuming. The delay can be masked by the depth of the incoming buffer queue, and perhaps by maintaining caches assuming a buffer will be soon enqueued again. On the other hand, to optimize memory usage drivers can limit the number of buffers locked in advance and recycle the most recently used buffers first. Of course, the pages of empty buffers in the incoming queue need not be saved to disk. Output buffers must be saved on the incoming and outgoing queue because an application may share them with other processes.

² At the driver level `select()` and `poll()` are the same, and `select()` is too important to be optional. The rest should be evident.

3.2.3.4 Streaming I/O (DMA buffer importing)

The DMABUF framework provides a generic method for sharing buffers between multiple devices. Device drivers that support DMABUF can export a DMA buffer to userspace as a file descriptor (known as the exporter role), import a DMA buffer from userspace using a file descriptor previously exported for a different or the same device (known as the importer role), or both. This section describes the DMABUF importer role API in V4L2.

Refer to *DMABUF exporting* for details about exporting V4L2 buffers as DMABUF file descriptors.

Input and output devices support the streaming I/O method when the `V4L2_CAP_STREAMING` flag in the `capabilities` field of struct `v4l2_capability` returned by the `VIDIOC_QUERYCAP` ioctl is set. Whether importing DMA buffers through DMABUF file descriptors is supported is determined by calling the `VIDIOC_REQBUFS` ioctl with the memory type set to `V4L2_MEMORY_DMABUF`.

This I/O method is dedicated to sharing DMA buffers between different devices, which may be V4L devices or other video-related devices (e.g. DRM). Buffers (planes) are allocated by a driver on behalf of an application. Next, these buffers are exported to the application as file descriptors using an API which is specific for an allocator driver. Only such file descriptor are exchanged. The descriptors and meta-information are passed in struct `v4l2_buffer` (or in struct `v4l2_plane` in the multi-planar API case). The driver must be switched into DMABUF I/O mode by calling the `VIDIOC_REQBUFS` with the desired buffer type.

Example: Initiating streaming I/O with DMABUF file descriptors

```
struct v4l2_requestbuffers reqbuf;

memset(&reqbuf, 0, sizeof (reqbuf));
reqbuf.type = V4L2_BUF_TYPE_VIDEO_CAPTURE;
reqbuf.memory = V4L2_MEMORY_DMABUF;
reqbuf.count = 1;

if (ioctl(fd, VIDIOC_REQBUFS, &reqbuf) == -1) {
    if (errno == EINVAL)
        printf("Video capturing or DMABUF streaming is not supported\n");
    else
        perror("VIDIOC_REQBUFS");

    exit(EXIT_FAILURE);
}
```

The buffer (plane) file descriptor is passed on the fly with the `VIDIOC_QBUF` ioctl. In case of multiplanar buffers, every plane can be associated with a different DMABUF descriptor. Although buffers are commonly cycled, applications can pass a different DMABUF descriptor at each `VIDIOC_QBUF` call.

Example: Queueing DMABUF using single plane API

```

int buffer_queue(int v4lfd, int index, int dmafd)
{
    struct v4l2_buffer buf;

    memset(&buf, 0, sizeof buf);
    buf.type = V4L2_BUF_TYPE_VIDEO_CAPTURE;
    buf.memory = V4L2_MEMORY_DMABUF;
    buf.index = index;
    buf.m.fd = dmafd;

    if (ioctl(v4lfd, VIDIOC_QBUF, &buf) == -1) {
        perror("VIDIOC_QBUF");
        return -1;
    }

    return 0;
}

```

Example 3.6. Queueing DMABUF using multi plane API

```

int buffer_queue_mp(int v4lfd, int index, int dmafd[], int n_planes)
{
    struct v4l2_buffer buf;
    struct v4l2_plane planes[VIDEO_MAX_PLANES];
    int i;

    memset(&buf, 0, sizeof buf);
    buf.type = V4L2_BUF_TYPE_VIDEO_CAPTURE_MPLANE;
    buf.memory = V4L2_MEMORY_DMABUF;
    buf.index = index;
    buf.m.planes = planes;
    buf.length = n_planes;

    memset(&planes, 0, sizeof planes);

    for (i = 0; i < n_planes; ++i)
        buf.m.planes[i].m.fd = dmafd[i];

    if (ioctl(v4lfd, VIDIOC_QBUF, &buf) == -1) {
        perror("VIDIOC_QBUF");
        return -1;
    }

    return 0;
}

```

Captured or displayed buffers are dequeued with the `VIDIOC_DQBUF` ioctl. The driver can unlock the buffer at any time between the completion of the DMA and this ioctl. The memory is also unlocked when `VIDIOC_STREAMOFF` is called, `VIDIOC_REQBUFS`, or when the device is closed.

For capturing applications it is customary to enqueue a number of empty buffers, to start capturing and enter the read loop. Here the application waits until a filled buffer can be dequeued,

and re-enqueues the buffer when the data is no longer needed. Output applications fill and enqueue buffers, when enough buffers are stacked up output is started. In the write loop, when the application runs out of free buffers it must wait until an empty buffer can be dequeued and reused. Two methods exist to suspend execution of the application until one or more buffers can be dequeued. By default `VIDIOC_DQBUF` blocks when no buffer is in the outgoing queue. When the `O_NONBLOCK` flag was given to the `open()` function, `VIDIOC_DQBUF` returns immediately with an `EAGAIN` error code when no buffer is available. The `select()` and `poll()` functions are always available.

To start and stop capturing or displaying applications call the `VIDIOC_STREAMON` and `VIDIOC_STREAMOFF` ioctls.

Note: `VIDIOC_STREAMOFF` removes all buffers from both queues and unlocks all buffers as a side effect. Since there is no notion of doing anything “now” on a multitasking system, if an application needs to synchronize with another event it should examine the struct `v4l2_buffer` timestamp of captured or outputted buffers.

Drivers implementing DMABUF importing I/O must support the `VIDIOC_REQBUFS`, `VIDIOC_QBUF`, `VIDIOC_DQBUF`, `VIDIOC_STREAMON` and `VIDIOC_STREAMOFF` ioctls, and the `select()` and `poll()` functions.

3.2.3.5 Asynchronous I/O

This method is not defined yet.

3.2.3.6 Buffers

A buffer contains data exchanged by application and driver using one of the Streaming I/O methods. In the multi-planar API, the data is held in planes, while the buffer structure acts as a container for the planes. Only pointers to buffers (planes) are exchanged, the data itself is not copied. These pointers, together with meta-information like timestamps or field parity, are stored in a struct `v4l2_buffer`, argument to the *ioctl* `VIDIOC_QUERYBUF`, `VIDIOC_QBUF` and `VIDIOC_DQBUF` ioctl. In the multi-planar API, some plane-specific members of struct `v4l2_buffer`, such as pointers and sizes for each plane, are stored in struct `v4l2_plane` instead. In that case, struct `v4l2_buffer` contains an array of plane structures.

Dequeued video buffers come with timestamps. The driver decides at which part of the frame and with which clock the timestamp is taken. Please see flags in the masks `V4L2_BUF_FLAG_TIMESTAMP_MASK` and `V4L2_BUF_FLAG_TSTAMP_SRC_MASK` in *Buffer Flags*. These flags are always valid and constant across all buffers during the whole video stream. Changes in these flags may take place as a side effect of `VIDIOC_S_INPUT` or `VIDIOC_S_OUTPUT` however. The `V4L2_BUF_FLAG_TIMESTAMP_COPY` timestamp type which is used by e.g. on mem-to-mem devices is an exception to the rule: the timestamp source flags are copied from the OUTPUT video buffer to the CAPTURE video buffer.

Interactions between formats, controls and buffers

V4L2 exposes parameters that influence the buffer size, or the way data is laid out in the buffer. Those parameters are exposed through both formats and controls. One example of such a control is the `V4L2_CID_ROTATE` control that modifies the direction in which pixels are stored in the buffer, as well as the buffer size when the selected format includes padding at the end of lines.

The set of information needed to interpret the content of a buffer (e.g. the pixel format, the line stride, the tiling orientation or the rotation) is collectively referred to in the rest of this section as the buffer layout.

Controls that can modify the buffer layout shall set the `V4L2_CTRL_FLAG_MODIFY_LAYOUT` flag.

Modifying formats or controls that influence the buffer size or layout require the stream to be stopped. Any attempt at such a modification while the stream is active shall cause the `ioctl` setting the format or the control to return the `EBUSY` error code. In that case drivers shall also set the `V4L2_CTRL_FLAG_GRABBED` flag when calling `VIDIOC_QUERYCTRL()` or `VIDIOC_QUERY_EXT_CTRL()` for such a control while the stream is active.

Note: The `VIDIOC_S_SELECTION()` `ioctl` can, depending on the hardware (for instance if the device doesn't include a scaler), modify the format in addition to the selection rectangle. Similarly, the `VIDIOC_S_INPUT()`, `VIDIOC_S_OUTPUT()`, `VIDIOC_S_STD()` and `VIDIOC_S_DV_TIMINGS()` `ioctls` can also modify the format and selection rectangles. When those `ioctls` result in a buffer size or layout change, drivers shall handle that condition as they would handle it in the `VIDIOC_S_FMT()` `ioctl` in all cases described in this section.

Controls that only influence the buffer layout can be modified at any time when the stream is stopped. As they don't influence the buffer size, no special handling is needed to synchronize those controls with buffer allocation and the `V4L2_CTRL_FLAG_GRABBED` flag is cleared once the stream is stopped.

Formats and controls that influence the buffer size interact with buffer allocation. The simplest way to handle this is for drivers to always require buffers to be reallocated in order to change those formats or controls. In that case, to perform such changes, userspace applications shall first stop the video stream with the `VIDIOC_STREAMOFF()` `ioctl` if it is running and free all buffers with the `VIDIOC_REQBUFS()` `ioctl` if they are allocated. After freeing all buffers the `V4L2_CTRL_FLAG_GRABBED` flag for controls is cleared. The format or controls can then be modified, and buffers shall then be reallocated and the stream restarted. A typical `ioctl` sequence is

1. `VIDIOC_STREAMOFF`
2. `VIDIOC_REQBUFS(0)`
3. `VIDIOC_S_EXT_CTRL`
4. `VIDIOC_S_FMT`
5. `VIDIOC_REQBUFS(n)`
6. `VIDIOC_QBUF`
7. `VIDIOC_STREAMON`

The second `VIDIOC_REQBUFS()` call will take the new format and control value into account to compute the buffer size to allocate. Applications can also retrieve the size by calling the `VIDIOC_G_FMT()` `ioctl` if needed.

Note: The API doesn't mandate the above order for control (3.) and format (4.) changes. Format and controls can be set in a different order, or even interleaved, depending on the device and use case. For instance some controls might behave differently for different pixel formats, in which case the format might need to be set first.

When reallocation is required, any attempt to modify format or controls that influences the buffer size while buffers are allocated shall cause the format or control set `ioctl` to return the `EBUSY` error. Any attempt to queue a buffer too small for the current format or controls shall cause the `VIDIOC_QBUF()` `ioctl` to return a `EINVAL` error.

Buffer reallocation is an expensive operation. To avoid that cost, drivers can (and are encouraged to) allow format or controls that influence the buffer size to be changed with buffers allocated. In that case, a typical `ioctl` sequence to modify format and controls is

1. `VIDIOC_STREAMOFF`
2. `VIDIOC_S_EXT_CTRLs`
3. `VIDIOC_S_FMT`
4. `VIDIOC_QBUF`
5. `VIDIOC_STREAMON`

For this sequence to operate correctly, queued buffers need to be large enough for the new format or controls. Drivers shall return a `ENOSPC` error in response to format change (`VIDIOC_S_FMT()`) or control changes (`VIDIOC_S_CTRL()` or `VIDIOC_S_EXT_CTRLs()`) if buffers too small for the new format are currently queued. As a simplification, drivers are allowed to return a `EBUSY` error from these `ioctls` if any buffer is currently queued, without checking the queued buffers sizes.

Additionally, drivers shall return a `EINVAL` error from the `VIDIOC_QBUF()` `ioctl` if the buffer being queued is too small for the current format or controls. Together, these requirements ensure that queued buffers will always be large enough for the configured format and controls.

Userspace applications can query the buffer size required for a given format and controls by first setting the desired control values and then trying the desired format. The `VIDIOC_TRY_FMT()` `ioctl` will return the required buffer size.

1. `VIDIOC_S_EXT_CTRLs(x)`
2. `VIDIOC_TRY_FMT()`
3. `VIDIOC_S_EXT_CTRLs(y)`
4. `VIDIOC_TRY_FMT()`

The `VIDIOC_CREATE_BUFS()` `ioctl` can then be used to allocate buffers based on the queried sizes (for instance by allocating a set of buffers large enough for all the desired formats and controls, or by allocating separate set of appropriately sized buffers for each use case).

v4l2_buffer

struct v4l2_buffer

Table 101: struct v4l2_buffer

__u32	index	Number of the buffer, set by the application except when calling <i>VIDIOC_DQBUF</i> , then it is set by the driver. This field can range from zero to the number of buffers allocated with the <i>ioctl VIDIOC_REQBUFS</i> ioctl (struct v4l2_requestbuffers count), plus any buffers allocated with <i>ioctl VIDIOC_CREATE_BUFS</i> minus one.
__u32	type	Type of the buffer, same as struct v4l2_format type or struct v4l2_requestbuffers type, set by the application. See v4l2_buf_type
__u32	bytesused	The number of bytes occupied by the data in the buffer. It depends on the negotiated data format and may change with each buffer for compressed variable size data like JPEG images. Drivers must set this field when type refers to a capture stream, applications when it refers to an output stream. If the application sets this to 0 for an output stream, then bytesused will be set to the size of the buffer (see the length field of this struct) by the driver. For multiplanar formats this field is ignored and the planes pointer is used instead.
__u32	flags	Flags set by the application or driver, see <i>Buffer Flags</i> .
__u32	field	Indicates the field order of the image in the buffer, see <i>v4l2_field</i> . This field is not used when the buffer contains VBI data. Drivers must set it when type refers to a capture stream, applications when it refers to an output stream.
struct timeval	timestamp	For capture streams this is time when the first data byte was captured, as returned by the clock_gettime() function for the relevant clock id; see V4L2_BUF_FLAG_TIMESTAMP_* in <i>Buffer Flags</i> . For output streams the driver stores the time at which the last data byte was actually sent out in the timestamp field. This permits applications to monitor the drift between the video and system clock. For output streams that use V4L2_BUF_FLAG_TIMESTAMP_COPY the application has to fill in the timestamp which will be copied by the driver to the capture stream.
struct v4l2_timecode	timecode	When the V4L2_BUF_FLAG_TIMECODE flag is set in flags, this structure contains a frame timecode. In <i>V4L2_FIELD_ALTERNATE</i> mode the top and bottom field contain the same timecode. Timecodes are intended to help video editing and are typically recorded on video tapes, but also embedded in compressed formats like MPEG. This field is independent of the timestamp and sequence fields.
__u32	sequence	Set by the driver, counting the frames (not fields!) in sequence. This field is set for both input and output devices.

Continued on next page

Table 101 - continued from previous page

In *V4L2_FIELD_ALTERNATE* mode the top and bottom field have the same sequence number. The count starts at zero and includes dropped or repeated frames. A dropped frame was received by an input device but could not be stored due to lack of free buffer space. A repeated frame was displayed again by an output device because the application did not pass new data in time.

Note: This may count the frames received e.g. over USB, without taking into account the frames dropped by the remote hardware due to limited compression throughput or bus bandwidth. These devices identify by not enumerating any video standards, see *Video Standards*.

__u32	memory	This field must be set by applications and/or drivers in accordance with the selected I/O method. See <i>v4l2_memory</i>
union {	m	
__u32	offset	For the single-planar API and when memory is <i>V4L2_MEMORY_MMAP</i> this is the offset of the buffer from the start of the device memory. The value is returned by the driver and apart of serving as parameter to the <i>mmap()</i> function not useful for applications. See <i>Streaming I/O (Memory Mapping)</i> for details
unsigned long	userptr	For the single-planar API and when memory is <i>V4L2_MEMORY_USERPTR</i> this is a pointer to the buffer (casted to unsigned long type) in virtual memory, set by the application. See <i>Streaming I/O (User Pointers)</i> for details.
struct v4l2_plane	*planes	When using the multi-planar API, contains a userspace pointer to an array of struct <i>v4l2_plane</i> . The size of the array should be put in the <i>length</i> field of this struct <i>v4l2_buffer</i> structure.
int	fd	For the single-plane API and when memory is <i>V4L2_MEMORY_DMABUF</i> this is the file descriptor associated with a <i>DMABUF</i> buffer.
}		
__u32	length	Size of the buffer (not the payload) in bytes for the single-planar API. This is set by the driver based on the calls to <i>ioctl VIDIOC_REQBUFS</i> and/or <i>ioctl VIDIOC_CREATE_BUFS</i> . For the multi-planar API the application sets this to the number of elements in the <i>planes</i> array. The driver will fill in the actual number of valid elements in that array.
__u32	reserved2	A place holder for future extensions. Drivers and applications must set this to 0.

Continued on next page

Table 101 - continued from previous page

<code>__u32</code>	<code>request_fd</code>	<p>The file descriptor of the request to queue the buffer to. If the flag <code>V4L2_BUF_FLAG_REQUEST_FD</code> is set, then the buffer will be queued to this request. If the flag is not set, then this field will be ignored.</p> <p>The <code>V4L2_BUF_FLAG_REQUEST_FD</code> flag and this field are only used by <i>ioctl VIDIOC_QBUF</i> and ignored by other ioctls that take a <code>v4l2_buffer</code> as argument.</p> <p>Applications should not set <code>V4L2_BUF_FLAG_REQUEST_FD</code> for any ioctls other than <i>VIDIOC_QBUF</i>.</p> <p>If the device does not support requests, then <code>EBADR</code> will be returned. If requests are supported but an invalid request file descriptor is given, then <code>EINVAL</code> will be returned.</p>
--------------------	-------------------------	--

v4l2_plane

struct v4l2_plane

<code>__u32</code>	<code>bytesused</code>	<p>The number of bytes occupied by data in the plane (its payload). Drivers must set this field when <code>type</code> refers to a capture stream, applications when it refers to an output stream. If the application sets this to 0 for an output stream, then <code>bytesused</code> will be set to the size of the plane (see the <code>length</code> field of this struct) by the driver.</p> <hr/> <p>Note: Note that the actual image data starts at <code>data_offset</code> which may not be 0.</p> <hr/>
<code>__u32</code>	<code>length</code>	Size in bytes of the plane (not its payload). This is set by the driver based on the calls to <i>ioctl VIDIOC_REQBUFS</i> and/or <i>ioctl VIDIOC_CREATE_BUFS</i> .
<code>union {</code>	<code>m</code>	
<code>__u32</code>	<code>mem_offset</code>	When the memory type in the containing struct <code>v4l2_buffer</code> is <code>V4L2_MEMORY_MMAP</code> , this is the value that should be passed to <code>mmap()</code> , similar to the <code>offset</code> field in struct <code>v4l2_buffer</code> .
<code>unsigned long</code>	<code>userptr</code>	When the memory type in the containing struct <code>v4l2_buffer</code> is <code>V4L2_MEMORY_USERPTR</code> , this is a userspace pointer to the memory allocated for this plane by an application.
<code>int</code>	<code>fd</code>	When the memory type in the containing struct <code>v4l2_buffer</code> is <code>V4L2_MEMORY_DMABUF</code> , this is a file descriptor associated with a <code>DMABUF</code> buffer, similar to the <code>fd</code> field in struct <code>v4l2_buffer</code> .
<code>}</code>		

Continued on next page

Table 102 – continued from previous page

__u32	data_offset	Offset in bytes to video data in the plane. Drivers must set this field when type refers to a capture stream, applications when it refers to an output stream. Note: That data_offset is included in bytesused. So the size of the image in the plane is bytesused-data_offset at offset data_offset from the start of the plane.
__u32	reserved[11]	Reserved for future use. Should be zeroed by drivers and applications.

v4l2_buf_type**enum v4l2_buf_type**

V4L2_BUF_TYPE_VIDEO_CAPTURE	1	Buffer of a single-planar video capture stream, see <i>Video Capture Interface</i> .
V4L2_BUF_TYPE_VIDEO_CAPTURE_MPLANE	9	Buffer of a multi-planar video capture stream, see <i>Video Capture Interface</i> .
V4L2_BUF_TYPE_VIDEO_OUTPUT	2	Buffer of a single-planar video output stream, see <i>Video Output Interface</i> .
V4L2_BUF_TYPE_VIDEO_OUTPUT_MPLANE	10	Buffer of a multi-planar video output stream, see <i>Video Output Interface</i> .
V4L2_BUF_TYPE_VIDEO_OVERLAY	3	Buffer for video overlay, see <i>Video Overlay Interface</i> .
V4L2_BUF_TYPE_VBI_CAPTURE	4	Buffer of a raw VBI capture stream, see <i>Raw VBI Data Interface</i> .
V4L2_BUF_TYPE_VBI_OUTPUT	5	Buffer of a raw VBI output stream, see <i>Raw VBI Data Interface</i> .
V4L2_BUF_TYPE_SLICED_VBI_CAPTURE	6	Buffer of a sliced VBI capture stream, see <i>Sliced VBI Data Interface</i> .
V4L2_BUF_TYPE_SLICED_VBI_OUTPUT	7	Buffer of a sliced VBI output stream, see <i>Sliced VBI Data Interface</i> .
V4L2_BUF_TYPE_VIDEO_OUTPUT_OVERLAY	8	Buffer for video output overlay (OSD), see <i>Video Output Overlay Interface</i> .
V4L2_BUF_TYPE_SDR_CAPTURE	11	Buffer for Software Defined Radio (SDR) capture stream, see <i>Software Defined Radio Interface (SDR)</i> .
V4L2_BUF_TYPE_SDR_OUTPUT	12	Buffer for Software Defined Radio (SDR) output stream, see <i>Software Defined Radio Interface (SDR)</i> .
V4L2_BUF_TYPE_META_CAPTURE	13	Buffer for metadata capture, see <i>Metadata Interface</i> .
V4L2_BUF_TYPE_META_OUTPUT	14	Buffer for metadata output, see <i>Metadata Interface</i> .

Buffer Flags

V4L2_BUF_FLAG_MAPPED	0x00000001	The buffer resides in device memory and has been mapped into the application's address space, see <i>Streaming I/O (Memory Mapping)</i> for details. Drivers set or clear this flag when the <i>ioctl VIDIOC_QUERYBUF</i> , <i>ioctl VIDIOC_QBUF</i> , <i>VIDIOC_DQBUF</i> or <i>VIDIOC_DQBUF</i> <i>ioctl</i> is called. Set by the driver.
V4L2_BUF_FLAG_QUEUED	0x00000002	Internally drivers maintain two buffer queues, an incoming and outgoing queue. When this flag is set, the buffer is currently on the incoming queue. It automatically moves to the outgoing queue after the buffer has been filled (capture devices) or displayed (output devices). Drivers set or clear this flag when the <i>VIDIOC_QUERYBUF</i> <i>ioctl</i> is called. After (successful) calling the <i>VIDIOC_QBUF</i> <i>ioctl</i> it is always set and after <i>VIDIOC_DQBUF</i> always cleared.
V4L2_BUF_FLAG_DONE	0x00000004	When this flag is set, the buffer is currently on the outgoing queue, ready to be dequeued from the driver. Drivers set or clear this flag when the <i>VIDIOC_QUERYBUF</i> <i>ioctl</i> is called. After calling the <i>VIDIOC_QBUF</i> or <i>VIDIOC_DQBUF</i> it is always cleared. Of course a buffer cannot be on both queues at the same time, the <i>V4L2_BUF_FLAG_QUEUED</i> and <i>V4L2_BUF_FLAG_DONE</i> flag are mutually exclusive. They can be both cleared however, then the buffer is in "dequeued" state, in the application domain so to say.
V4L2_BUF_FLAG_ERROR	0x00000040	When this flag is set, the buffer has been dequeued successfully, although the data might have been corrupted. This is recoverable, streaming may continue as normal and the buffer may be reused normally. Drivers set this flag when the <i>VIDIOC_DQBUF</i> <i>ioctl</i> is called.
V4L2_BUF_FLAG_IN_REQUEST	0x00000080	This buffer is part of a request that hasn't been queued yet.
V4L2_BUF_FLAG_KEYFRAME	0x00000008	Drivers set or clear this flag when calling the <i>VIDIOC_DQBUF</i> <i>ioctl</i> . It may be set by video capture devices when the buffer contains a compressed image which is a key frame (or field), i. e. can be decompressed on its own. Also known as an I-frame. Applications can set this bit when type refers to an output stream.
V4L2_BUF_FLAG_PFRAME	0x00000010	Similar to <i>V4L2_BUF_FLAG_KEYFRAME</i> this flags predicted frames or fields which contain only differences to a previous key frame. Applications can set this bit when type refers to an output stream.
V4L2_BUF_FLAG_BFRAME	0x00000020	Similar to <i>V4L2_BUF_FLAG_KEYFRAME</i> this flags a bi-directional predicted frame or field which contains only the differences between the current frame and both the preceding and following key frames to specify its content. Applications can set this bit when type refers to an output stream.
V4L2_BUF_FLAG_TIMECODE	0x00000100	The timecode field is valid. Drivers set or clear this flag when the <i>VIDIOC_DQBUF</i> <i>ioctl</i> is called. Applications can set this bit and the corresponding timecode structure when type refers to an output stream.
V4L2_BUF_FLAG_PREPARED	0x00000400	The buffer has been prepared for I/O and can be queued by the application. Drivers set or clear this flag when the <i>ioctl VIDIOC_QUERYBUF</i> , <i>VIDIOC_PREPARE_BUF</i> , <i>ioctl VIDIOC_QBUF</i> , <i>VIDIOC_DQBUF</i> or <i>VIDIOC_DQBUF</i> <i>ioctl</i> is called.

Continued on next page

Table 103 - continued from previous page

V4L2_BUF_FLAG_NO_CACHE_INVALIDATE	0x00000800	Caches do not have to be invalidated for this buffer. Typically applications shall use this flag if the data captured in the buffer is not going to be touched by the CPU, instead the buffer will, probably, be passed on to a DMA-capable hardware unit for further processing or output. This flag is ignored unless the queue is used for <i>memory mapping</i> streaming I/O and reports V4L2_BUF_CAP_SUPPORTS_MMAP_CACHE_HINTS capability.
V4L2_BUF_FLAG_NO_CACHE_CLEAN	0x00001000	Caches do not have to be cleaned for this buffer. Typically applications shall use this flag for output buffers if the data in this buffer has not been created by the CPU but by some DMA-capable unit, in which case caches have not been used. This flag is ignored unless the queue is used for <i>memory mapping</i> streaming I/O and reports V4L2_BUF_CAP_SUPPORTS_MMAP_CACHE_HINTS capability.
V4L2_BUF_FLAG_M2M_HOLD_CAPTURE_BUF	0x00000200	Only valid if struct <code>v4l2_requestbuffers</code> flag <code>V4L2_BUF_CAP_SUPPORTS_M2M_HOLD_CAPTURE_BUF</code> is set. It is typically used with stateless decoders where multiple output buffers each decode to a slice of the decoded frame. Applications can set this flag when queueing the output buffer to prevent the driver from dequeuing the capture buffer after the output buffer has been decoded (i.e. the capture buffer is 'held'). If the timestamp of this output buffer differs from that of the previous output buffer, then that indicates the start of a new frame and the previously held capture buffer is dequeued.
V4L2_BUF_FLAG_LAST	0x00100000	Last buffer produced by the hardware. <code>mem2mem</code> codec drivers set this flag on the capture queue for the last buffer when the <code>ioctl VIDIOC_QUERYBUF</code> or <code>VIDIOC_DQBUF</code> <code>ioctl</code> is called. Due to hardware limitations, the last buffer may be empty. In this case the driver will set the <code>bytesused</code> field to 0, regardless of the format. Any subsequent call to the <code>VIDIOC_DQBUF</code> <code>ioctl</code> will not block anymore, but return an <code>EPIPE</code> error code.
V4L2_BUF_FLAG_REQUEST_FD	0x00800000	The <code>request_fd</code> field contains a valid file descriptor.
V4L2_BUF_FLAG_TIMESTAMP_MASK	0x0000e000	Mask for timestamp types below. To test the timestamp type, mask out bits not belonging to timestamp type by performing a logical and operation with buffer flags and timestamp mask.
V4L2_BUF_FLAG_TIMESTAMP_UNKNOWN	0x00000000	Unknown timestamp type. This type is used by drivers before Linux 3.9 and may be either monotonic (see below) or realtime (wall clock). Monotonic clock has been favoured in embedded systems whereas most of the drivers use the realtime clock. Either kinds of timestamps are available in user space via <code>clock_gettime()</code> using clock IDs <code>CLOCK_MONOTONIC</code> and <code>CLOCK_REALTIME</code> , respectively.
V4L2_BUF_FLAG_TIMESTAMP_MONOTONIC	0x00002000	The buffer timestamp has been taken from the <code>CLOCK_MONOTONIC</code> clock. To access the same clock outside V4L2, use <code>clock_gettime()</code> .
V4L2_BUF_FLAG_TIMESTAMP_COPY	0x00004000	The CAPTURE buffer timestamp has been taken from the corresponding OUTPUT buffer. This flag applies only to <code>mem2mem</code> devices.

Continued on next page

Table 103 - continued from previous page

V4L2_BUF_FLAG_TSTAMP_SRC_MASK	0x00070000	Mask for timestamp sources below. The timestamp source defines the point of time the timestamp is taken in relation to the frame. Logical 'and' operation between the flags field and V4L2_BUF_FLAG_TSTAMP_SRC_MASK produces the value of the timestamp source. Applications must set the timestamp source when type refers to an output stream and V4L2_BUF_FLAG_TIMESTAMP_COPY is set.
V4L2_BUF_FLAG_TSTAMP_SRC_EOF	0x00000000	End Of Frame. The buffer timestamp has been taken when the last pixel of the frame has been received or the last pixel of the frame has been transmitted. In practice, software generated timestamps will typically be read from the clock a small amount of time after the last pixel has been received or transmitted, depending on the system and other activity in it.
V4L2_BUF_FLAG_TSTAMP_SRC_SOE	0x00010000	Start Of Exposure. The buffer timestamp has been taken when the exposure of the frame has begun. This is only valid for the V4L2_BUF_TYPE_VIDEO_CAPTURE buffer type.

enum v4l2_memory

V4L2_MEMORY_MMAP	1	The buffer is used for <i>memory mapping</i> I/O.
V4L2_MEMORY_USERPTR	2	The buffer is used for <i>user pointer</i> I/O.
V4L2_MEMORY_OVERLAY	3	[to do]
V4L2_MEMORY_DMABUF	4	The buffer is used for <i>DMA shared buffer</i> I/O.

Memory Consistency Flags

V4L2_MEMORY_FLAG_NON_COHERENT	0x00000001	A buffer is allocated either in coherent (it will be automatically coherent between the CPU and the bus) or non-coherent memory. The latter can provide performance gains, for instance the CPU cache sync/flush operations can be avoided if the buffer is accessed by the corresponding device only and the CPU does not read/write to/from that buffer. However, this requires extra care from the driver - it must guarantee memory consistency by issuing a cache flush/sync when consistency is needed. If this flag is set V4L2 will attempt to allocate the buffer in non-coherent memory. The flag takes effect only if the buffer is used for <i>memory mapping</i> I/O and the queue reports the <i>V4L2_BUF_CAP_SUPPORTS_MMAP_CACHE_HINTS</i> capability.
-------------------------------	------------	---

Timecodes

The `v4l2_buffer_timecode` structure is designed to hold a *SMPTE 12M* or similar timecode. (struct timeval timestamps are stored in the struct v4l2_buffer timestamp field.)

v4l2_timecode

struct v4l2_timecode

__u32	type	Frame rate the timecodes are based on, see <i>Timecode Types</i> .
__u32	flags	Timecode flags, see <i>Timecode Flags</i> .
__u8	frames	Frame count, 0 ... 23/24/29/49/59, depending on the type of timecode.
__u8	seconds	Seconds count, 0 ... 59. This is a binary, not BCD number.
__u8	minutes	Minutes count, 0 ... 59. This is a binary, not BCD number.
__u8	hours	Hours count, 0 ... 29. This is a binary, not BCD number.
__u8	userbits[4]	The “user group” bits from the timecode.

Timecode Types

V4L2_TC_TYPE_24FPS	1	24 frames per second, i. e. film.
V4L2_TC_TYPE_25FPS	2	25 frames per second, i. e. PAL or SECAM video.
V4L2_TC_TYPE_30FPS	3	30 frames per second, i. e. NTSC video.
V4L2_TC_TYPE_50FPS	4	
V4L2_TC_TYPE_60FPS	5	

Timecode Flags

V4L2_TC_FLAG_DROPFRAME	0x0001	Indicates “drop frame” semantics for counting frames in 29.97 fps material. When set, frame numbers 0 and 1 at the start of each minute, except minutes 0, 10, 20, 30, 40, 50 are omitted from the count.
V4L2_TC_FLAG_COLORFRAME	0x0002	The “color frame” flag.
V4L2_TC_USERBITS_field	0x000C	Field mask for the “binary group flags”.
V4L2_TC_USERBITS_USERDEFINED	0x0000	Unspecified format.
V4L2_TC_USERBITS_8BITCHARS	0x0008	8-bit ISO characters.

3.2.3.7 Field Order

We have to distinguish between progressive and interlaced video. Progressive video transmits all lines of a video image sequentially. Interlaced video divides an image into two fields, containing only the odd and even lines of the image, respectively. Alternating the so called odd and even field are transmitted, and due to a small delay between fields a cathode ray TV displays the lines interleaved, yielding the original frame. This curious technique was invented because at refresh rates similar to film the image would fade out too quickly. Transmitting fields reduces the flicker without the necessity of doubling the frame rate and with it the bandwidth required for each channel.

It is important to understand a video camera does not expose one frame at a time, merely transmitting the frames separated into fields. The fields are in fact captured at two different instances in time. An object on screen may well move between one field and the next. For applications analysing motion it is of paramount importance to recognize which field of a frame is older, the *temporal order*.

When the driver provides or accepts images field by field rather than interleaved, it is also important applications understand how the fields combine to frames. We distinguish between top (aka odd) and bottom (aka even) fields, the *spatial order*: The first line of the top field is the first line of an interlaced frame, the first line of the bottom field is the second line of that frame.

However because fields were captured one after the other, arguing whether a frame commences with the top or bottom field is pointless. Any two successive top and bottom, or bottom and top fields yield a valid frame. Only when the source was progressive to begin with, e. g. when transferring film to video, two fields may come from the same frame, creating a natural order.

Counter to intuition the top field is not necessarily the older field. Whether the older field contains the top or bottom lines is a convention determined by the video standard. Hence the distinction between temporal and spatial order of fields. The diagrams below should make this clearer.

In V4L it is assumed that all video cameras transmit fields on the media bus in the same order they were captured, so if the top field was captured first (is the older field), the top field is also transmitted first on the bus.

All video capture and output devices must report the current field order. Some drivers may permit the selection of a different order, to this end applications initialize the `field` field of struct `v4l2_pix_format` before calling the `VIDIOC_S_FMT` ioctl. If this is not desired it should have the value `V4L2_FIELD_ANY` (0).

enum v4l2_field

v4l2_field

V4L2_FIELD_ANY	0	Applications request this field order when any field format is acceptable. Drivers choose depending on hardware capabilities or e.g. the requested image size, and return the actual field order. Drivers must never return V4L2_FIELD_ANY. If multiple field orders are possible the driver must choose one of the possible field orders during <i>VIDIOC_S_FMT</i> or <i>VIDIOC_TRY_FMT</i> . struct <i>v4l2_buffer</i> field can never be V4L2_FIELD_ANY.
V4L2_FIELD_NONE	1	Images are in progressive (frame-based) format, not interlaced (field-based).
V4L2_FIELD_TOP	2	Images consist of the top (aka odd) field only.
V4L2_FIELD_BOTTOM	3	Images consist of the bottom (aka even) field only. Applications may wish to prevent a device from capturing interlaced images because they will have “comb” or “feathering” artefacts around moving objects.
V4L2_FIELD_INTERLACED	4	Images contain both fields, interleaved line by line. The temporal order of the fields (whether the top or bottom field is older) depends on the current video standard. In M/NTSC the bottom field is the older field. In all other standards the top field is the older field.
V4L2_FIELD_SEQ_TB	5	Images contain both fields, the top field lines are stored first in memory, immediately followed by the bottom field lines. Fields are always stored in temporal order, the older one first in memory. Image sizes refer to the frame, not fields.
V4L2_FIELD_SEQ_BT	6	Images contain both fields, the bottom field lines are stored first in memory, immediately followed by the top field lines. Fields are always stored in temporal order, the older one first in memory. Image sizes refer to the frame, not fields.
V4L2_FIELD_ALTERNATE	7	The two fields of a frame are passed in separate buffers, in temporal order, i. e. the older one first. To indicate the field parity (whether the current field is a top or bottom field) the driver or application, depending on data direction, must set struct <i>v4l2_buffer</i> field to V4L2_FIELD_TOP or V4L2_FIELD_BOTTOM. Any two successive fields pair to build a frame. If fields are successive, without any dropped fields between them (fields can drop individually), can be determined from the struct <i>v4l2_buffer</i> sequence field. This format cannot be selected when using the read/write I/O method since there is no way to communicate if a field was a top or bottom field.
V4L2_FIELD_INTERLACED_TB	8	Images contain both fields, interleaved line by line, top field first. The top field is the older field.
V4L2_FIELD_INTERLACED_BT	9	Images contain both fields, interleaved line by line, top field first. The bottom field is the older field.

Field Order, Top Field First Transmitted

Field Order, Bottom Field First Transmitted

3.2.4 Interfaces

3.2.4.1 Video Capture Interface

Video capture devices sample an analog video signal and store the digitized images in memory. Today nearly all devices can capture at full 25 or 30 frames/second. With this interface applications can control the capture process and move images from the driver into user space.

Conventionally V4L2 video capture devices are accessed through character device special files named `/dev/video` and `/dev/video0` to `/dev/video63` with major number 81 and minor numbers 0 to 63. `/dev/video` is typically a symbolic link to the preferred video device.

Note: The same device file names are used for video output devices.

Querying Capabilities

Devices supporting the video capture interface set the `V4L2_CAP_VIDEO_CAPTURE` or `V4L2_CAP_VIDEO_CAPTURE_MPLANE` flag in the capabilities field of struct `v4l2_capability` returned by the *ioctl* `VIDIOC_QUERYCAP` *ioctl*. As secondary device functions they may also support the *video overlay* (`V4L2_CAP_VIDEO_OVERLAY`) and the *raw VBI capture* (`V4L2_CAP_VBI_CAPTURE`) interface. At least one of the read/write or streaming I/O methods must be supported. Tuners and audio inputs are optional.

Supplemental Functions

Video capture devices shall support *audio input*, *Tuners and Modulators*, *controls*, *cropping and scaling* and *streaming parameter* *ioctls* as needed. The *video input* *ioctls* must be supported by all video capture devices.

Image Format Negotiation

The result of a capture operation is determined by cropping and image format parameters. The former select an area of the video picture to capture, the latter how images are stored in memory, i. e. in RGB or YUV format, the number of bits per pixel or width and height. Together they also define how images are scaled in the process.

As usual these parameters are *not* reset at `open()` time to permit Unix tool chains, programming a device and then reading from it as if it was a plain file. Well written V4L2 applications ensure they really get what they want, including cropping and scaling.

Cropping initialization at minimum requires to reset the parameters to defaults. An example is given in *Image Cropping, Insertion and Scaling - the CROP API*.

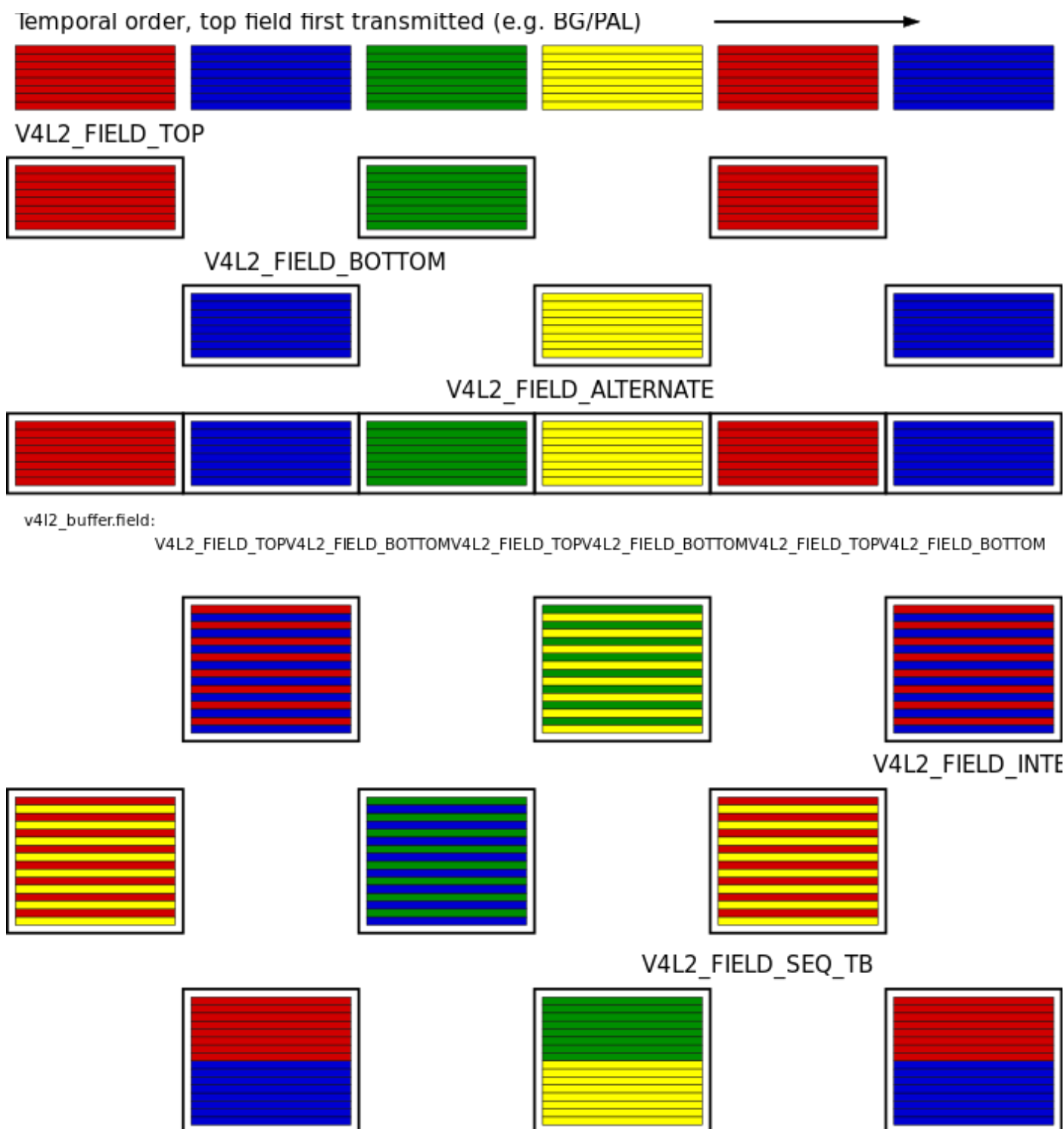


Fig. 6: Field Order, Top Field First Transmitted

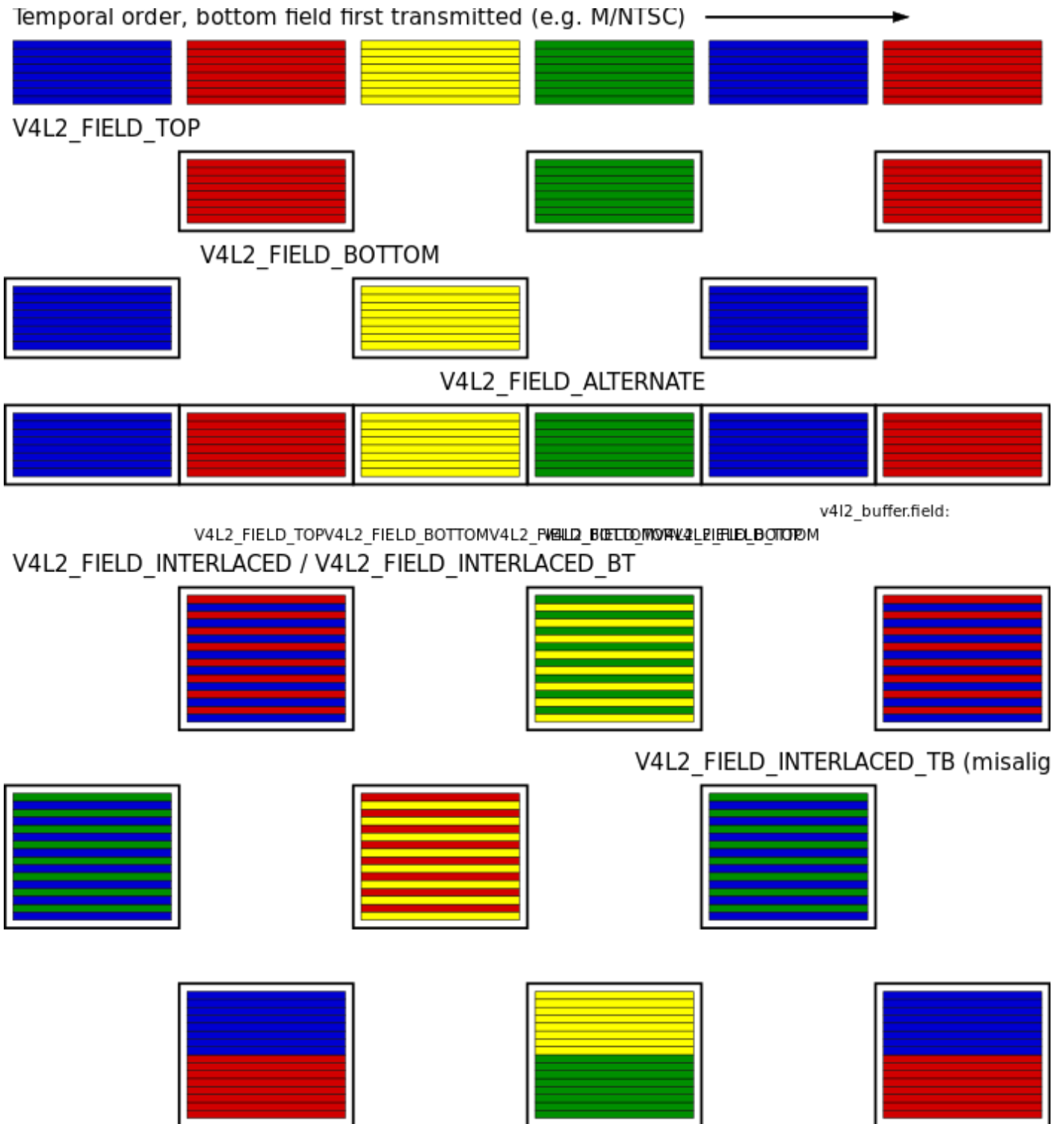


Fig. 7: Field Order, Bottom Field First Transmitted

To query the current image format applications set the `type` field of a struct `v4l2_format` to `V4L2_BUF_TYPE_VIDEO_CAPTURE` or `V4L2_BUF_TYPE_VIDEO_CAPTURE_MPLANE` and call the `VIDIOC_G_FMT` ioctl with a pointer to this structure. Drivers fill the struct `v4l2_pix_format` `pix` or the struct `v4l2_pix_format_mplane` `pix_mp` member of the `fmt` union.

To request different parameters applications set the `type` field of a struct `v4l2_format` as above and initialize all fields of the struct `v4l2_pix_format` `vbi` member of the `fmt` union, or better just modify the results of `VIDIOC_G_FMT`, and call the `VIDIOC_S_FMT` ioctl with a pointer to this structure. Drivers may adjust the parameters and finally return the actual parameters as `VIDIOC_G_FMT` does.

Like `VIDIOC_S_FMT` the `VIDIOC_TRY_FMT` ioctl can be used to learn about hardware limitations without disabling I/O or possibly time consuming hardware preparations.

The contents of struct `v4l2_pix_format` and struct `v4l2_pix_format_mplane` are discussed in *Image Formats*. See also the specification of the `VIDIOC_G_FMT`, `VIDIOC_S_FMT` and `VIDIOC_TRY_FMT` ioctls for details. Video capture devices must implement both the `VIDIOC_G_FMT` and `VIDIOC_S_FMT` ioctl, even if `VIDIOC_S_FMT` ignores all requests and always returns default parameters as `VIDIOC_G_FMT` does. `VIDIOC_TRY_FMT` is optional.

Reading Images

A video capture device may support the *read()* function and/or streaming (*memory mapping* or *user pointer*) I/O. See *Input/Output* for details.

3.2.4.2 Video Overlay Interface

Also known as Framebuffer Overlay or Previewing.

Video overlay devices have the ability to genlock (TV-)video into the (VGA-)video signal of a graphics card, or to store captured images directly in video memory of a graphics card, typically with clipping. This can be considerable more efficient than capturing images and displaying them by other means. In the old days when only nuclear power plants needed cooling towers this used to be the only way to put live video into a window.

Video overlay devices are accessed through the same character special files as *video capture* devices.

Note: The default function of a `/dev/video` device is video capturing. The overlay function is only available after calling the `VIDIOC_S_FMT` ioctl.

The driver may support simultaneous overlay and capturing using the *read/write* and streaming I/O methods. If so, operation at the nominal frame rate of the video standard is not guaranteed. Frames may be directed away from overlay to capture, or one field may be used for overlay and the other for capture if the capture parameters permit this.

Applications should use different file descriptors for capturing and overlay. This must be supported by all drivers capable of simultaneous capturing and overlay. Optionally these drivers may also permit capturing and overlay with a single file descriptor for compatibility with V4L and earlier versions of V4L2.¹

¹ In the opinion of the designers of this API, no driver writer taking the efforts to support simultaneous capturing

A common application of two file descriptors is the X11 *Xv/V4L* interface driver and a V4L2 application. While the X server controls video overlay, the application can take advantage of memory mapping and DMA.

Querying Capabilities

Devices supporting the video overlay interface set the `V4L2_CAP_VIDEO_OVERLAY` flag in the `capabilities` field of struct `v4l2_capability` returned by the `ioctl VIDIOC_QUERYCAP` ioctl. The overlay I/O method specified below must be supported. Tuners and audio inputs are optional.

Supplemental Functions

Video overlay devices shall support *audio input*, *Tuners and Modulators*, *controls*, *cropping and scaling* and *streaming parameter* ioctls as needed. The *video input* and *video standard* ioctls must be supported by all video overlay devices.

Setup

Before overlay can commence applications must program the driver with frame buffer parameters, namely the address and size of the frame buffer and the image format, for example RGB 5:6:5. The `VIDIOC_G_FBUF` and `VIDIOC_S_FBUF` ioctls are available to get and set these parameters, respectively. The `VIDIOC_S_FBUF` ioctl is privileged because it allows to set up DMA into physical memory, bypassing the memory protection mechanisms of the kernel. Only the superuser can change the frame buffer address and size. Users are not supposed to run TV applications as root or with SUID bit set. A small helper application with suitable privileges should query the graphics system and program the V4L2 driver at the appropriate time.

Some devices add the video overlay to the output signal of the graphics card. In this case the frame buffer is not modified by the video device, and the frame buffer address and pixel format are not needed by the driver. The `VIDIOC_S_FBUF` ioctl is not privileged. An application can check for this type of device by calling the `VIDIOC_G_FBUF` ioctl.

A driver may support any (or none) of five clipping/blending methods:

1. Chroma-keying displays the overlaid image only where pixels in the primary graphics surface assume a certain color.
2. A bitmap can be specified where each bit corresponds to a pixel in the overlaid image. When the bit is set, the corresponding video pixel is displayed, otherwise a pixel of the graphics surface.
3. A list of clipping rectangles can be specified. In these regions *no* video is displayed, so the graphics surface can be seen here.
4. The framebuffer has an alpha channel that can be used to clip or blend the framebuffer with the video.

and overlay will restrict this ability by requiring a single file descriptor, as in V4L and earlier versions of V4L2. Making this optional means applications depending on two file descriptors need backup routines to be compatible with all drivers, which is considerable more work than using two fds in applications which do not. Also two fd's fit the general concept of one file descriptor for each logical stream. Hence as a complexity trade-off drivers *must* support two file descriptors and *may* support single fd operation.

5. A global alpha value can be specified to blend the framebuffer contents with video images.

When simultaneous capturing and overlay is supported and the hardware prohibits different image and frame buffer formats, the format requested first takes precedence. The attempt to capture (*VIDIOC_S_FMT*) or overlay (*VIDIOC_S_FBUF*) may fail with an EBUSY error code or return accordingly modified parameters..

Overlay Window

The overlaid image is determined by cropping and overlay window parameters. The former select an area of the video picture to capture, the latter how images are overlaid and clipped. Cropping initialization at minimum requires to reset the parameters to defaults. An example is given in *Image Cropping, Insertion and Scaling - the CROP API*.

The overlay window is described by a struct `v4l2_window`. It defines the size of the image, its position over the graphics surface and the clipping to be applied. To get the current parameters applications set the `type` field of a struct `v4l2_format` to `V4L2_BUF_TYPE_VIDEO_OVERLAY` and call the *VIDIOC_G_FMT* ioctl. The driver fills the struct `v4l2_window` substructure named `win`. It is not possible to retrieve a previously programmed clipping list or bitmap.

To program the overlay window applications set the `type` field of a struct `v4l2_format` to `V4L2_BUF_TYPE_VIDEO_OVERLAY`, initialize the `win` substructure and call the *VIDIOC_S_FMT* ioctl. The driver adjusts the parameters against hardware limits and returns the actual parameters as *VIDIOC_G_FMT* does. Like *VIDIOC_S_FMT*, the *VIDIOC_TRY_FMT* ioctl can be used to learn about driver capabilities without actually changing driver state. Unlike *VIDIOC_S_FMT* this also works after the overlay has been enabled.

The scaling factor of the overlaid image is implied by the width and height given in struct `v4l2_window` and the size of the cropping rectangle. For more information see *Image Cropping, Insertion and Scaling - the CROP API*.

When simultaneous capturing and overlay is supported and the hardware prohibits different image and window sizes, the size requested first takes precedence. The attempt to capture or overlay as well (*VIDIOC_S_FMT*) may fail with an EBUSY error code or return accordingly modified parameters.

`v4l2_window`

struct `v4l2_window`

struct `v4l2_rect` `w` Size and position of the window relative to the top, left corner of the frame buffer defined with *VIDIOC_S_FBUF*. The window can extend the frame buffer width and height, the `x` and `y` coordinates can be negative, and it can lie completely outside the frame buffer. The driver clips the window accordingly, or if that is not possible, modifies its size and/or position.

enum `v4l2_field` `field` Applications set this field to determine which video field shall be overlaid, typically one of `V4L2_FIELD_ANY` (0), `V4L2_FIELD_TOP`, `V4L2_FIELD_BOTTOM` or `V4L2_FIELD_INTERLACED`. Drivers may have to choose a different field order and return the actual setting here.

__u32 `chromakey` When chroma-keying has been negotiated with *VIDIOC_S_FBUF* applications set this field to the desired pixel value for the chroma key. The format is the same

as the pixel format of the framebuffer (struct `v4l2_framebuffer` `fmt.pixelformat` field), with bytes in host order. E. g. for `V4L2_PIX_FMT_BGR24` the value should be `0xRRGGBB` on a little endian, `0xBBGGRR` on a big endian host.

struct v4l2_clip * clips When chroma-keying has *not* been negotiated and `VIDIOC_G_FBUF` indicated this capability, applications can set this field to point to an array of clipping rectangles.

Like the window coordinates `w`, clipping rectangles are defined relative to the top, left corner of the frame buffer. However clipping rectangles must not extend the frame buffer width and height, and they must not overlap. If possible applications should merge adjacent rectangles. Whether this must create x-y or y-x bands, or the order of rectangles, is not defined. When clip lists are not supported the driver ignores this field. Its contents after calling `VIDIOC_S_FMT` are undefined.

__u32 clipcount When the application set the `clips` field, this field must contain the number of clipping rectangles in the list. When clip lists are not supported the driver ignores this field, its contents after calling `VIDIOC_S_FMT` are undefined. When clip lists are supported but no clipping is desired this field must be set to zero.

void * bitmap When chroma-keying has *not* been negotiated and `VIDIOC_G_FBUF` indicated this capability, applications can set this field to point to a clipping bit mask.

It must be of the same size as the window, `w.width` and `w.height`. Each bit corresponds to a pixel in the overlaid image, which is displayed only when the bit is *set*. Pixel coordinates translate to bits like:

```
((__u8 *) bitmap)[w.width * y + x / 8] & (1 << (x & 7))
```

where $0 \leq x < w.width$ and $0 \leq y < w.height$.²

When a clipping bit mask is not supported the driver ignores this field, its contents after calling `VIDIOC_S_FMT` are undefined. When a bit mask is supported but no clipping is desired this field must be set to `NULL`.

Applications need not create a clip list or bit mask. When they pass both, or despite negotiating chroma-keying, the results are undefined. Regardless of the chosen method, the clipping abilities of the hardware may be limited in quantity or quality. The results when these limits are exceeded are undefined.³

__u8 global_alpha The global alpha value used to blend the framebuffer with video images, if global alpha blending has been negotiated (`V4L2_FBUF_FLAG_GLOBAL_ALPHA`, see `VIDIOC_S_FBUF`, *Frame Buffer Flags*).

Note: This field was added in Linux 2.6.23, extending the structure. However the `VIDIOC_[G|S|TRY]_FMT` ioctls, which take a pointer to a `v4l2_format` parent structure with padding bytes at the end, are not affected.

v4l2_clip

² Should we require `w.width` to be a multiple of eight?

³ When the image is written into frame buffer memory it will be undesirable if the driver clips out less pixels than expected, because the application and graphics system are not aware these regions need to be refreshed. The driver should clip out more pixels or not write the image at all.

struct v4l2_clip⁴

struct v4l2_rect c Coordinates of the clipping rectangle, relative to the top, left corner of the frame buffer. Only window pixels *outside* all clipping rectangles are displayed.

struct v4l2_clip * next Pointer to the next clipping rectangle, NULL when this is the last rectangle. Drivers ignore this field, it cannot be used to pass a linked list of clipping rectangles.

v4l2_rect

struct v4l2_rect

__s32 left Horizontal offset of the top, left corner of the rectangle, in pixels.

__s32 top Vertical offset of the top, left corner of the rectangle, in pixels. Offsets increase to the right and down.

__u32 width Width of the rectangle, in pixels.

__u32 height Height of the rectangle, in pixels.

Enabling Overlay

To start or stop the frame buffer overlay applications call the *ioctl VIDIOC_OVERLAY* ioctl.

3.2.4.3 Video Output Interface

Video output devices encode stills or image sequences as analog video signal. With this interface applications can control the encoding process and move images from user space to the driver.

Conventionally V4L2 video output devices are accessed through character device special files named `/dev/video` and `/dev/video0` to `/dev/video63` with major number 81 and minor numbers 0 to 63. `/dev/video` is typically a symbolic link to the preferred video device.

Note: The same device file names are used also for video capture devices.

⁴ The X Window system defines “regions” which are vectors of `struct BoxRec { short x1, y1, x2, y2; }` with `width = x2 - x1` and `height = y2 - y1`, so one cannot pass X11 clip lists directly.

Querying Capabilities

Devices supporting the video output interface set the `V4L2_CAP_VIDEO_OUTPUT` or `V4L2_CAP_VIDEO_OUTPUT_MPLANE` flag in the capabilities field of struct `v4l2_capability` returned by the *ioctl* `VIDIOC_QUERYCAP` *ioctl*. As secondary device functions they may also support the *raw VBI output* (`V4L2_CAP_VBI_OUTPUT`) interface. At least one of the read/write or streaming I/O methods must be supported. Modulators and audio outputs are optional.

Supplemental Functions

Video output devices shall support *audio output*, *modulator*, *controls*, *cropping and scaling* and *streaming parameter* *ioctls* as needed. The *video output* *ioctls* must be supported by all video output devices.

Image Format Negotiation

The output is determined by cropping and image format parameters. The former select an area of the video picture where the image will appear, the latter how images are stored in memory, i. e. in RGB or YUV format, the number of bits per pixel or width and height. Together they also define how images are scaled in the process.

As usual these parameters are *not* reset at `open()` time to permit Unix tool chains, programming a device and then writing to it as if it was a plain file. Well written V4L2 applications ensure they really get what they want, including cropping and scaling.

Cropping initialization at minimum requires to reset the parameters to defaults. An example is given in *Image Cropping, Insertion and Scaling - the CROP API*.

To query the current image format applications set the type field of a struct `v4l2_format` to `V4L2_BUF_TYPE_VIDEO_OUTPUT` or `V4L2_BUF_TYPE_VIDEO_OUTPUT_MPLANE` and call the *VIDIOC_G_FMT* *ioctl* with a pointer to this structure. Drivers fill the struct `v4l2_pix_format` `pix` or the struct `v4l2_pix_format_mplane` `pix_mp` member of the `fmt` union.

To request different parameters applications set the type field of a struct `v4l2_format` as above and initialize all fields of the struct `v4l2_pix_format` `vbi` member of the `fmt` union, or better just modify the results of *VIDIOC_G_FMT*, and call the *VIDIOC_S_FMT* *ioctl* with a pointer to this structure. Drivers may adjust the parameters and finally return the actual parameters as *VIDIOC_G_FMT* does.

Like *VIDIOC_S_FMT* the *VIDIOC_TRY_FMT* *ioctl* can be used to learn about hardware limitations without disabling I/O or possibly time consuming hardware preparations.

The contents of struct `v4l2_pix_format` and struct `v4l2_pix_format_mplane` are discussed in *Image Formats*. See also the specification of the *VIDIOC_G_FMT*, *VIDIOC_S_FMT* and *VIDIOC_TRY_FMT* *ioctls* for details. Video output devices must implement both the *VIDIOC_G_FMT* and *VIDIOC_S_FMT* *ioctl*, even if *VIDIOC_S_FMT* ignores all requests and always returns default parameters as *VIDIOC_G_FMT* does. *VIDIOC_TRY_FMT* is optional.

Writing Images

A video output device may support the *write()* function and/or streaming (*memory mapping* or *user pointer*) I/O. See *Input/Output* for details.

3.2.4.4 Video Output Overlay Interface

Also known as On-Screen Display (OSD)

Some video output devices can overlay a framebuffer image onto the outgoing video signal. Applications can set up such an overlay using this interface, which borrows structures and ioctls of the *Video Overlay* interface.

The OSD function is accessible through the same character special file as the *Video Output* function.

Note: The default function of such a `/dev/video` device is video capturing or output. The OSD function is only available after calling the `VIDIOC_S_FMT` ioctl.

Querying Capabilities

Devices supporting the *Video Output Overlay* interface set the `V4L2_CAP_VIDEO_OUTPUT_OVERLAY` flag in the capabilities field of struct `v4l2_capability` returned by the *ioctl* `VIDIOC_QUERYCAP` ioctl.

Framebuffer

Contrary to the *Video Overlay* interface the framebuffer is normally implemented on the TV card and not the graphics card. On Linux it is accessible as a framebuffer device (`/dev/fbN`). Given a V4L2 device, applications can find the corresponding framebuffer device by calling the `VIDIOC_G_FBUF` ioctl. It returns, amongst other information, the physical address of the framebuffer in the base field of struct `v4l2_framebuffer`. The framebuffer device ioctl `FBIOGET_FSCREENINFO` returns the same address in the `smem_start` field of struct `fb_fix_screeninfo`. The `FBIOGET_FSCREENINFO` ioctl and struct `fb_fix_screeninfo` are defined in the `linux/fb.h` header file.

The width and height of the framebuffer depends on the current video standard. A V4L2 driver may reject attempts to change the video standard (or any other ioctl which would imply a framebuffer size change) with an `EBUSY` error code until all applications closed the framebuffer device.

Example: Finding a framebuffer device for OSD

```

#include <linux/fb.h>

struct v4l2_framebuffer fbuf;
unsigned int i;
int fb_fd;

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

for (i = 0; i < 30; i++) {
    char dev_name[16];
    struct fb_fix_screeninfo si;

    snprintf(dev_name, sizeof(dev_name), "/dev/fb%u", i);

    fb_fd = open(dev_name, O_RDWR);
    if (-1 == fb_fd) {
        switch (errno) {
            case ENOENT: /* no such file */
            case ENXIO: /* no driver */
                continue;

            default:
                perror("open");
                exit(EXIT_FAILURE);
        }
    }

    if (0 == ioctl(fb_fd, FBIOGET_FSCREENINFO, &si)) {
        if (si.smem_start == (unsigned long)fbuf.base)
            break;
    } else {
        /* Apparently not a framebuffer device. */
    }

    close(fb_fd);
    fb_fd = -1;
}

/* fb_fd is the file descriptor of the framebuffer device
   for the video output overlay, or -1 if no device was found. */

```

Overlay Window and Scaling

The overlay is controlled by source and target rectangles. The source rectangle selects a subsection of the framebuffer image to be overlaid, the target rectangle an area in the outgoing video signal where the image will appear. Drivers may or may not support scaling, and arbitrary sizes and positions of these rectangles. Further drivers may support any (or none) of the clipping/blending methods defined for the *Video Overlay* interface.

A struct `v4l2_window` defines the size of the source rectangle, its position in the framebuffer and the clipping/blending method to be used for the overlay. To get the current parameters applications set the `type` field of a struct `v4l2_format` to `V4L2_BUF_TYPE_VIDEO_OUTPUT_OVERLAY` and call the `VIDIOC_G_FMT` ioctl. The driver fills the struct `v4l2_window` substructure named `win`. It is not possible to retrieve a previously programmed clipping list or bitmap.

To program the source rectangle applications set the `type` field of a struct `v4l2_format` to `V4L2_BUF_TYPE_VIDEO_OUTPUT_OVERLAY`, initialize the `win` substructure and call the `VIDIOC_S_FMT` ioctl. The driver adjusts the parameters against hardware limits and returns the actual parameters as `VIDIOC_G_FMT` does. Like `VIDIOC_S_FMT`, the `VIDIOC_TRY_FMT` ioctl can be used to learn about driver capabilities without actually changing driver state. Unlike `VIDIOC_S_FMT` this also works after the overlay has been enabled.

A struct `v4l2_crop` defines the size and position of the target rectangle. The scaling factor of the overlay is implied by the width and height given in struct `v4l2_window` and struct `v4l2_crop`. The cropping API applies to *Video Output* and *Video Output Overlay* devices in the same way as to *Video Capture* and *Video Overlay* devices, merely reversing the direction of the data flow. For more information see *Image Cropping, Insertion and Scaling - the CROP API*.

Enabling Overlay

There is no V4L2 ioctl to enable or disable the overlay, however the framebuffer interface of the driver may support the `FBIOBLANK` ioctl.

3.2.4.5 Video Memory-To-Memory Interface

A V4L2 memory-to-memory device can compress, decompress, transform, or otherwise convert video data from one format into another format, in memory. Such memory-to-memory devices set the `V4L2_CAP_VIDEO_M2M` or `V4L2_CAP_VIDEO_M2M_MPLANE` capability. Examples of memory-to-memory devices are codecs, scalers, deinterlacers or format converters (i.e. converting from YUV to RGB).

A memory-to-memory video node acts just like a normal video node, but it supports both output (sending frames from memory to the hardware) and capture (receiving the processed frames from the hardware into memory) stream I/O. An application will have to setup the stream I/O for both sides and finally call `VIDIOC_STREAMON` for both capture and output to start the hardware.

Memory-to-memory devices function as a shared resource: you can open the video node multiple times, each application setting up their own properties that are local to the file handle, and each can use it independently from the others. The driver will arbitrate access to the hardware and reprogram it whenever another file handler gets access. This is different from the usual video node behavior where the video properties are global to the device (i.e. changing something through one file handle is visible through another file handle).

One of the most common memory-to-memory device is the codec. Codecs are more complicated than most and require additional setup for their codec parameters. This is done through codec controls. See *Codec Control Reference*. More details on how to use codec memory-to-memory devices are given in the following sections.

Memory-to-Memory Stateful Video Decoder Interface

A stateful video decoder takes complete chunks of the bytestream (e.g. Annex-B H.264/HEVC stream, raw VP8/9 stream) and decodes them into raw video frames in display order. The decoder is expected not to require any additional information from the client to process these buffers.

Performing software parsing, processing etc. of the stream in the driver in order to support this interface is strongly discouraged. In case such operations are needed, use of the Stateless Video Decoder Interface (in development) is strongly advised.

Conventions and Notations Used in This Document

1. The general V4L2 API rules apply if not specified in this document otherwise.
2. The meaning of words “must”, “may”, “should”, etc. is as per [RFC 2119](#).
3. All steps not marked “optional” are required.
4. `VIDIOC_G_EXT_CTRL()` and `VIDIOC_S_EXT_CTRL()` may be used interchangeably with `VIDIOC_G_CTRL()` and `VIDIOC_S_CTRL()`, unless specified otherwise.
5. Single-planar API (see *Single- and multi-planar APIs*) and applicable structures may be used interchangeably with multi-planar API, unless specified otherwise, depending on decoder capabilities and following the general V4L2 guidelines.
6. $i = [a..b]$: sequence of integers from a to b, inclusive, i.e. $i = [0..2]$: $i = 0, 1, 2$.
7. Given an OUTPUT buffer A, then A' represents a buffer on the CAPTURE queue containing data that resulted from processing buffer A.

Glossary

CAPTURE the destination buffer queue; for decoders, the queue of buffers containing decoded frames; for encoders, the queue of buffers containing an encoded bytestream; `V4L2_BUF_TYPE_VIDEO_CAPTURE` or `V4L2_BUF_TYPE_VIDEO_CAPTURE_MPLANE`; data is captured from the hardware into CAPTURE buffers.

client the application communicating with the decoder or encoder implementing this interface.

coded format encoded/compressed video bytestream format (e.g. H.264, VP8, etc.); see also: raw format.

coded height height for given coded resolution.

coded resolution stream resolution in pixels aligned to codec and hardware requirements; typically visible resolution rounded up to full macroblocks; see also: visible resolution.

coded width width for given coded resolution.

decode order the order in which frames are decoded; may differ from display order if the coded format includes a feature of frame reordering; for decoders, OUTPUT buffers must be queued by the client in decode order; for encoders CAPTURE buffers must be returned by the encoder in decode order.

destination data resulting from the decode process; see CAPTURE.

display order the order in which frames must be displayed; for encoders, OUTPUT buffers must be queued by the client in display order; for decoders, CAPTURE buffers must be returned by the decoder in display order.

DPB Decoded Picture Buffer; an H.264/HEVC term for a buffer that stores a decoded raw frame available for reference in further decoding steps.

EOS end of stream.

IDR Instantaneous Decoder Refresh; a type of a keyframe in an H.264/HEVC-encoded stream, which clears the list of earlier reference frames (DPBs).

keyframe an encoded frame that does not reference frames decoded earlier, i.e. can be decoded fully on its own.

macroblock a processing unit in image and video compression formats based on linear block transforms (e.g. H.264, VP8, VP9); codec-specific, but for most of popular codecs the size is 16x16 samples (pixels).

OUTPUT the source buffer queue; for decoders, the queue of buffers containing an encoded bytestream; for encoders, the queue of buffers containing raw frames; V4L2_BUF_TYPE_VIDEO_OUTPUT or V4L2_BUF_TYPE_VIDEO_OUTPUT_MPLANE; the hardware is fed with data from OUTPUT buffers.

PPS Picture Parameter Set; a type of metadata entity in an H.264/HEVC bytestream.

raw format uncompressed format containing raw pixel data (e.g. YUV, RGB formats).

resume point a point in the bytestream from which decoding may start/continue, without any previous state/data present, e.g.: a keyframe (VP8/VP9) or SPS/PPS/IDR sequence (H.264/HEVC); a resume point is required to start decode of a new stream, or to resume decoding after a seek.

source data fed to the decoder or encoder; see OUTPUT.

source height height in pixels for given source resolution; relevant to encoders only.

source resolution resolution in pixels of source frames being source to the encoder and subject to further cropping to the bounds of visible resolution; relevant to encoders only.

source width width in pixels for given source resolution; relevant to encoders only.

SPS Sequence Parameter Set; a type of metadata entity in an H.264/HEVC bytestream.

stream metadata additional (non-visual) information contained inside encoded bytestream; for example: coded resolution, visible resolution, codec profile.

visible height height for given visible resolution; display height.

visible resolution stream resolution of the visible picture, in pixels, to be used for display purposes; must be smaller or equal to coded resolution; display resolution.

visible width width for given visible resolution; display width.

State Machine

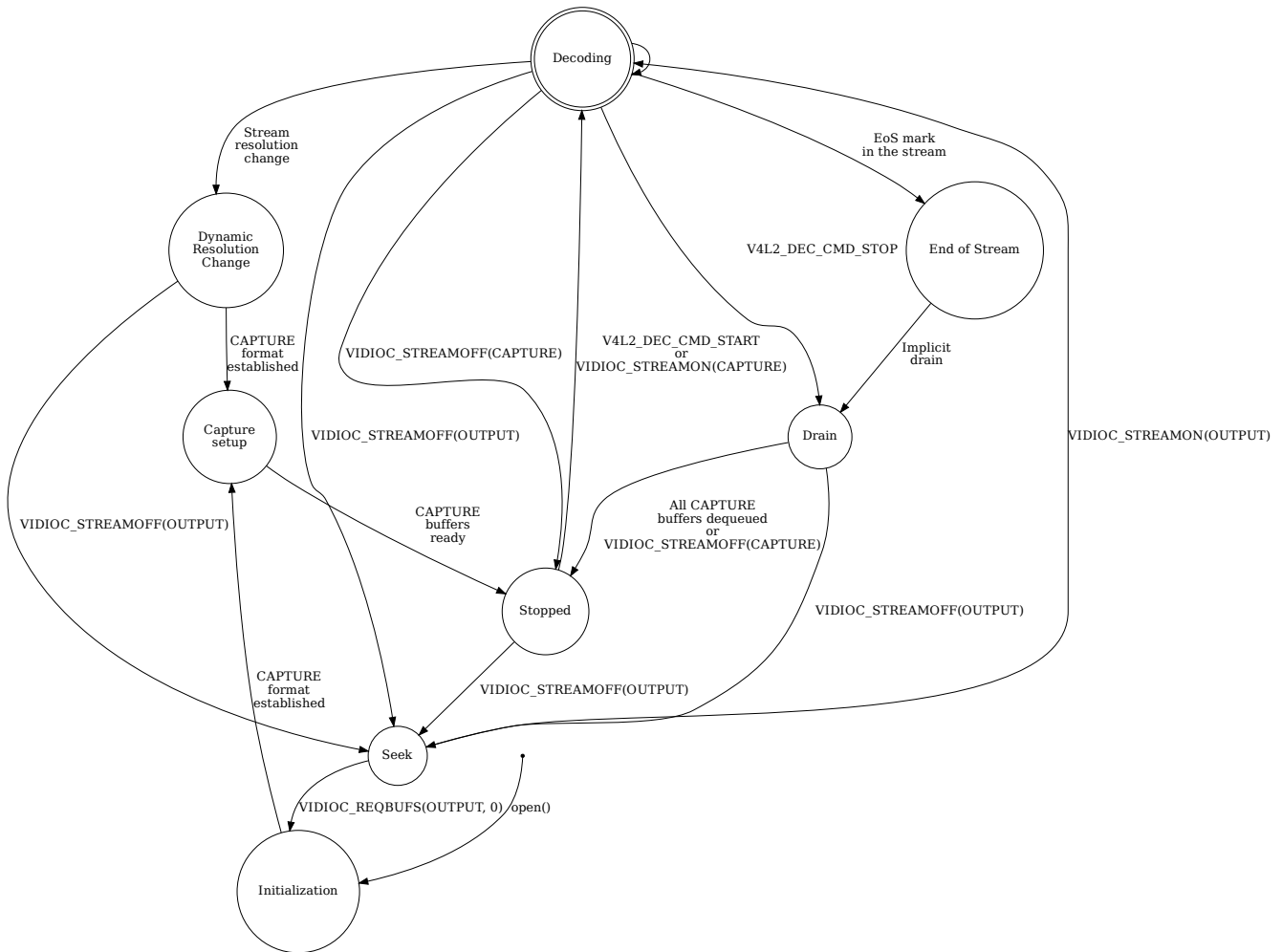


Fig. 8: Decoder State Machine

Querying Capabilities

- To enumerate the set of coded formats supported by the decoder, the client may call `VIDIIOC_ENUM_FMT()` on `OUTPUT`.
 - The full set of supported formats will be returned, regardless of the format set on `CAPTURE`.
 - Check the `flags` field of `v4l2_fmtdesc` for more information about the decoder's capabilities with respect to each coded format. In particular whether or not the decoder has a full-fledged bytestream parser and if the decoder supports dynamic resolution changes.
- To enumerate the set of supported raw formats, the client may call `VIDIIOC_ENUM_FMT()` on `CAPTURE`.
 - Only the formats supported for the format currently active on `OUTPUT` will be returned.

- In order to enumerate raw formats supported by a given coded format, the client must first set that coded format on OUTPUT and then enumerate formats on CAPTURE.
3. The client may use `VIDIOC_ENUM_FRAMESIZES()` to detect supported resolutions for a given format, passing desired pixel format in `v4l2_frmsizeenum pixel_format`.
 - Values returned by `VIDIOC_ENUM_FRAMESIZES()` for a coded pixel format will include all possible coded resolutions supported by the decoder for given coded pixel format.
 - Values returned by `VIDIOC_ENUM_FRAMESIZES()` for a raw pixel format will include all possible frame buffer resolutions supported by the decoder for given raw pixel format and the coded format currently set on OUTPUT.
 4. Supported profiles and levels for the coded format currently set on OUTPUT, if applicable, may be queried using their respective controls via `VIDIOC_QUERYCTRL()`.

Initialization

1. Set the coded format on OUTPUT via `VIDIOC_S_FMT()`.
 - **Required fields:**
 - type** a `V4L2_BUF_TYPE_*` enum appropriate for OUTPUT.
 - pixelformat** a coded pixel format.
 - width, height** coded resolution of the stream; required only if it cannot be parsed from the stream for the given coded format; otherwise the decoder will use this resolution as a placeholder resolution that will likely change as soon as it can parse the actual coded resolution from the stream.
 - sizeimage** desired size of OUTPUT buffers; the decoder may adjust it to match hardware requirements.
 - other fields** follow standard semantics.
 - **Return fields:**
 - sizeimage** adjusted size of OUTPUT buffers.
 - The CAPTURE format will be updated with an appropriate frame buffer resolution instantly based on the width and height returned by `VIDIOC_S_FMT()`. However, for coded formats that include stream resolution information, after the decoder is done parsing the information from the stream, it will update the CAPTURE format with new values and signal a source change event, regardless of whether they match the values set by the client or not.

Important: Changing the OUTPUT format may change the currently set CAPTURE format. How the new CAPTURE format is determined is up to the decoder and the client must ensure it matches its needs afterwards.

2. Allocate source (bytestream) buffers via `VIDIOC_REQBUFS()` on OUTPUT.
 - **Required fields:**
 - count** requested number of buffers to allocate; greater than zero.

type a `V4L2_BUF_TYPE_*` enum appropriate for OUTPUT.

memory follows standard semantics.

- **Return fields:**

count the actual number of buffers allocated.

Warning: The actual number of allocated buffers may differ from the count given. The client must check the updated value of count after the call returns.

Alternatively, `VIDIOC_CREATE_BUFS()` on the OUTPUT queue can be used to have more control over buffer allocation.

- **Required fields:**

count requested number of buffers to allocate; greater than zero.

type a `V4L2_BUF_TYPE_*` enum appropriate for OUTPUT.

memory follows standard semantics.

format follows standard semantics.

- **Return fields:**

count adjusted to the number of allocated buffers.

Warning: The actual number of allocated buffers may differ from the count given. The client must check the updated value of count after the call returns.

3. Start streaming on the OUTPUT queue via `VIDIOC_STREAMON()`.

4. **This step only applies to coded formats that contain resolution information in the stream.** Continue queuing/dequeuing bytestream buffers to/from the OUTPUT queue via `VIDIOC_QBUF()` and `VIDIOC_DQBUF()`. The buffers will be processed and returned to the client in order, until required metadata to configure the CAPTURE queue are found. This is indicated by the decoder sending a `V4L2_EVENT_SOURCE_CHANGE` event with changes set to `V4L2_EVENT_SRC_CH_RESOLUTION`.

- It is not an error if the first buffer does not contain enough data for this to occur. Processing of the buffers will continue as long as more data is needed.
- If data in a buffer that triggers the event is required to decode the first frame, it will not be returned to the client, until the initialization sequence completes and the frame is decoded.
- If the client has not set the coded resolution of the stream on its own, calling `VIDIOC_G_FMT()`, `VIDIOC_S_FMT()`, `VIDIOC_TRY_FMT()` or `VIDIOC_REQBUFS()` on the CAPTURE queue will not return the real values for the stream until a `V4L2_EVENT_SOURCE_CHANGE` event with changes set to `V4L2_EVENT_SRC_CH_RESOLUTION` is signaled.

Important: Any client query issued after the decoder queues the event will return values applying to the just parsed stream, including queue formats, selection rectangles and

controls.

Note: A client capable of acquiring stream parameters from the bytestream on its own may attempt to set the width and height of the `OUTPUT` format to non-zero values matching the coded size of the stream, skip this step and continue with the *Capture Setup* sequence. However, it must not rely on any driver queries regarding stream parameters, such as selection rectangles and controls, since the decoder has not parsed them from the stream yet. If the values configured by the client do not match those parsed by the decoder, a *Dynamic Resolution Change* will be triggered to reconfigure them.

Note: No decoded frames are produced during this phase.

5. Continue with the *Capture Setup* sequence.

Capture Setup

1. Call `VIDIOC_G_FMT()` on the `CAPTURE` queue to get format for the destination buffers parsed/decoded from the bytestream.
 - **Required fields:**
 - type** a `V4L2_BUF_TYPE_*` enum appropriate for `CAPTURE`.
 - **Return fields:**
 - width, height** frame buffer resolution for the decoded frames.
 - pixelformat** pixel format for decoded frames.
 - num_planes (for `_MPLANE` type only)** number of planes for `pixelformat`.
 - sizeimage, bytesperline** as per standard semantics; matching frame buffer format.
-

Note: The value of `pixelformat` may be any pixel format supported by the decoder for the current stream. The decoder should choose a preferred/optimal format for the default configuration. For example, a YUV format may be preferred over an RGB format if an additional conversion step would be required for the latter.

2. **Optional.** Acquire the visible resolution via `VIDIOC_G_SELECTION()`.
 - **Required fields:**
 - type** a `V4L2_BUF_TYPE_*` enum appropriate for `CAPTURE`.
 - target** set to `V4L2_SEL_TGT_COMPOSE`.
 - **Return fields:**
 - r.left, r.top, r.width, r.height** the visible rectangle; it must fit within the frame buffer resolution returned by `VIDIOC_G_FMT()` on `CAPTURE`.
 - The following selection targets are supported on `CAPTURE`:

- V4L2_SEL_TGT_CROP_BOUNDS** corresponds to the coded resolution of the stream.
- V4L2_SEL_TGT_CROP_DEFAULT** the rectangle covering the part of the CAPTURE buffer that contains meaningful picture data (visible area); width and height will be equal to the visible resolution of the stream.
- V4L2_SEL_TGT_CROP** the rectangle within the coded resolution to be output to CAPTURE; defaults to V4L2_SEL_TGT_CROP_DEFAULT; read-only on hardware without additional compose/scaling capabilities.
- V4L2_SEL_TGT_COMPOSE_BOUNDS** the maximum rectangle within a CAPTURE buffer, which the cropped frame can be composed into; equal to V4L2_SEL_TGT_CROP if the hardware does not support compose/scaling.
- V4L2_SEL_TGT_COMPOSE_DEFAULT** equal to V4L2_SEL_TGT_CROP.
- V4L2_SEL_TGT_COMPOSE** the rectangle inside a CAPTURE buffer into which the cropped frame is written; defaults to V4L2_SEL_TGT_COMPOSE_DEFAULT; read-only on hardware without additional compose/scaling capabilities.
- V4L2_SEL_TGT_COMPOSE_PADDED** the rectangle inside a CAPTURE buffer which is overwritten by the hardware; equal to V4L2_SEL_TGT_COMPOSE if the hardware does not write padding pixels.

Warning: The values are guaranteed to be meaningful only after the decoder successfully parses the stream metadata. The client must not rely on the query before that happens.

3. **Optional.** Enumerate CAPTURE formats via VIDIOC_ENUM_FMT() on the CAPTURE queue. Once the stream information is parsed and known, the client may use this ioctl to discover which raw formats are supported for given stream and select one of them via VIDIOC_S_FMT().

Important: The decoder will return only formats supported for the currently established coded format, as per the OUTPUT format and/or stream metadata parsed in this initialization sequence, even if more formats may be supported by the decoder in general. In other words, the set returned will be a subset of the initial query mentioned in the *Querying Capabilities* section.

For example, a decoder may support YUV and RGB formats for resolutions 1920x1088 and lower, but only YUV for higher resolutions (due to hardware limitations). After parsing a resolution of 1920x1088 or lower, VIDIOC_ENUM_FMT() may return a set of YUV and RGB pixel formats, but after parsing resolution higher than 1920x1088, the decoder will not return RGB, unsupported for this resolution.

However, subsequent resolution change event triggered after discovering a resolution change within the same stream may switch the stream into a lower resolution and VIDIOC_ENUM_FMT() would return RGB formats again in that case.

4. **Optional.** Set the CAPTURE format via VIDIOC_S_FMT() on the CAPTURE queue. The client may choose a different format than selected/suggested by the decoder in VIDIOC_G_FMT().

- **Required fields:**

type a V4L2_BUF_TYPE_* enum appropriate for CAPTURE.

pixelformat a raw pixel format.

width, height frame buffer resolution of the decoded stream; typically unchanged from what was returned with VIDIOC_G_FMT(), but it may be different if the hardware supports composition and/or scaling.

- Setting the CAPTURE format will reset the compose selection rectangles to their default values, based on the new resolution, as described in the previous step.
5. **Optional.** Set the compose rectangle via VIDIOC_S_SELECTION() on the CAPTURE queue if it is desired and if the decoder has compose and/or scaling capabilities.

- **Required fields:**

type a V4L2_BUF_TYPE_* enum appropriate for CAPTURE.

target set to V4L2_SEL_TGT_COMPOSE.

r.left, r.top, r.width, r.height the rectangle inside a CAPTURE buffer into which the cropped frame is written; defaults to V4L2_SEL_TGT_COMPOSE_DEFAULT; read-only on hardware without additional compose/scaling capabilities.

- **Return fields:**

r.left, r.top, r.width, r.height the visible rectangle; it must fit within the frame buffer resolution returned by VIDIOC_G_FMT() on CAPTURE.

Warning: The decoder may adjust the compose rectangle to the nearest supported one to meet codec and hardware requirements. The client needs to check the adjusted rectangle returned by VIDIOC_S_SELECTION().

6. If all the following conditions are met, the client may resume the decoding instantly:
- sizeimage of the new format (determined in previous steps) is less than or equal to the size of currently allocated buffers,
 - the number of buffers currently allocated is greater than or equal to the minimum number of buffers acquired in previous steps. To fulfill this requirement, the client may use VIDIOC_CREATE_BUFS() to add new buffers.

In that case, the remaining steps do not apply and the client may resume the decoding by one of the following actions:

- if the CAPTURE queue is streaming, call VIDIOC_DECODER_CMD() with the V4L2_DEC_CMD_START command,
- if the CAPTURE queue is not streaming, call VIDIOC_STREAMON() on the CAPTURE queue.

However, if the client intends to change the buffer set, to lower memory usage or for any other reasons, it may be achieved by following the steps below.

7. **If the CAPTURE queue is streaming,** keep queuing and dequeuing buffers on the CAPTURE queue until a buffer marked with the V4L2_BUF_FLAG_LAST flag is dequeued.
8. **If the CAPTURE queue is streaming,** call VIDIOC_STREAMOFF() on the CAPTURE queue to stop streaming.

Warning: The OUTPUT queue must remain streaming. Calling VIDIOC_STREAMOFF() on it would abort the sequence and trigger a seek.

9. **If the CAPTURE queue has buffers allocated**, free the CAPTURE buffers using VIDIOC_REQBUFS().

- **Required fields:**

count set to 0.

type a V4L2_BUF_TYPE_* enum appropriate for CAPTURE.

memory follows standard semantics.

10. Allocate CAPTURE buffers via VIDIOC_REQBUFS() on the CAPTURE queue.

- **Required fields:**

count requested number of buffers to allocate; greater than zero.

type a V4L2_BUF_TYPE_* enum appropriate for CAPTURE.

memory follows standard semantics.

- **Return fields:**

count actual number of buffers allocated.

Warning: The actual number of allocated buffers may differ from the count given. The client must check the updated value of count after the call returns.

Note: To allocate more than the minimum number of buffers (for pipeline depth), the client may query the V4L2_CID_MIN_BUFFERS_FOR_CAPTURE control to get the minimum number of buffers required, and pass the obtained value plus the number of additional buffers needed in the count field to VIDIOC_REQBUFS().

Alternatively, VIDIOC_CREATE_BUFS() on the CAPTURE queue can be used to have more control over buffer allocation. For example, by allocating buffers larger than the current CAPTURE format, future resolution changes can be accommodated.

- **Required fields:**

count requested number of buffers to allocate; greater than zero.

type a V4L2_BUF_TYPE_* enum appropriate for CAPTURE.

memory follows standard semantics.

format a format representing the maximum framebuffer resolution to be accommodated by newly allocated buffers.

- **Return fields:**

count adjusted to the number of allocated buffers.

Warning: The actual number of allocated buffers may differ from the count given. The client must check the updated value of count after the call returns.

Note: To allocate buffers for a format different than parsed from the stream metadata, the client must proceed as follows, before the metadata parsing is initiated:

- set width and height of the OUTPUT format to desired coded resolution to let the decoder configure the CAPTURE format appropriately,
- query the CAPTURE format using `VIDIOC_G_FMT()` and save it until this step.

The format obtained in the query may be then used with `VIDIOC_CREATE_BUFS()` in this step to allocate the buffers.

11. Call `VIDIOC_STREAMON()` on the CAPTURE queue to start decoding frames.

Decoding

This state is reached after the *Capture Setup* sequence finishes successfully. In this state, the client queues and dequeues buffers to both queues via `VIDIOC_QBUF()` and `VIDIOC_DQBUF()`, following the standard semantics.

The content of the source OUTPUT buffers depends on the active coded pixel format and may be affected by codec-specific extended controls, as stated in the documentation of each format.

Both queues operate independently, following the standard behavior of V4L2 buffer queues and memory-to-memory devices. In addition, the order of decoded frames dequeued from the CAPTURE queue may differ from the order of queuing coded frames to the OUTPUT queue, due to properties of the selected coded format, e.g. frame reordering.

The client must not assume any direct relationship between CAPTURE and OUTPUT buffers and any specific timing of buffers becoming available to dequeue. Specifically:

- a buffer queued to OUTPUT may result in no buffers being produced on CAPTURE (e.g. if it does not contain encoded data, or if only metadata syntax structures are present in it),
- a buffer queued to OUTPUT may result in more than one buffer produced on CAPTURE (if the encoded data contained more than one frame, or if returning a decoded frame allowed the decoder to return a frame that preceded it in decode, but succeeded it in the display order),
- a buffer queued to OUTPUT may result in a buffer being produced on CAPTURE later into decode process, and/or after processing further OUTPUT buffers, or be returned out of order, e.g. if display reordering is used,
- buffers may become available on the CAPTURE queue without additional buffers queued to OUTPUT (e.g. during drain or EOS), because of the OUTPUT buffers queued in the past whose decoding results are only available at later time, due to specifics of the decoding process.

Note: To allow matching decoded CAPTURE buffers with OUTPUT buffers they originated from, the client can set the timestamp field of the `v4l2_buffer` struct when queuing an OUTPUT

buffer. The CAPTURE buffer(s), which resulted from decoding that OUTPUT buffer will have their timestamp field set to the same value when dequeued.

In addition to the straightforward case of one OUTPUT buffer producing one CAPTURE buffer, the following cases are defined:

- one OUTPUT buffer generates multiple CAPTURE buffers: the same OUTPUT timestamp will be copied to multiple CAPTURE buffers.
- multiple OUTPUT buffers generate one CAPTURE buffer: timestamp of the OUTPUT buffer queued first will be copied.
- the decoding order differs from the display order (i.e. the CAPTURE buffers are out-of-order compared to the OUTPUT buffers): CAPTURE timestamps will not retain the order of OUTPUT timestamps.

During the decoding, the decoder may initiate one of the special sequences, as listed below. The sequences will result in the decoder returning all the CAPTURE buffers that originated from all the OUTPUT buffers processed before the sequence started. Last of the buffers will have the `V4L2_BUF_FLAG_LAST` flag set. To determine the sequence to follow, the client must check if there is any pending event and:

- if a `V4L2_EVENT_SOURCE_CHANGE` event with `changes` set to `V4L2_EVENT_SRC_CH_RESOLUTION` is pending, the *Dynamic Resolution Change* sequence needs to be followed,
- if a `V4L2_EVENT_EOS` event is pending, the *End of Stream* sequence needs to be followed.

Some of the sequences can be intermixed with each other and need to be handled as they happen. The exact operation is documented for each sequence.

Should a decoding error occur, it will be reported to the client with the level of details depending on the decoder capabilities. Specifically:

- the CAPTURE buffer that contains the results of the failed decode operation will be returned with the `V4L2_BUF_FLAG_ERROR` flag set,
- if the decoder is able to precisely report the OUTPUT buffer that triggered the error, such buffer will be returned with the `V4L2_BUF_FLAG_ERROR` flag set.

In case of a fatal failure that does not allow the decoding to continue, any further operations on corresponding decoder file handle will return the `-EIO` error code. The client may close the file handle and open a new one, or alternatively reinitialize the instance by stopping streaming on both queues, releasing all buffers and performing the Initialization sequence again.

Seek

Seek is controlled by the OUTPUT queue, as it is the source of coded data. The seek does not require any specific operation on the CAPTURE queue, but it may be affected as per normal decoder operation.

1. Stop the OUTPUT queue to begin the seek sequence via `VIDIOC_STREAMOFF()`.

- **Required fields:**

type a `V4L2_BUF_TYPE_*` enum appropriate for OUTPUT.

- The decoder will drop all the pending OUTPUT buffers and they must be treated as returned to the client (following standard semantics).
2. Restart the OUTPUT queue via `VIDIOC_STREAMON()`.
 - **Required fields:**
 - type** a `V4L2_BUF_TYPE_*` enum appropriate for OUTPUT.
 - The decoder will start accepting new source bytestream buffers after the call returns.
 3. Start queuing buffers containing coded data after the seek to the OUTPUT queue until a suitable resume point is found.

Note: There is no requirement to begin queuing coded data starting exactly from a resume point (e.g. SPS or a keyframe). Any queued OUTPUT buffers will be processed and returned to the client until a suitable resume point is found. While looking for a resume point, the decoder should not produce any decoded frames into CAPTURE buffers.

Some hardware is known to mishandle seeks to a non-resume point. Such an operation may result in an unspecified number of corrupted decoded frames being made available on the CAPTURE queue. Drivers must ensure that no fatal decoding errors or crashes occur, and implement any necessary handling and workarounds for hardware issues related to seek operations.

Warning: In case of the H.264/HEVC codec, the client must take care not to seek over a change of SPS/PPS. Even though the target frame could be a keyframe, the stale SPS/PPS inside decoder state would lead to undefined results when decoding. Although the decoder must handle that case without a crash or a fatal decode error, the client must not expect a sensible decode output.

If the hardware can detect such corrupted decoded frames, then corresponding buffers will be returned to the client with the `V4L2_BUF_FLAG_ERROR` set. See the *Decoding* section for further description of decode error reporting.

4. After a resume point is found, the decoder will start returning CAPTURE buffers containing decoded frames.

Important: A seek may result in the *Dynamic Resolution Change* sequence being initiated, due to the seek target having decoding parameters different from the part of the stream decoded before the seek. The sequence must be handled as per normal decoder operation.

Warning: It is not specified when the CAPTURE queue starts producing buffers containing decoded data from the OUTPUT buffers queued after the seek, as it operates independently from the OUTPUT queue.

The decoder may return a number of remaining CAPTURE buffers containing decoded frames originating from the OUTPUT buffers queued before the seek sequence is performed.

The `VIDIOC_STREAMOFF` operation discards any remaining queued `OUTPUT` buffers, which means that not all of the `OUTPUT` buffers queued before the seek sequence may have matching `CAPTURE` buffers produced. For example, given the sequence of operations on the `OUTPUT` queue:

`QBUF(A), QBUF(B), STREAMOFF(), STREAMON(), QBUF(G), QBUF(H),`

any of the following results on the `CAPTURE` queue is allowed:

`{A', B', G', H'}, {A', G', H'}, {G', H'}.`

To determine the `CAPTURE` buffer containing the first decoded frame after the seek, the client may observe the timestamps to match the `CAPTURE` and `OUTPUT` buffers or use `V4L2_DEC_CMD_STOP` and `V4L2_DEC_CMD_START` to drain the decoder.

Note: To achieve instantaneous seek, the client may restart streaming on the `CAPTURE` queue too to discard decoded, but not yet dequeued buffers.

Dynamic Resolution Change

Streams that include resolution metadata in the bytestream may require switching to a different resolution during the decoding.

Note: Not all decoders can detect resolution changes. Those that do set the `V4L2_FMT_FLAG_DYN_RESOLUTION` flag for the coded format when `VIDIOC_ENUM_FMT()` is called.

The sequence starts when the decoder detects a coded frame with one or more of the following parameters different from those previously established (and reflected by corresponding queries):

- coded resolution (`OUTPUT` width and height),
- visible resolution (selection rectangles),
- the minimum number of buffers needed for decoding,
- bit-depth of the bitstream has been changed.

Whenever that happens, the decoder must proceed as follows:

1. After encountering a resolution change in the stream, the decoder sends a `V4L2_EVENT_SOURCE_CHANGE` event with changes set to `V4L2_EVENT_SRC_CH_RESOLUTION`.

Important: Any client query issued after the decoder queues the event will return values applying to the stream after the resolution change, including queue formats, selection rectangles and controls.

2. The decoder will then process and decode all remaining buffers from before the resolution change point.

- The last buffer from before the change must be marked with the `V4L2_BUF_FLAG_LAST` flag, similarly to the *Drain* sequence above.

Warning: The last buffer may be empty (with `v4l2_buffer.bytesused = 0`) and in that case it must be ignored by the client, as it does not contain a decoded frame.

Note: Any attempt to dequeue more CAPTURE buffers beyond the buffer marked with `V4L2_BUF_FLAG_LAST` will result in a `-EPIPE` error from `VIDIOC_DQBUF()`.

The client must continue the sequence as described below to continue the decoding process.

1. Dequeue the source change event.

Important: A source change triggers an implicit decoder drain, similar to the explicit *Drain* sequence. The decoder is stopped after it completes. The decoding process must be resumed with either a pair of calls to `VIDIOC_STREAMOFF()` and `VIDIOC_STREAMON()` on the CAPTURE queue, or a call to `VIDIOC_DECODER_CMD()` with the `V4L2_DEC_CMD_START` command.

2. Continue with the *Capture Setup* sequence.

Note: During the resolution change sequence, the OUTPUT queue must remain streaming. Calling `VIDIOC_STREAMOFF()` on the OUTPUT queue would abort the sequence and initiate a seek.

In principle, the OUTPUT queue operates separately from the CAPTURE queue and this remains true for the duration of the entire resolution change sequence as well.

The client should, for best performance and simplicity, keep queuing/dequeuing buffers to/from the OUTPUT queue even while processing this sequence.

Drain

To ensure that all queued OUTPUT buffers have been processed and related CAPTURE buffers are given to the client, the client must follow the drain sequence described below. After the drain sequence ends, the client has received all decoded frames for all OUTPUT buffers queued before the sequence was started.

1. Begin drain by issuing `VIDIOC_DECODER_CMD()`.

- **Required fields:**

cmd set to `V4L2_DEC_CMD_STOP`.

flags set to 0.

pts set to 0.

Warning: The sequence can be only initiated if both OUTPUT and CAPTURE queues are streaming. For compatibility reasons, the call to VIDIOC_DECODER_CMD() will not fail even if any of the queues is not streaming, but at the same time it will not initiate the *Drain* sequence and so the steps described below would not be applicable.

2. Any OUTPUT buffers queued by the client before the VIDIOC_DECODER_CMD() was issued will be processed and decoded as normal. The client must continue to handle both queues independently, similarly to normal decode operation. This includes:
 - handling any operations triggered as a result of processing those buffers, such as the *Dynamic Resolution Change* sequence, before continuing with the drain sequence,
 - queuing and dequeuing CAPTURE buffers, until a buffer marked with the V4L2_BUF_FLAG_LAST flag is dequeued,

Warning: The last buffer may be empty (with v4l2_buffer bytesused = 0) and in that case it must be ignored by the client, as it does not contain a decoded frame.

Note: Any attempt to dequeue more CAPTURE buffers beyond the buffer marked with V4L2_BUF_FLAG_LAST will result in a -EPIPE error from VIDIOC_DQBUF().

- dequeuing processed OUTPUT buffers, until all the buffers queued before the V4L2_DEC_CMD_STOP command are dequeued,
- dequeuing the V4L2_EVENT_EOS event, if the client subscribed to it.

Note: For backwards compatibility, the decoder will signal a V4L2_EVENT_EOS event when the last frame has been decoded and all frames are ready to be dequeued. It is a deprecated behavior and the client must not rely on it. The V4L2_BUF_FLAG_LAST buffer flag should be used instead.

3. Once all the OUTPUT buffers queued before the V4L2_DEC_CMD_STOP call are dequeued and the last CAPTURE buffer is dequeued, the decoder is stopped and it will accept, but not process, any newly queued OUTPUT buffers until the client issues any of the following operations:
 - V4L2_DEC_CMD_START - the decoder will not be reset and will resume operation normally, with all the state from before the drain,
 - a pair of VIDIOC_STREAMOFF() and VIDIOC_STREAMON() on the CAPTURE queue - the decoder will resume the operation normally, however any CAPTURE buffers still in the queue will be returned to the client,
 - a pair of VIDIOC_STREAMOFF() and VIDIOC_STREAMON() on the OUTPUT queue - any pending source buffers will be returned to the client and the *Seek* sequence will be triggered.

Note: Once the drain sequence is initiated, the client needs to drive it to completion, as

described by the steps above, unless it aborts the process by issuing `VIDIOC_STREAMOFF()` on any of the `OUTPUT` or `CAPTURE` queues. The client is not allowed to issue `V4L2_DEC_CMD_START` or `V4L2_DEC_CMD_STOP` again while the drain sequence is in progress and they will fail with `-EBUSY` error code if attempted.

Although not mandatory, the availability of decoder commands may be queried using `VIDIOC_TRY_DECODER_CMD()`.

End of Stream

If the decoder encounters an end of stream marking in the stream, the decoder will initiate the *Drain* sequence, which the client must handle as described above, skipping the initial `VIDIOC_DECODER_CMD()`.

Commit Points

Setting formats and allocating buffers trigger changes in the behavior of the decoder.

1. Setting the format on the `OUTPUT` queue may change the set of formats supported/advertised on the `CAPTURE` queue. In particular, it also means that the `CAPTURE` format may be reset and the client must not rely on the previously set format being preserved.
2. Enumerating formats on the `CAPTURE` queue always returns only formats supported for the current `OUTPUT` format.
3. Setting the format on the `CAPTURE` queue does not change the list of formats available on the `OUTPUT` queue. An attempt to set a `CAPTURE` format that is not supported for the currently selected `OUTPUT` format will result in the decoder adjusting the requested `CAPTURE` format to a supported one.
4. Enumerating formats on the `OUTPUT` queue always returns the full set of supported coded formats, irrespectively of the current `CAPTURE` format.
5. While buffers are allocated on any of the `OUTPUT` or `CAPTURE` queues, the client must not change the format on the `OUTPUT` queue. Drivers will return the `-EBUSY` error code for any such format change attempt.

To summarize, setting formats and allocation must always start with the `OUTPUT` queue and the `OUTPUT` queue is the master that governs the set of supported formats for the `CAPTURE` queue.

Memory-to-Memory Stateful Video Encoder Interface

A stateful video encoder takes raw video frames in display order and encodes them into a bytestream. It generates complete chunks of the bytestream, including all metadata, headers, etc. The resulting bytestream does not require any further post-processing by the client.

Performing software stream processing, header generation etc. in the driver in order to support this interface is strongly discouraged. In case such operations are needed, use of the Stateless Video Encoder Interface (in development) is strongly advised.

Querying Capabilities

1. To enumerate the set of coded formats supported by the encoder, the client may call `VIDIOC_ENUM_FMT()` on `CAPTURE`.
 - The full set of supported formats will be returned, regardless of the format set on `OUTPUT`.
2. To enumerate the set of supported raw formats, the client may call `VIDIOC_ENUM_FMT()` on `OUTPUT`.
 - Only the formats supported for the format currently active on `CAPTURE` will be returned.
 - In order to enumerate raw formats supported by a given coded format, the client must first set that coded format on `CAPTURE` and then enumerate the formats on `OUTPUT`.
3. The client may use `VIDIOC_ENUM_FRAMEIZES()` to detect supported resolutions for a given format, passing the desired pixel format in `v4l2_frmsizeenum pixel_format`.
 - Values returned by `VIDIOC_ENUM_FRAMEIZES()` for a coded pixel format will include all possible coded resolutions supported by the encoder for the given coded pixel format.
 - Values returned by `VIDIOC_ENUM_FRAMEIZES()` for a raw pixel format will include all possible frame buffer resolutions supported by the encoder for the given raw pixel format and coded format currently set on `CAPTURE`.
4. The client may use `VIDIOC_ENUM_FRAMEINTERVALS()` to detect supported frame intervals for a given format and resolution, passing the desired pixel format in `v4l2_frmsizeenum pixel_format` and the resolution in `v4l2_frmsizeenum width` and `v4l2_frmsizeenum height`.
 - Values returned by `VIDIOC_ENUM_FRAMEINTERVALS()` for a coded pixel format and coded resolution will include all possible frame intervals supported by the encoder for the given coded pixel format and resolution.
 - Values returned by `VIDIOC_ENUM_FRAMEINTERVALS()` for a raw pixel format and resolution will include all possible frame intervals supported by the encoder for the given raw pixel format and resolution and for the coded format, coded resolution and coded frame interval currently set on `CAPTURE`.
 - Support for `VIDIOC_ENUM_FRAMEINTERVALS()` is optional. If it is not implemented, then there are no special restrictions other than the limits of the codec itself.
5. Supported profiles and levels for the coded format currently set on `CAPTURE`, if applicable, may be queried using their respective controls via `VIDIOC_QUERYCTRL()`.
6. Any additional encoder capabilities may be discovered by querying their respective controls.

Initialization

1. Set the coded format on the CAPTURE queue via `VIDIOC_S_FMT()`.

- **Required fields:**

type a `V4L2_BUF_TYPE_*` enum appropriate for CAPTURE.

pixelformat the coded format to be produced.

sizeimage desired size of CAPTURE buffers; the encoder may adjust it to match hardware requirements.

width, height ignored (read-only).

other fields follow standard semantics.

- **Return fields:**

sizeimage adjusted size of CAPTURE buffers.

width, height the coded size selected by the encoder based on current state, e.g. OUTPUT format, selection rectangles, etc. (read-only).

Important: Changing the CAPTURE format may change the currently set OUTPUT format. How the new OUTPUT format is determined is up to the encoder and the client must ensure it matches its needs afterwards.

2. **Optional.** Enumerate supported OUTPUT formats (raw formats for source) for the selected coded format via `VIDIOC_ENUM_FMT()`.

- **Required fields:**

type a `V4L2_BUF_TYPE_*` enum appropriate for OUTPUT.

other fields follow standard semantics.

- **Return fields:**

pixelformat raw format supported for the coded format currently selected on the CAPTURE queue.

other fields follow standard semantics.

3. Set the raw source format on the OUTPUT queue via `VIDIOC_S_FMT()`.

- **Required fields:**

type a `V4L2_BUF_TYPE_*` enum appropriate for OUTPUT.

pixelformat raw format of the source.

width, height source resolution.

other fields follow standard semantics.

- **Return fields:**

width, height may be adjusted to match encoder minimums, maximums and alignment requirements, as required by the currently selected formats, as reported by `VIDIOC_ENUM_FRAMESIZES()`.

other fields follow standard semantics.

- Setting the OUTPUT format will reset the selection rectangles to their default values, based on the new resolution, as described in the next step.
4. Set the raw frame interval on the OUTPUT queue via `VIDIOC_S_PARM()`. This also sets the coded frame interval on the CAPTURE queue to the same value.
- **** Required fields:****
 - type** a `V4L2_BUF_TYPE_*` enum appropriate for OUTPUT.
 - parm.output** set all fields except `parm.output.timeperframe` to 0.
 - parm.output.timeperframe** the desired frame interval; the encoder may adjust it to match hardware requirements.
 - **Return fields:**
 - parm.output.timeperframe** the adjusted frame interval.

Important: Changing the OUTPUT frame interval *also* sets the framerate that the encoder uses to encode the video. So setting the frame interval to 1/24 (or 24 frames per second) will produce a coded video stream that can be played back at that speed. The frame interval for the OUTPUT queue is just a hint, the application may provide raw frames at a different rate. It can be used by the driver to help schedule multiple encoders running in parallel.

In the next step the CAPTURE frame interval can optionally be changed to a different value. This is useful for off-line encoding where the coded frame interval can be different from the rate at which raw frames are supplied.

Important: `timeperframe` deals with *frames*, not fields. So for interlaced formats this is the time per two fields, since a frame consists of a top and a bottom field.

Note: It is due to historical reasons that changing the OUTPUT frame interval also changes the coded frame interval on the CAPTURE queue. Ideally these would be independent settings, but that would break the existing API.

5. **Optional** Set the coded frame interval on the CAPTURE queue via `VIDIOC_S_PARM()`. This is only necessary if the coded frame interval is different from the raw frame interval, which is typically the case for off-line encoding. Support for this feature is signalled by the `V4L2_FMT_FLAG_ENC_CAP_FRAME_INTERVAL` format flag.
- **** Required fields:****
 - type** a `V4L2_BUF_TYPE_*` enum appropriate for CAPTURE.
 - parm.capture** set all fields except `parm.capture.timeperframe` to 0.
 - parm.capture.timeperframe** the desired coded frame interval; the encoder may adjust it to match hardware requirements.
 - **Return fields:**
 - parm.capture.timeperframe** the adjusted frame interval.

Important: Changing the CAPTURE frame interval sets the framerate for the coded video. It does *not* set the rate at which buffers arrive on the CAPTURE queue, that depends on how fast the encoder is and how fast raw frames are queued on the OUTPUT queue.

Important: `timeperframe` deals with *frames*, not fields. So for interlaced formats this is the time per two fields, since a frame consists of a top and a bottom field.

Note: Not all drivers support this functionality, in that case just set the desired coded frame interval for the OUTPUT queue.

However, drivers that can schedule multiple encoders based on the OUTPUT frame interval must support this optional feature.

6. **Optional.** Set the visible resolution for the stream metadata via `VIDIOC_S_SELECTION()` on the OUTPUT queue if it is desired to be different than the full OUTPUT resolution.

- **Required fields:**

type a `V4L2_BUF_TYPE_*` enum appropriate for OUTPUT.

target set to `V4L2_SEL_TGT_CROP`.

r.left, r.top, r.width, r.height visible rectangle; this must fit within the `V4L2_SEL_TGT_CROP_BOUNDS` rectangle and may be subject to adjustment to match codec and hardware constraints.

- **Return fields:**

r.left, r.top, r.width, r.height visible rectangle adjusted by the encoder.

- The following selection targets are supported on OUTPUT:

V4L2_SEL_TGT_CROP_BOUNDS equal to the full source frame, matching the active OUTPUT format.

V4L2_SEL_TGT_CROP_DEFAULT equal to `V4L2_SEL_TGT_CROP_BOUNDS`.

V4L2_SEL_TGT_CROP rectangle within the source buffer to be encoded into the CAPTURE stream; defaults to `V4L2_SEL_TGT_CROP_DEFAULT`.

Note: A common use case for this selection target is encoding a source video with a resolution that is not a multiple of a macroblock, e.g. the common 1920x1080 resolution may require the source buffers to be aligned to 1920x1088 for codecs with 16x16 macroblock size. To avoid encoding the padding, the client needs to explicitly configure this selection target to 1920x1080.

Warning: The encoder may adjust the crop/compose rectangles to the nearest supported ones to meet codec and hardware requirements. The client needs to check the adjusted rectangle returned by `VIDIOC_S_SELECTION()`.

7. Allocate buffers for both OUTPUT and CAPTURE via `VIDIOC_REQBUFS()`. This may be performed in any order.

- **Required fields:**

count requested number of buffers to allocate; greater than zero.

type a `V4L2_BUF_TYPE_*` enum appropriate for OUTPUT or CAPTURE.

other fields follow standard semantics.

- **Return fields:**

count actual number of buffers allocated.

Warning: The actual number of allocated buffers may differ from the count given. The client must check the updated value of count after the call returns.

Note: To allocate more than the minimum number of OUTPUT buffers (for pipeline depth), the client may query the `V4L2_CID_MIN_BUFFERS_FOR_OUTPUT` control to get the minimum number of buffers required, and pass the obtained value plus the number of additional buffers needed in the count field to `VIDIOC_REQBUFS()`.

Alternatively, `VIDIOC_CREATE_BUFS()` can be used to have more control over buffer allocation.

- **Required fields:**

count requested number of buffers to allocate; greater than zero.

type a `V4L2_BUF_TYPE_*` enum appropriate for OUTPUT.

other fields follow standard semantics.

- **Return fields:**

count adjusted to the number of allocated buffers.

8. Begin streaming on both OUTPUT and CAPTURE queues via `VIDIOC_STREAMON()`. This may be performed in any order. The actual encoding process starts when both queues start streaming.

Note: If the client stops the CAPTURE queue during the encode process and then restarts it again, the encoder will begin generating a stream independent from the stream generated before the stop. The exact constraints depend on the coded format, but may include the following implications:

- encoded frames produced after the restart must not reference any frames produced before the stop, e.g. no long term references for H.264/HEVC,
 - any headers that must be included in a standalone stream must be produced again, e.g. SPS and PPS for H.264/HEVC.
-

Encoding

This state is reached after the *Initialization* sequence finishes successfully. In this state, the client queues and dequeues buffers to both queues via `VIDIOC_QBUF()` and `VIDIOC_DQBUF()`, following the standard semantics.

The content of encoded CAPTURE buffers depends on the active coded pixel format and may be affected by codec-specific extended controls, as stated in the documentation of each format.

Both queues operate independently, following standard behavior of V4L2 buffer queues and memory-to-memory devices. In addition, the order of encoded frames dequeued from the CAPTURE queue may differ from the order of queuing raw frames to the OUTPUT queue, due to properties of the selected coded format, e.g. frame reordering.

The client must not assume any direct relationship between CAPTURE and OUTPUT buffers and any specific timing of buffers becoming available to dequeue. Specifically:

- a buffer queued to OUTPUT may result in more than one buffer produced on CAPTURE (for example, if returning an encoded frame allowed the encoder to return a frame that preceded it in display, but succeeded it in the decode order; however, there may be other reasons for this as well),
- a buffer queued to OUTPUT may result in a buffer being produced on CAPTURE later into encode process, and/or after processing further OUTPUT buffers, or be returned out of order, e.g. if display reordering is used,
- buffers may become available on the CAPTURE queue without additional buffers queued to OUTPUT (e.g. during drain or EOS), because of the OUTPUT buffers queued in the past whose encoding results are only available at later time, due to specifics of the encoding process,
- buffers queued to OUTPUT may not become available to dequeue instantly after being encoded into a corresponding CAPTURE buffer, e.g. if the encoder needs to use the frame as a reference for encoding further frames.

Note: To allow matching encoded CAPTURE buffers with OUTPUT buffers they originated from, the client can set the `timestamp` field of the `v4l2_buffer` struct when queuing an OUTPUT buffer. The CAPTURE buffer(s), which resulted from encoding that OUTPUT buffer will have their `timestamp` field set to the same value when dequeued.

In addition to the straightforward case of one OUTPUT buffer producing one CAPTURE buffer, the following cases are defined:

- one OUTPUT buffer generates multiple CAPTURE buffers: the same OUTPUT timestamp will be copied to multiple CAPTURE buffers,
- the encoding order differs from the presentation order (i.e. the CAPTURE buffers are out-of-order compared to the OUTPUT buffers): CAPTURE timestamps will not retain the order of OUTPUT timestamps.

Note: To let the client distinguish between frame types (keyframes, intermediate frames; the exact list of types depends on the coded format), the CAPTURE buffers will have corresponding flag bits set in their `v4l2_buffer` struct when dequeued. See the documentation of `v4l2_buffer` and each coded pixel format for exact list of flags and their meanings.

Should an encoding error occur, it will be reported to the client with the level of details depending on the encoder capabilities. Specifically:

- the CAPTURE buffer (if any) that contains the results of the failed encode operation will be returned with the `V4L2_BUF_FLAG_ERROR` flag set,
- if the encoder is able to precisely report the OUTPUT buffer(s) that triggered the error, such buffer(s) will be returned with the `V4L2_BUF_FLAG_ERROR` flag set.

Note: If a CAPTURE buffer is too small then it is just returned with the `V4L2_BUF_FLAG_ERROR` flag set. More work is needed to detect that this error occurred because the buffer was too small, and to provide support to free existing buffers that were too small.

In case of a fatal failure that does not allow the encoding to continue, any further operations on corresponding encoder file handle will return the `-EIO` error code. The client may close the file handle and open a new one, or alternatively reinitialize the instance by stopping streaming on both queues, releasing all buffers and performing the Initialization sequence again.

Encoding Parameter Changes

The client is allowed to use `VIDIOC_S_CTRL()` to change encoder parameters at any time. The availability of parameters is encoder-specific and the client must query the encoder to find the set of available controls.

The ability to change each parameter during encoding is encoder-specific, as per the standard semantics of the V4L2 control interface. The client may attempt to set a control during encoding and if the operation fails with the `-EBUSY` error code, the CAPTURE queue needs to be stopped for the configuration change to be allowed. To do this, it may follow the *Drain* sequence to avoid losing the already queued/encoded frames.

The timing of parameter updates is encoder-specific, as per the standard semantics of the V4L2 control interface. If the client needs to apply the parameters exactly at specific frame, using the Request API (*Request API*) should be considered, if supported by the encoder.

Drain

To ensure that all the queued OUTPUT buffers have been processed and the related CAPTURE buffers are given to the client, the client must follow the drain sequence described below. After the drain sequence ends, the client has received all encoded frames for all OUTPUT buffers queued before the sequence was started.

1. Begin the drain sequence by issuing `VIDIOC_ENCODER_CMD()`.

- **Required fields:**

- `cmd` set to `V4L2_ENC_CMD_STOP`.

- `flags` set to 0.

- `pts` set to 0.

Warning: The sequence can be only initiated if both OUTPUT and CAPTURE queues are streaming. For compatibility reasons, the call to `VIDIOC_ENCODER_CMD()` will not fail even if any of the queues is not streaming, but at the same time it will not initiate the *Drain* sequence and so the steps described below would not be applicable.

2. Any OUTPUT buffers queued by the client before the `VIDIOC_ENCODER_CMD()` was issued will be processed and encoded as normal. The client must continue to handle both queues independently, similarly to normal encode operation. This includes:
 - queuing and dequeuing CAPTURE buffers, until a buffer marked with the `V4L2_BUF_FLAG_LAST` flag is dequeued,

Warning: The last buffer may be empty (with `v4l2_buffer.bytesused = 0`) and in that case it must be ignored by the client, as it does not contain an encoded frame.

Note: Any attempt to dequeue more CAPTURE buffers beyond the buffer marked with `V4L2_BUF_FLAG_LAST` will result in a `-EPIPE` error from `VIDIOC_DQBUF()`.

- dequeuing processed OUTPUT buffers, until all the buffers queued before the `V4L2_ENC_CMD_STOP` command are dequeued,
- dequeuing the `V4L2_EVENT_EOS` event, if the client subscribes to it.

Note: For backwards compatibility, the encoder will signal a `V4L2_EVENT_EOS` event when the last frame has been encoded and all frames are ready to be dequeued. It is deprecated behavior and the client must not rely on it. The `V4L2_BUF_FLAG_LAST` buffer flag should be used instead.

3. Once all OUTPUT buffers queued before the `V4L2_ENC_CMD_STOP` call are dequeued and the last CAPTURE buffer is dequeued, the encoder is stopped and it will accept, but not process any newly queued OUTPUT buffers until the client issues any of the following operations:
 - `V4L2_ENC_CMD_START` - the encoder will not be reset and will resume operation normally, with all the state from before the drain,
 - a pair of `VIDIOC_STREAMOFF()` and `VIDIOC_STREAMON()` on the CAPTURE queue - the encoder will be reset (see the *Reset* sequence) and then resume encoding,
 - a pair of `VIDIOC_STREAMOFF()` and `VIDIOC_STREAMON()` on the OUTPUT queue - the encoder will resume operation normally, however any source frames queued to the OUTPUT queue between `V4L2_ENC_CMD_STOP` and `VIDIOC_STREAMOFF()` will be discarded.

Note: Once the drain sequence is initiated, the client needs to drive it to completion, as described by the steps above, unless it aborts the process by issuing `VIDIOC_STREAMOFF()` on any of the OUTPUT or CAPTURE queues. The client is not allowed to issue `V4L2_ENC_CMD_START` or `V4L2_ENC_CMD_STOP` again while the drain sequence is in progress and they will fail with `-EBUSY` error code if attempted.

For reference, handling of various corner cases is described below:

- In case of no buffer in the OUTPUT queue at the time the `V4L2_ENC_CMD_STOP` command was issued, the drain sequence completes immediately and the encoder returns an empty CAPTURE buffer with the `V4L2_BUF_FLAG_LAST` flag set.
- In case of no buffer in the CAPTURE queue at the time the drain sequence completes, the next time the client queues a CAPTURE buffer it is returned at once as an empty buffer with the `V4L2_BUF_FLAG_LAST` flag set.
- If `VIDIOC_STREAMOFF()` is called on the CAPTURE queue in the middle of the drain sequence, the drain sequence is canceled and all CAPTURE buffers are implicitly returned to the client.
- If `VIDIOC_STREAMOFF()` is called on the OUTPUT queue in the middle of the drain sequence, the drain sequence completes immediately and next CAPTURE buffer will be returned empty with the `V4L2_BUF_FLAG_LAST` flag set.

Although not mandatory, the availability of encoder commands may be queried using `VIDIOC_TRY_ENCODER_CMD()`.

Reset

The client may want to request the encoder to reinitialize the encoding, so that the following stream data becomes independent from the stream data generated before. Depending on the coded format, that may imply that:

- encoded frames produced after the restart must not reference any frames produced before the stop, e.g. no long term references for H.264/HEVC,
- any headers that must be included in a standalone stream must be produced again, e.g. SPS and PPS for H.264/HEVC.

This can be achieved by performing the reset sequence.

1. Perform the *Drain* sequence to ensure all the in-flight encoding finishes and respective buffers are dequeued.
2. Stop streaming on the CAPTURE queue via `VIDIOC_STREAMOFF()`. This will return all currently queued CAPTURE buffers to the client, without valid frame data.
3. Start streaming on the CAPTURE queue via `VIDIOC_STREAMON()` and continue with regular encoding sequence. The encoded frames produced into CAPTURE buffers from now on will contain a standalone stream that can be decoded without the need for frames encoded before the reset sequence, starting at the first OUTPUT buffer queued after issuing the `V4L2_ENC_CMD_STOP` of the *Drain* sequence.

This sequence may be also used to change encoding parameters for encoders without the ability to change the parameters on the fly.

Commit Points

Setting formats and allocating buffers triggers changes in the behavior of the encoder.

1. Setting the format on the CAPTURE queue may change the set of formats supported/advertised on the OUTPUT queue. In particular, it also means that the OUTPUT format may be reset and the client must not rely on the previously set format being preserved.
2. Enumerating formats on the OUTPUT queue always returns only formats supported for the current CAPTURE format.
3. Setting the format on the OUTPUT queue does not change the list of formats available on the CAPTURE queue. An attempt to set the OUTPUT format that is not supported for the currently selected CAPTURE format will result in the encoder adjusting the requested OUTPUT format to a supported one.
4. Enumerating formats on the CAPTURE queue always returns the full set of supported coded formats, irrespective of the current OUTPUT format.
5. While buffers are allocated on any of the OUTPUT or CAPTURE queues, the client must not change the format on the CAPTURE queue. Drivers will return the -EBUSY error code for any such format change attempt.

To summarize, setting formats and allocation must always start with the CAPTURE queue and the CAPTURE queue is the master that governs the set of supported formats for the OUTPUT queue.

Memory-to-memory Stateless Video Decoder Interface

A stateless decoder is a decoder that works without retaining any kind of state between processed frames. This means that each frame is decoded independently of any previous and future frames, and that the client is responsible for maintaining the decoding state and providing it to the decoder with each decoding request. This is in contrast to the stateful video decoder interface, where the hardware and driver maintain the decoding state and all the client has to do is to provide the raw encoded stream and dequeue decoded frames in display order.

This section describes how user-space (“the client”) is expected to communicate with stateless decoders in order to successfully decode an encoded stream. Compared to stateful codecs, the decoder/client sequence is simpler, but the cost of this simplicity is extra complexity in the client which is responsible for maintaining a consistent decoding state.

Stateless decoders make use of the *Request API*. A stateless decoder must expose the V4L2_BUF_CAP_SUPPORTS_REQUESTS capability on its OUTPUT queue when VIDIOC_REQBUFS() or VIDIOC_CREATE_BUFS() are invoked.

Depending on the encoded formats supported by the decoder, a single decoded frame may be the result of several decode requests (for instance, H.264 streams with multiple slices per frame). Decoders that support such formats must also expose the V4L2_BUF_CAP_SUPPORTS_M2M_HOLD_CAPTURE_BUF capability on their OUTPUT queue.

Querying capabilities

1. To enumerate the set of coded formats supported by the decoder, the client calls `VIDIOC_ENUM_FMT()` on the `OUTPUT` queue.
 - The driver must always return the full set of supported `OUTPUT` formats, irrespective of the format currently set on the `CAPTURE` queue.
 - Simultaneously, the driver must restrain the set of values returned by codec-specific capability controls (such as H.264 profiles) to the set actually supported by the hardware.
2. To enumerate the set of supported raw formats, the client calls `VIDIOC_ENUM_FMT()` on the `CAPTURE` queue.
 - The driver must return only the formats supported for the format currently active on the `OUTPUT` queue.
 - Depending on the currently set `OUTPUT` format, the set of supported raw formats may depend on the value of some codec-dependent controls. The client is responsible for making sure that these controls are set before querying the `CAPTURE` queue. Failure to do so will result in the default values for these controls being used, and a returned set of formats that may not be usable for the media the client is trying to decode.
3. The client may use `VIDIOC_ENUM_FRAMESIZES()` to detect supported resolutions for a given format, passing desired pixel format in `v4l2_frmsizeenum`'s `pixel_format`.
4. Supported profiles and levels for the current `OUTPUT` format, if applicable, may be queried using their respective controls via `VIDIOC_QUERYCTRL()`.

Initialization

1. Set the coded format on the `OUTPUT` queue via `VIDIOC_S_FMT()`.
 - **Required fields:**
 - type** a `V4L2_BUF_TYPE_*` enum appropriate for `OUTPUT`.
 - pixelformat** a coded pixel format.
 - width, height** coded width and height parsed from the stream.
 - other fields** follow standard semantics.

Note: Changing the `OUTPUT` format may change the currently set `CAPTURE` format. The driver will derive a new `CAPTURE` format from the `OUTPUT` format being set, including resolution, colorimetry parameters, etc. If the client needs a specific `CAPTURE` format, it must adjust it afterwards.

2. Call `VIDIOC_S_EXT_CTRL()` to set all the controls (parsed headers, etc.) required by the `OUTPUT` format to enumerate the `CAPTURE` formats.
3. Call `VIDIOC_G_FMT()` for `CAPTURE` queue to get the format for the destination buffers parsed/decoded from the bytestream.
 - **Required fields:**

type a V4L2_BUF_TYPE_* enum appropriate for CAPTURE.

- **Returned fields:**

width, height frame buffer resolution for the decoded frames.

pixelformat pixel format for decoded frames.

num_planes (for _MPLANE type only) number of planes for pixelformat.

sizeimage, bytesperline as per standard semantics; matching frame buffer format.

Note: The value of pixelformat may be any pixel format supported for the OUTPUT format, based on the hardware capabilities. It is suggested that the driver chooses the preferred/optimal format for the current configuration. For example, a YUV format may be preferred over an RGB format, if an additional conversion step would be required for RGB.

4. *[optional]* Enumerate CAPTURE formats via VIDIOC_ENUM_FMT() on the CAPTURE queue. The client may use this ioctl to discover which alternative raw formats are supported for the current OUTPUT format and select one of them via VIDIOC_S_FMT().

Note: The driver will return only formats supported for the currently selected OUTPUT format and currently set controls, even if more formats may be supported by the decoder in general.

For example, a decoder may support YUV and RGB formats for resolutions 1920x1088 and lower, but only YUV for higher resolutions (due to hardware limitations). After setting a resolution of 1920x1088 or lower as the OUTPUT format, VIDIOC_ENUM_FMT() may return a set of YUV and RGB pixel formats, but after setting a resolution higher than 1920x1088, the driver will not return RGB pixel formats, since they are unsupported for this resolution.

5. *[optional]* Choose a different CAPTURE format than suggested via VIDIOC_S_FMT() on CAPTURE queue. It is possible for the client to choose a different format than selected/suggested by the driver in VIDIOC_G_FMT().

- **Required fields:**

type a V4L2_BUF_TYPE_* enum appropriate for CAPTURE.

pixelformat a raw pixel format.

width, height frame buffer resolution of the decoded stream; typically unchanged from what was returned with VIDIOC_G_FMT(), but it may be different if the hardware supports composition and/or scaling.

After performing this step, the client must perform step 3 again in order to obtain up-to-date information about the buffers size and layout.

6. Allocate source (bytestream) buffers via VIDIOC_REQBUFS() on OUTPUT queue.

- **Required fields:**

count requested number of buffers to allocate; greater than zero.

type a V4L2_BUF_TYPE_* enum appropriate for OUTPUT.

memory follows standard semantics.

- **Return fields:**

count actual number of buffers allocated.

- If required, the driver will adjust count to be equal or bigger to the minimum of required number of OUTPUT buffers for the given format and requested count. The client must check this value after the ioctl returns to get the actual number of buffers allocated.

7. Allocate destination (raw format) buffers via `VIDIOC_REQBUFS()` on the CAPTURE queue.

- **Required fields:**

count requested number of buffers to allocate; greater than zero. The client is responsible for deducing the minimum number of buffers required for the stream to be properly decoded (taking e.g. reference frames into account) and pass an equal or bigger number.

type a `V4L2_BUF_TYPE_*` enum appropriate for CAPTURE.

memory follows standard semantics. `V4L2_MEMORY_USERPTR` is not supported for CAPTURE buffers.

- **Return fields:**

count adjusted to allocated number of buffers, in case the codec requires more buffers than requested.

- The driver must adjust count to the minimum of required number of CAPTURE buffers for the current format, stream configuration and requested count. The client must check this value after the ioctl returns to get the number of buffers allocated.

8. **Allocate requests (likely one per OUTPUT buffer) via `MEDIA_IOC_REQUEST_ALLOC()`** on the media device.

9. **Start streaming on both OUTPUT and CAPTURE queues via `VIDIOC_STREAMON()`.**

Decoding

For each frame, the client is responsible for submitting at least one request to which the following is attached:

- The amount of encoded data expected by the codec for its current configuration, as a buffer submitted to the OUTPUT queue. Typically, this corresponds to one frame worth of encoded data, but some formats may allow (or require) different amounts per unit.
- All the metadata needed to decode the submitted encoded data, in the form of controls relevant to the format being decoded.

The amount of data and contents of the source OUTPUT buffer, as well as the controls that must be set on the request, depend on the active coded pixel format and might be affected by codec-specific extended controls, as stated in documentation of each format.

If there is a possibility that the decoded frame will require one or more decode requests after the current one in order to be produced, then the client must set the `V4L2_BUF_FLAG_M2M_HOLD_CAPTURE_BUF` flag on the OUTPUT buffer. This will result in the (potentially partially) decoded CAPTURE buffer not being made available for dequeuing, and reused for the next decode request if the timestamp of the next OUTPUT buffer has not changed.

A typical frame would thus be decoded using the following sequence:

1. Queue an OUTPUT buffer containing one unit of encoded bytestream data for the decoding request, using `VIDIOC_QBUF()`.

- **Required fields:**

index index of the buffer being queued.

type type of the buffer.

bytesused number of bytes taken by the encoded data frame in the buffer.

flags the `V4L2_BUF_FLAG_REQUEST_FD` flag must be set. Additionally, if we are not sure that the current decode request is the last one needed to produce a fully decoded frame, then `V4L2_BUF_FLAG_M2M_HOLD_CAPTURE_BUF` must also be set.

request_fd must be set to the file descriptor of the decoding request.

timestamp must be set to a unique value per frame. This value will be propagated into the decoded frame's buffer and can also be used to use this frame as the reference of another. If using multiple decode requests per frame, then the timestamps of all the OUTPUT buffers for a given frame must be identical. If the timestamp changes, then the currently held CAPTURE buffer will be made available for dequeuing and the current request will work on a new CAPTURE buffer.

2. Set the codec-specific controls for the decoding request, using `VIDIOC_S_EXT_CTRL()`.

- **Required fields:**

which must be `V4L2_CTRL_WHICH_REQUEST_VAL`.

request_fd must be set to the file descriptor of the decoding request.

other fields other fields are set as usual when setting controls. The controls array must contain all the codec-specific controls required to decode a frame.

Note: It is possible to specify the controls in different invocations of `VIDIOC_S_EXT_CTRL()`, or to overwrite a previously set control, as long as `request_fd` and `which` are properly set. The controls state at the moment of request submission is the one that will be considered.

Note: The order in which steps 1 and 2 take place is interchangeable.

3. Submit the request by invoking `MEDIA_REQUEST_IOC_QUEUE()` on the request FD.

If the request is submitted without an OUTPUT buffer, or if some of the required controls are missing from the request, then `MEDIA_REQUEST_IOC_QUEUE()` will return `-ENOENT`. If more than one OUTPUT buffer is queued, then it will return `-EINVAL`. `MEDIA_REQUEST_IOC_QUEUE()` returning non-zero means that no CAPTURE buffer will be produced for this request.

CAPTURE buffers must not be part of the request, and are queued independently. They are returned in decode order (i.e. the same order as coded frames were submitted to the OUTPUT queue).

Runtime decoding errors are signaled by the dequeued CAPTURE buffers carrying the `V4L2_BUF_FLAG_ERROR` flag. If a decoded reference frame has an error, then all following decoded frames that refer to it also have the `V4L2_BUF_FLAG_ERROR` flag set, although the decoder will still try to produce (likely corrupted) frames.

Buffer management while decoding

Contrary to stateful decoders, a stateless decoder does not perform any kind of buffer management: it only guarantees that dequeued CAPTURE buffers can be used by the client for as long as they are not queued again. “Used” here encompasses using the buffer for compositing or display.

A dequeued capture buffer can also be used as the reference frame of another buffer.

A frame is specified as reference by converting its timestamp into nanoseconds, and storing it into the relevant member of a codec-dependent control structure. The `v4l2_timeval_to_ns()` function must be used to perform that conversion. The timestamp of a frame can be used to reference it as soon as all its units of encoded data are successfully submitted to the OUTPUT queue.

A decoded buffer containing a reference frame must not be reused as a decoding target until all the frames referencing it have been decoded. The safest way to achieve this is to refrain from queueing a reference buffer until all the decoded frames referencing it have been dequeued. However, if the driver can guarantee that buffers queued to the CAPTURE queue are processed in queued order, then user-space can take advantage of this guarantee and queue a reference buffer when the following conditions are met:

1. All the requests for frames affected by the reference frame have been queued, and
2. A sufficient number of CAPTURE buffers to cover all the decoded referencing frames have been queued.

When queueing a decoding request, the driver will increase the reference count of all the resources associated with reference frames. This means that the client can e.g. close the DMABUF file descriptors of reference frame buffers if it won't need them afterwards.

Seeking

In order to seek, the client just needs to submit requests using input buffers corresponding to the new stream position. It must however be aware that resolution may have changed and follow the dynamic resolution change sequence in that case. Also depending on the codec used, picture parameters (e.g. SPS/PPS for H.264) may have changed and the client is responsible for making sure that a valid state is sent to the decoder.

The client is then free to ignore any returned CAPTURE buffer that comes from the pre-seek position.

Pausing

In order to pause, the client can just cease queuing buffers onto the OUTPUT queue. Without source bytestream data, there is no data to process and the codec will remain idle.

Dynamic resolution change

If the client detects a resolution change in the stream, it will need to perform the initialization sequence again with the new resolution:

1. If the last submitted request resulted in a CAPTURE buffer being held by the use of the `V4L2_BUF_FLAG_M2M_HOLD_CAPTURE_BUF` flag, then the last frame is not available on the CAPTURE queue. In this case, a `V4L2_DEC_CMD_FLUSH` command shall be sent. This will make the driver dequeue the held CAPTURE buffer.
2. Wait until all submitted requests have completed and dequeue the corresponding output buffers.
3. Call `VIDIOC_STREAMOFF()` on both the OUTPUT and CAPTURE queues.
4. Free all CAPTURE buffers by calling `VIDIOC_REQBUFS()` on the CAPTURE queue with a buffer count of zero.
5. Perform the initialization sequence again (minus the allocation of OUTPUT buffers), with the new resolution set on the OUTPUT queue. Note that due to resolution constraints, a different format may need to be picked on the CAPTURE queue.

Drain

If the last submitted request resulted in a CAPTURE buffer being held by the use of the `V4L2_BUF_FLAG_M2M_HOLD_CAPTURE_BUF` flag, then the last frame is not available on the CAPTURE queue. In this case, a `V4L2_DEC_CMD_FLUSH` command shall be sent. This will make the driver dequeue the held CAPTURE buffer.

After that, in order to drain the stream on a stateless decoder, the client just needs to wait until all the submitted requests are completed.

3.2.4.6 Raw VBI Data Interface

VBI is an abbreviation of Vertical Blanking Interval, a gap in the sequence of lines of an analog video signal. During VBI no picture information is transmitted, allowing some time while the electron beam of a cathode ray tube TV returns to the top of the screen. Using an oscilloscope you will find here the vertical synchronization pulses and short data packages ASK modulated¹ onto the video signal. These are transmissions of services such as Teletext or Closed Caption.

Subject of this interface type is raw VBI data, as sampled off a video signal, or to be added to a signal for output. The data format is similar to uncompressed video images, a number of lines times a number of samples per line, we call this a VBI image.

Conventionally V4L2 VBI devices are accessed through character device special files named `/dev/vbi` and `/dev/vbi0` to `/dev/vbi31` with major number 81 and minor numbers 224 to 255.

¹ ASK: Amplitude-Shift Keying. A high signal level represents a '1' bit, a low level a '0' bit.

`/dev/vbi` is typically a symbolic link to the preferred VBI device. This convention applies to both input and output devices.

To address the problems of finding related video and VBI devices VBI capturing and output is also available as device function under `/dev/video`. To capture or output raw VBI data with these devices applications must call the `VIDIOC_S_FMT` ioctl. Accessed as `/dev/vbi`, raw VBI capturing or output is the default device function.

Querying Capabilities

Devices supporting the raw VBI capturing or output API set the `V4L2_CAP_VBI_CAPTURE` or `V4L2_CAP_VBI_OUTPUT` flags, respectively, in the `capabilities` field of struct `v4l2_capability` returned by the `ioctl VIDIOC_QUERYCAP` ioctl. At least one of the read/write, streaming or asynchronous I/O methods must be supported. VBI devices may or may not have a tuner or modulator.

Supplemental Functions

VBI devices shall support *video input or output*, *tuner or modulator*, and *controls* ioctls as needed. The *video standard* ioctls provide information vital to program a VBI device, therefore must be supported.

Raw VBI Format Negotiation

Raw VBI sampling abilities can vary, in particular the sampling frequency. To properly interpret the data V4L2 specifies an ioctl to query the sampling parameters. Moreover, to allow for some flexibility applications can also suggest different parameters.

As usual these parameters are *not* reset at `open()` time to permit Unix tool chains, programming a device and then reading from it as if it was a plain file. Well written V4L2 applications should always ensure they really get what they want, requesting reasonable parameters and then checking if the actual parameters are suitable.

To query the current raw VBI capture parameters applications set the `type` field of a struct `v4l2_format` to `V4L2_BUF_TYPE_VBI_CAPTURE` or `V4L2_BUF_TYPE_VBI_OUTPUT`, and call the `VIDIOC_G_FMT` ioctl with a pointer to this structure. Drivers fill the struct `v4l2_vbi_format` `vbi` member of the `fmt` union.

To request different parameters applications set the `type` field of a struct `v4l2_format` as above and initialize all fields of the struct `v4l2_vbi_format` `vbi` member of the `fmt` union, or better just modify the results of `VIDIOC_G_FMT`, and call the `VIDIOC_S_FMT` ioctl with a pointer to this structure. Drivers return an `EINVAL` error code only when the given parameters are ambiguous, otherwise they modify the parameters according to the hardware capabilities and return the actual parameters. When the driver allocates resources at this point, it may return an `EBUSY` error code to indicate the returned parameters are valid but the required resources are currently not available. That may happen for instance when the video and VBI areas to capture would overlap, or when the driver supports multiple opens and another process already requested VBI capturing or output. Anyway, applications must expect other resource allocation points which may return `EBUSY`, at the `ioctl VIDIOC_STREAMON`, `VIDIOC_STREAMOFF` ioctl and the first `read()`, `write()` and `select()` calls.

VBI devices must implement both the `VIDIOC_G_FMT` and `VIDIOC_S_FMT` ioctl, even if `VIDIOC_S_FMT` ignores all requests and always returns default parameters as `VIDIOC_G_FMT` does. `VIDIOC_TRY_FMT` is optional.

v4l2_vbi_format

Table 106: struct v4l2_vbi_format

__u32	sampling_rate	Samples per second, i. e. unit 1 Hz.
__u32	offset	Horizontal offset of the VBI image, relative to the leading edge of the line synchronization pulse and counted in samples: The first sample in the VBI image will be located offset / sampling_rate seconds following the leading edge. See also <i>Figure 4.1. Line synchronization</i> .
__u32	samples_per_line	
__u32	sample_format	Defines the sample format as in <i>Image Formats</i> , a four-character-code. ² Usually this is <code>V4L2_PIX_FMT_GREY</code> , i. e. each sample consists of 8 bits with lower values oriented towards the black level. Do not assume any other correlation of values with the signal level. For example, the MSB does not necessarily indicate if the signal is 'high' or 'low' because 128 may not be the mean value of the signal. Drivers shall not convert the sample format by software.
__u32	start ²	This is the scanning system line number associated with the first line of the VBI image, of the first and the second field respectively. See <i>Figure 4.2. ITU-R 525 line numbering (M/NTSC and M/PAL)</i> and <i>Figure 4.3. ITU-R 625 line numbering</i> for valid values. The <code>V4L2_VBI_ITU_525_F1_START</code> , <code>V4L2_VBI_ITU_525_F2_START</code> , <code>V4L2_VBI_ITU_625_F1_START</code> and <code>V4L2_VBI_ITU_625_F2_START</code> defines give the start line numbers for each field for each 525 or 625 line format as a convenience. Don't forget that ITU line numbering starts at 1, not 0. VBI input drivers can return start values 0 if the hardware cannot reliably identify scanning lines, VBI acquisition may not require this information.
__u32	count ²	The number of lines in the first and second field image, respectively.
<p>Drivers should be as flexibility as possible. For example, it may be possible to extend or move the VBI capture window down to the picture area, implementing a 'full field mode' to capture data service transmissions embedded in the picture.</p> <p>An application can set the first or second count value to zero if no data is required from the respective field; count[1] if the scanning system is progressive, i. e. not interlaced. The corresponding start value shall be ignored by the application and driver. Anyway, drivers may not support single field capturing and return both count values non-zero.</p> <p>Both count values set to zero, or line numbers are outside the bounds depicted⁴, or a field image covering lines of two fields, are invalid and shall not be returned by the driver.</p> <p>To initialize the start and count fields, applications must first determine the current video standard selection. The <code>v4l2_std_id</code> or the <code>framelines</code> field of struct <code>v4l2_standard</code> can be evaluated for this purpose.</p>		
__u32	flags	See <i>Raw VBI Format Flags</i> below. Currently only drivers set flags, applications must set this field to zero.

Continued on next page

Table 106 - continued from previous page

<code>__u32</code>	reserved ²	This array is reserved for future extensions. Drivers and applications must set it to zero.
--------------------	-----------------------	---

Table 107: Raw VBI Format Flags

<code>V4L2_VBI_UNSYNC</code>	0x0001	This flag indicates hardware which does not properly distinguish between fields. Normally the VBI image stores the first field (lower scanning line numbers) first in memory. This may be a top or bottom field depending on the video standard. When this flag is set the first or second field may be stored first, however the fields are still in correct temporal order with the older field first in memory. ³
<code>V4L2_VBI_INTERLACED</code>	0x0002	By default the two field images will be passed sequentially; all lines of the first field followed by all lines of the second field (compare <i>Field Order</i> <code>V4L2_FIELD_SEQ_TB</code> and <code>V4L2_FIELD_SEQ_BT</code> , whether the top or bottom field is first in memory depends on the video standard). When this flag is set, the two fields are interlaced (cf. <code>V4L2_FIELD_INTERLACED</code>). The first line of the first field followed by the first line of the second field, then the two second lines, and so on. Such a layout may be necessary when the hardware has been programmed to capture or output interlaced video images and is unable to separate the fields for VBI capturing at the same time. For simplicity setting this flag implies that both count values are equal and non-zero.

Remember the VBI image format depends on the selected video standard, therefore the application must choose a new standard or query the current standard first. Attempts to read or write data ahead of format negotiation, or after switching the video standard which may invalidate the negotiated VBI parameters, should be refused by the driver. A format change during active I/O is not permitted.

Reading and writing VBI images

To assure synchronization with the field number and easier implementation, the smallest unit of data passed at a time is one frame, consisting of two fields of VBI images immediately following in memory.

The total size of a frame computes as follows:

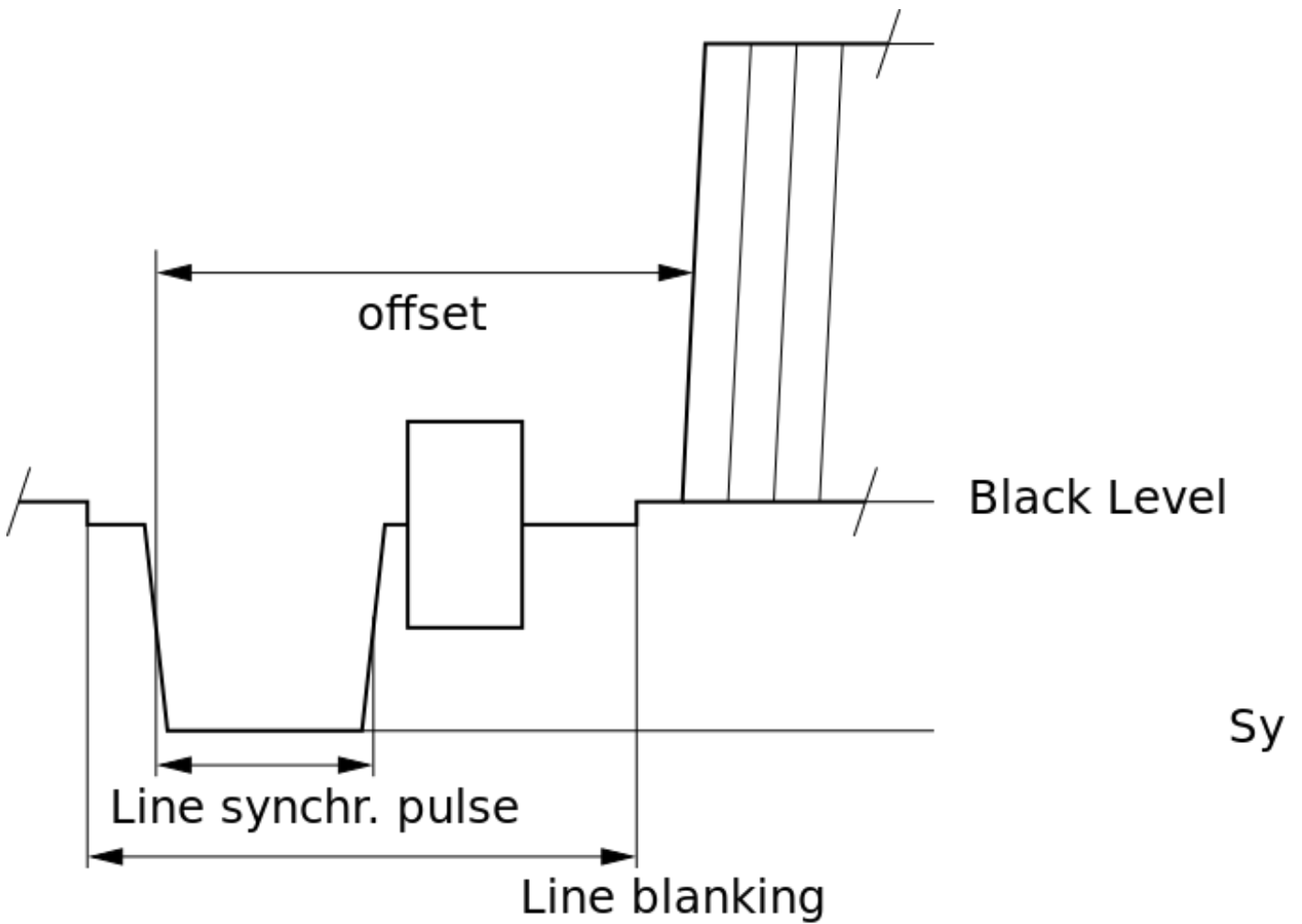
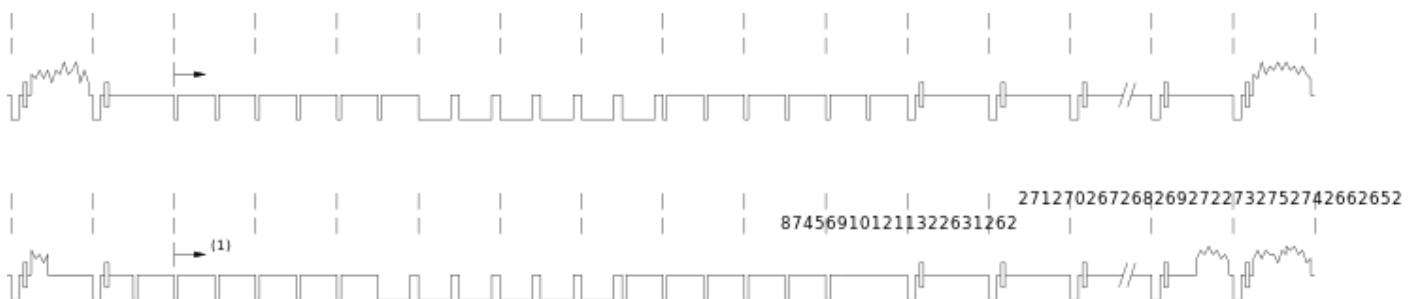
$$(\text{count}[0] + \text{count}[1]) * \text{samples_per_line} * \text{sample size in bytes}$$

The sample size is most likely always one byte, applications must check the `sample_format` field though, to function properly with other drivers.

² A few devices may be unable to sample VBI data at all but can extend the video capture window to the VBI region.

⁴ The valid values are shown at *Figure 4.2. ITU-R 525 line numbering (M/NTSC and M/PAL)* and *Figure 4.3. ITU-R 625 line numbering*.

³ Most VBI services transmit on both fields, but some have different semantics depending on the field number. These cannot be reliably decoded or encoded when `V4L2_VBI_UNSYNC` is set.

Fig. 10: **Figure 4.1. Line synchronization**Fig. 11: **Figure 4.2. ITU-R 525 line numbering (M/NTSC and M/PAL)**

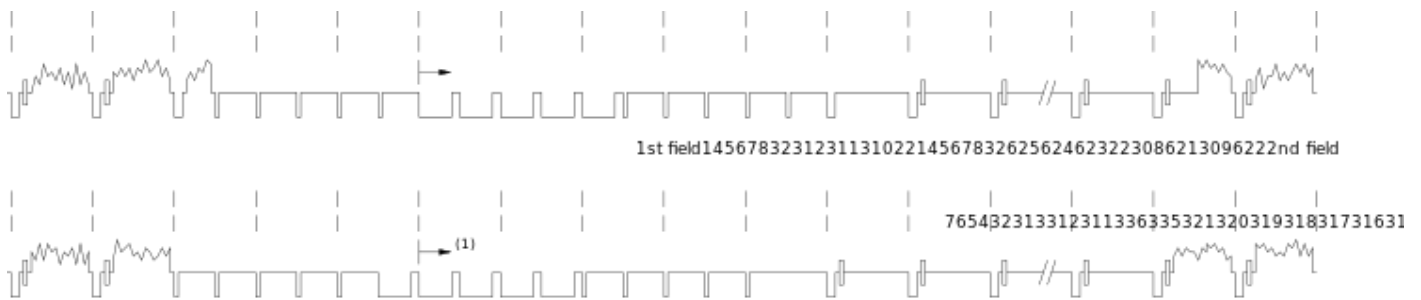


Fig. 12: **Figure 4.3. ITU-R 625 line numbering**

A VBI device may support *read/write* and/or streaming (*memory mapping* or *user pointer*) I/O. The latter bears the possibility of synchronizing video and VBI data by using buffer timestamps. Remember the `VIDIOC_STREAMON` ioctl and the first `read()`, `write()` and `select()` call can be resource allocation points returning an EBUSY error code if the required hardware resources are temporarily unavailable, for example the device is already in use by another process.

3.2.4.7 Sliced VBI Data Interface

VBI stands for Vertical Blanking Interval, a gap in the sequence of lines of an analog video signal. During VBI no picture information is transmitted, allowing some time while the electron beam of a cathode ray tube TV returns to the top of the screen.

Sliced VBI devices use hardware to demodulate data transmitted in the VBI. V4L2 drivers shall *not* do this by software, see also the *raw VBI interface*. The data is passed as short packets of fixed size, covering one scan line each. The number of packets per video frame is variable.

Sliced VBI capture and output devices are accessed through the same character special files as raw VBI devices. When a driver supports both interfaces, the default function of a `/dev/vbi` device is *raw* VBI capturing or output, and the sliced VBI function is only available after calling the `VIDIOC_S_FMT` ioctl as defined below. Likewise a `/dev/video` device may support the sliced VBI API, however the default function here is video capturing or output. Different file descriptors must be used to pass raw and sliced VBI data simultaneously, if this is supported by the driver.

Querying Capabilities

Devices supporting the sliced VBI capturing or output API set the `V4L2_CAP_SLICED_VBI_CAPTURE` or `V4L2_CAP_SLICED_VBI_OUTPUT` flag respectively, in the `capabilities` field of struct `v4l2_capability` returned by the *ioctl* `VIDIOC_QUERYCAP` ioctl. At least one of the *read/write*, *streaming* or *asynchronous I/O methods* must be supported. Sliced VBI devices may have a tuner or modulator.

Supplemental Functions

Sliced VBI devices shall support *video input or output* and *tuner or modulator* ioctls if they have these capabilities, and they may support *User Controls* ioctls. The *video standard* ioctls provide information vital to program a sliced VBI device, therefore must be supported.

Sliced VBI Format Negotiation

To find out which data services are supported by the hardware applications can call the *VIDIOC_G_SLICED_VBI_CAP* ioctl. All drivers implementing the sliced VBI interface must support this ioctl. The results may differ from those of the *VIDIOC_S_FMT* ioctl when the number of VBI lines the hardware can capture or output per frame, or the number of services it can identify on a given line are limited. For example on PAL line 16 the hardware may be able to look for a VPS or Teletext signal, but not both at the same time.

To determine the currently selected services applications set the type field of struct `v4l2_format` to `V4L2_BUF_TYPE_SLICED_VBI_CAPTURE` or `V4L2_BUF_TYPE_SLICED_VBI_OUTPUT`, and the *VIDIOC_G_FMT* ioctl fills the `fmt.sliced` member, a struct `v4l2_sliced_vbi_format`.

Applications can request different parameters by initializing or modifying the `fmt.sliced` member and calling the *VIDIOC_S_FMT* ioctl with a pointer to the struct `v4l2_format` structure.

The sliced VBI API is more complicated than the raw VBI API because the hardware must be told which VBI service to expect on each scan line. Not all services may be supported by the hardware on all lines (this is especially true for VBI output where Teletext is often unsupported and other services can only be inserted in one specific line). In many cases, however, it is sufficient to just set the `service_set` field to the required services and let the driver fill the `service_lines` array according to hardware capabilities. Only if more precise control is needed should the programmer set the `service_lines` array explicitly.

The *VIDIOC_S_FMT* ioctl modifies the parameters according to hardware capabilities. When the driver allocates resources at this point, it may return an `EBUSY` error code if the required resources are temporarily unavailable. Other resource allocation points which may return `EBUSY` can be the *ioctl VIDIOC_STREAMON*, *VIDIOC_STREAMOFF* ioctl and the first `read()`, `write()` and `select()` call.

`v4l2_sliced_vbi_format`

`struct v4l2_sliced_vbi_format`

__u16	<code>service_set</code>	If <code>service_set</code> is non-zero when passed with <i>VIDIOC_S_FMT</i> or <i>VIDIOC_TRY_FMT</i> , the <code>service_lines</code> array will be filled by the driver according to the services specified in this field. For example, if <code>service_set</code> is initialized with <code>V4L2_SLICED_TELETEXT_B V4L2_SLICED_WSS_625</code> , a driver for the <code>cx25840</code> video decoder sets lines 7-22 of both fields ¹ to <code>V4L2_SLICED_TELETEXT_B</code> and line 23 of the first field to <code>V4L2_SLICED_WSS_625</code> . If <code>service_set</code> is set to zero, then the values of <code>service_lines</code> will be used instead. On return the driver sets this field to the union of all elements of the returned <code>service_lines</code> array. It may contain less services than requested, perhaps just one, if the hardware cannot handle more services simultaneously. It may be empty (zero) if none of the requested services are supported by the hardware.
-------	--------------------------	--

Continued on next page

Table 108 - continued from previous page

__u16	service_lines[2][24]	Applications initialize this array with sets of data services the driver shall look for or insert on the respective scan line. Subject to hardware capabilities drivers return the requested set, a subset, which may be just a single service, or an empty set. When the hardware cannot handle multiple services on the same line the driver shall choose one. No assumptions can be made on which service the driver chooses. Data services are defined in <i>Sliced VBI services</i> . Array indices map to ITU-R line numbers ² as follows:		
	Element	525 line systems	625 line systems	
	service_lines[0][1]	1	1	
	service_lines[0][23]	23	23	
	service_lines[1][1]	264	314	
	service_lines[1][23]	286	336	
		Drivers must set service_lines [0][0] and service_lines[1][0] to zero. The V4L2_VBI_ITU_525_F1_START, V4L2_VBI_ITU_525_F2_START, V4L2_VBI_ITU_625_F1_START and V4L2_VBI_ITU_625_F2_START defines give the start line numbers for each field for each 525 or 625 line format as a convenience. Don't forget that ITU line numbering starts at 1, not 0.		
__u32	io_size	Maximum number of bytes passed by one read() or write() call, and the buffer size in bytes for the <i>ioctl VIDIOC_QBUF</i> , <i>VIDIOC_DQBUF</i> and <i>VIDIOC_DQBUF</i> ioctl. Drivers set this field to the size of struct v4l2_sliced_vbi_data times the number of non-zero elements in the returned service_lines array (that is the number of lines potentially carrying data).		
__u32	reserved[2]	This array is reserved for future extensions. Applications and drivers must set it to zero.		

Sliced VBI services

Symbol	Value	Reference	Lines, usually	Payload
V4L2_SLICED_TELETEXT_B (Teletext System B)	0x0001	<i>ETS 300 706</i> , <i>ITU BT.653</i>	PAL/SECAM line 7-22, 320-335 (second field 7-22)	Last 42 of the 45 byte Teletext packet, that is without clock run-in and framing code, lsb first transmitted.
V4L2_SLICED_VPS	0x0400	<i>ETS 300 231</i>	PAL line 16	Byte number 3 to 15 according to Figure 9 of ETS 300 231, lsb first transmitted.
V4L2_SLICED_CAPTION_525	0x1000	<i>CEA 608-E</i>	NTSC line 21, 284 (second field 21)	Two bytes in transmission order, including parity bit, lsb first transmitted.
V4L2_SLICED_WSS_625	0x4000	<i>ITU BT.1119</i> , <i>EN 300 294</i>	PAL/SECAM line 23	See <i>V4L2_SLICED_WSS_625 payload</i> below.
V4L2_SLICED_VBI_525	0x1000	Set of services applicable to 525 line systems.		
V4L2_SLICED_VBI_625	0x4401	Set of services applicable to 625 line systems.		

Drivers may return an `EINVAL` error code when applications attempt to read or write data without prior format negotiation, after switching the video standard (which may invalidate the negotiated VBI parameters) and after switching the video input (which may change the video standard as a side effect). The `VIDIOC_S_FMT` ioctl may return an `EBUSY` error code when applications attempt to change the format while i/o is in progress (between a `ioctl VIDIOC_STREAMON`, `VIDIOC_STREAMOFF` and `VIDIOC_STREAMOFF` call, and after the first `read()` or `write()` call).

¹ According to *ETS 300 706* lines 6-22 of the first field and lines 5-22 of the second field may carry Teletext data.

² See also *Figure 4.2. ITU-R 525 line numbering (M/NTSC and M/PAL)* and *Figure 4.3. ITU-R 625 line numbering*.

V4L2_SLICED_WSS_625 payload

The payload for V4L2_SLICED_WSS_625 is:

Byte	0								1							
Bit	msb				lsb				msb				lsb			
	7	6	5	4	3	2	1	0	x	x	13	12	11	10	9	8

Reading and writing sliced VBI data

A single `read()` or `write()` call must pass all data belonging to one video frame. That is an array of `struct v4l2_sliced_vbi_data` structures with one or more elements and a total size not exceeding `io_size` bytes. Likewise in streaming I/O mode one buffer of `io_size` bytes must contain data of one video frame. The `id` of unused `struct v4l2_sliced_vbi_data` elements must be zero.

v4l2_sliced_vbi_data

struct v4l2_sliced_vbi_data

__u32	id	A flag from <i>Sliced VBI services</i> identifying the type of data in this packet. Only a single bit must be set. When the <code>id</code> of a captured packet is zero, the packet is empty and the contents of other fields are undefined. Applications shall ignore empty packets. When the <code>id</code> of a packet for output is zero the contents of the <code>data</code> field are undefined and the driver must no longer insert data on the requested field and line.
__u32	field	The video field number this data has been captured from, or shall be inserted at. 0 for the first field, 1 for the second field.
__u32	line	The field (as opposed to frame) line number this data has been captured from, or shall be inserted at. See <i>Figure 4.2. ITU-R 525 line numbering (M/NTSC and M/PAL)</i> and <i>Figure 4.3. ITU-R 625 line numbering</i> for valid values. Sliced VBI capture devices can set the line number of all packets to 0 if the hardware cannot reliably identify scan lines. The field number must always be valid.
__u32	reserved	This field is reserved for future extensions. Applications and drivers must set it to zero.
__u8	data[48]	The packet payload. See <i>Sliced VBI services</i> for the contents and number of bytes passed for each data type. The contents of padding bytes at the end of this array are undefined, drivers and applications shall ignore them.

Packets are always passed in ascending line number order, without duplicate line numbers. The `write()` function and the `ioctl VIDIOC_QBUF`, `VIDIOC_DQBUF` `ioctl` must return an `EINVAL` error code when applications violate this rule. They must also return an `EINVAL` error code when applications pass an incorrect field or line number, or a combination of field, line and `id` which has not been negotiated with the `VIDIOC_G_FMT` or `VIDIOC_S_FMT` `ioctl`. When the line numbers are unknown the driver must pass the packets in transmitted order. The driver can insert empty packets with `id` set to zero anywhere in the packet array.

To assure synchronization and to distinguish from frame dropping, when a captured frame does not carry any of the requested data services drivers must pass one or more empty packets. When an application fails to pass VBI data in time for output, the driver must output the last VPS and WSS packet again, and disable the output of Closed Caption and Teletext data, or output data which is ignored by Closed Caption and Teletext decoders.

A sliced VBI device may support *read/write* and/or streaming (*memory mapping* and/or *user pointer*) I/O. The latter bears the possibility of synchronizing video and VBI data by using buffer timestamps.

Sliced VBI Data in MPEG Streams

If a device can produce an MPEG output stream, it may be capable of providing *negotiated sliced VBI services* as data embedded in the MPEG stream. Users or applications control this sliced VBI data insertion with the `V4L2_CID_MPEG_STREAM_VBI_FMT` control.

If the driver does not provide the `V4L2_CID_MPEG_STREAM_VBI_FMT` control, or only allows that control to be set to `V4L2_MPEG_STREAM_VBI_FMT_NONE`, then the device cannot embed sliced VBI data in the MPEG stream.

The `V4L2_CID_MPEG_STREAM_VBI_FMT` control does not implicitly set the device driver to capture nor cease capturing sliced VBI data. The control only indicates to embed sliced VBI data in the MPEG stream, if an application has negotiated sliced VBI service be captured.

It may also be the case that a device can embed sliced VBI data in only certain types of MPEG streams: for example in an MPEG-2 PS but not an MPEG-2 TS. In this situation, if sliced VBI data insertion is requested, the sliced VBI data will be embedded in MPEG stream types when supported, and silently omitted from MPEG stream types where sliced VBI data insertion is not supported by the device.

The following subsections specify the format of the embedded sliced VBI data.

MPEG Stream Embedded, Sliced VBI Data Format: NONE

The `V4L2_MPEG_STREAM_VBI_FMT_NONE` embedded sliced VBI format shall be interpreted by drivers as a control to cease embedding sliced VBI data in MPEG streams. Neither the device nor driver shall insert “empty” embedded sliced VBI data packets in the MPEG stream when this format is set. No MPEG stream data structures are specified for this format.

MPEG Stream Embedded, Sliced VBI Data Format: IVTV

The `V4L2_MPEG_STREAM_VBI_FMT_IVTV` embedded sliced VBI format, when supported, indicates to the driver to embed up to 36 lines of sliced VBI data per frame in an MPEG-2 *Private Stream 1 PES* packet encapsulated in an MPEG-2 *Program Pack* in the MPEG stream.

Historical context: This format specification originates from a custom, embedded, sliced VBI data format used by the `ivtv` driver. This format has already been informally specified in the kernel sources in the file `Documentation/userspace-api/media/drivers/cx2341x-uapi.rst`. The maximum size of the payload and other aspects of this format are driven by the CX23415 MPEG decoder’s capabilities and limitations with respect to extracting, decoding, and displaying sliced VBI data embedded within an MPEG stream.

This format's use is *not* exclusive to the `ivtv` driver *nor* exclusive to CX2341x devices, as the sliced VBI data packet insertion into the MPEG stream is implemented in driver software. At least the `cx18` driver provides sliced VBI data insertion into an MPEG-2 PS in this format as well.

The following definitions specify the payload of the MPEG-2 *Private Stream 1 PES* packets that contain sliced VBI data when `V4L2_MPEG_STREAM_VBI_FMT_IVTV` is set. (The MPEG-2 *Private Stream 1 PES* packet header and encapsulating MPEG-2 *Program Pack* header are not detailed here. Please refer to the MPEG-2 specifications for details on those packet headers.)

The payload of the MPEG-2 *Private Stream 1 PES* packets that contain sliced VBI data is specified by struct `v4l2_mpeg_vbi_fmt_ivtv`. The payload is variable length, depending on the actual number of lines of sliced VBI data present in a video frame. The payload may be padded at the end with unspecified fill bytes to align the end of the payload to a 4-byte boundary. The payload shall never exceed 1552 bytes (2 fields with 18 lines/field with 43 bytes of data/line and a 4 byte magic number).

`v4l2_mpeg_vbi_fmt_ivtv`

struct v4l2_mpeg_vbi_fmt_ivtv

<code>__u8</code>	<code>magic[4]</code>	A "magic" constant from <i>Magic Constants for struct v4l2_mpeg_vbi_fmt_ivtv magic field</i> that indicates this is a valid sliced VBI data payload and also indicates which member of the anonymous union, <code>itv0</code> or <code>ITV0</code> , to use for the payload data.
<code>union {</code>	(anonymous)	
<code>struct v4l2_mpeg_vbi_itv0</code>	<code>itv0</code>	The primary form of the sliced VBI data payload that contains anywhere from 1 to 35 lines of sliced VBI data. Line masks are provided in this form of the payload indicating which VBI lines are provided.
<code>struct v4l2_mpeg_vbi_ITV0</code>	<code>ITV0</code>	An alternate form of the sliced VBI data payload used when 36 lines of sliced VBI data are present. No line masks are provided in this form of the payload; all valid line mask bits are implicitly set.
<code>}</code>		

Magic Constants for struct v4l2_mpeg_vbi_fmt_ivtv magic field

Defined Symbol	Value	Description
<code>V4L2_MPEG_VBI_IVTV_MAGIC0</code>	"itv0"	Indicates the <code>itv0</code> member of the union in struct <code>v4l2_mpeg_vbi_fmt_ivtv</code> is valid.
<code>V4L2_MPEG_VBI_IVTV_MAGIC1</code>	"ITV0"	Indicates the <code>ITV0</code> member of the union in struct <code>v4l2_mpeg_vbi_fmt_ivtv</code> is valid and that 36 lines of sliced VBI data are present.

`v4l2_mpeg_vbi_itv0`

v4l2_mpeg_vbi_ITV0

structs v4l2_mpeg_vbi_itv0 and v4l2_mpeg_vbi_ITV0

__le32	linemask[2]	<p>Bitmasks indicating the VBI service lines present. These linemask values are stored in little endian byte order in the MPEG stream. Some reference linemask bit positions with their corresponding VBI line number and video field are given below. b_0 indicates the least significant bit of a linemask value:</p> <pre> linemask[0] b0: line 6 first field linemask[0] b17: line 23 first field linemask[0] b18: line 6 second field linemask[0] b31: line 19 second field linemask[1] b0: line 20 second field linemask[1] b3: line 23 second field linemask[1] b4-b31: unused and set to 0 </pre>
struct v4l2_mpeg_vbi_itv0_line	line[35]	<p>This is a variable length array that holds from 1 to 35 lines of sliced VBI data. The sliced VBI data lines present correspond to the bits set in the linemask array, starting from b_0 of linemask[0] up through b_{31} of linemask[0], and from b_0 of linemask[1] up through b_3 of linemask[1]. line[0] corresponds to the first bit found set in the linemask array, line[1] corresponds to the second bit found set in the linemask array, etc. If no linemask array bits are set, then line[0] may contain one line of unspecified data that should be ignored by applications.</p>

struct v4l2_mpeg_vbi_ITV0

struct v4l2_mpeg_vbi_itv0_line	line[36]	<p>A fixed length array of 36 lines of sliced VBI data. line[0] through line[17] correspond to lines 6 through 23 of the first field. line[18] through line[35] corresponds to lines 6 through 23 of the second field.</p>
--------------------------------	----------	--

v4l2_mpeg_vbi_itv0_line

struct v4l2_mpeg_vbi_itv0_line

__u8	id	<p>A line identifier value from <i>Line Identifiers for struct v4l2_mpeg_vbi_itv0_line id field</i> that indicates the type of sliced VBI data stored on this line.</p>
__u8	data[42]	<p>The sliced VBI data for the line.</p>

Line Identifiers for struct `v4l2_mpeg_vbi_itv0_line` id field

Defined Symbol	Value	Description
<code>V4L2_MPEG_VBI_IVTV_TELETEXT_B</code>	1	Refer to <i>Sliced VBI services</i> for a description of the line payload.
<code>V4L2_MPEG_VBI_IVTV_CAPTION_525</code>	4	Refer to <i>Sliced VBI services</i> for a description of the line payload.
<code>V4L2_MPEG_VBI_IVTV_WSS_625</code>	5	Refer to <i>Sliced VBI services</i> for a description of the line payload.
<code>V4L2_MPEG_VBI_IVTV_VPS</code>	7	Refer to <i>Sliced VBI services</i> for a description of the line payload.

3.2.4.8 Radio Interface

This interface is intended for AM and FM (analog) radio receivers and transmitters.

Conventionally V4L2 radio devices are accessed through character device special files named `/dev/radio` and `/dev/radio0` to `/dev/radio63` with major number 81 and minor numbers 64 to 127.

Querying Capabilities

Devices supporting the radio interface set the `V4L2_CAP_RADIO` and `V4L2_CAP_TUNER` or `V4L2_CAP_MODULATOR` flag in the `capabilities` field of struct `v4l2_capability` returned by the `ioctl VIDIOC_QUERYCAP` ioctl. Other combinations of capability flags are reserved for future extensions.

Supplemental Functions

Radio devices can support *controls*, and must support the *tuner or modulator* ioctls.

They do not support the video input or output, audio input or output, video standard, cropping and scaling, compression and streaming parameter, or overlay ioctls. All other ioctls and I/O methods are reserved for future extensions.

Programming

Radio devices may have a couple audio controls (as discussed in *User Controls*) such as a volume control, possibly custom controls. Further all radio devices have one tuner or modulator (these are discussed in *Tuners and Modulators*) with index number zero to select the radio frequency and to determine if a monaural or FM stereo program is received/emitted. Drivers switch automatically between AM and FM depending on the selected frequency. The `VIDIOC_G_TUNER` or `VIDIOC_G_MODULATOR` ioctl reports the supported frequency range.

3.2.4.9 RDS Interface

The Radio Data System transmits supplementary information in binary format, for example the station name or travel information, on an inaudible audio subcarrier of a radio program. This interface is aimed at devices capable of receiving and/or transmitting RDS information.

For more information see the core RDS standard *IEC 62106* and the RBDS standard *NRSC-4-B*.

Note: Note that the RBDS standard as is used in the USA is almost identical to the RDS standard. Any RDS decoder/encoder can also handle RBDS. Only some of the fields have slightly different meanings. See the RBDS standard for more information.

The RBDS standard also specifies support for MMBS (Modified Mobile Search). This is a proprietary format which seems to be discontinued. The RDS interface does not support this format. Should support for MMBS (or the so-called 'E blocks' in general) be needed, then please contact the linux-media mailing list: <https://linuxtv.org/lists.php>.

Querying Capabilities

Devices supporting the RDS capturing API set the `V4L2_CAP_RDS_CAPTURE` flag in the capabilities field of struct `v4l2_capability` returned by the *ioctl* `VIDIOC_QUERYCAP` *ioctl*. Any tuner that supports RDS will set the `V4L2_TUNER_CAP_RDS` flag in the capability field of struct `v4l2_tuner`. If the driver only passes RDS blocks without interpreting the data the `V4L2_TUNER_CAP_RDS_BLOCK_IO` flag has to be set, see *Reading RDS data*. For future use the flag `V4L2_TUNER_CAP_RDS_CONTROLS` has also been defined. However, a driver for a radio tuner with this capability does not yet exist, so if you are planning to write such a driver you should discuss this on the linux-media mailing list: <https://linuxtv.org/lists.php>.

Whether an RDS signal is present can be detected by looking at the `rxsubchans` field of struct `v4l2_tuner`: the `V4L2_TUNER_SUB_RDS` will be set if RDS data was detected.

Devices supporting the RDS output API set the `V4L2_CAP_RDS_OUTPUT` flag in the capabilities field of struct `v4l2_capability` returned by the *ioctl* `VIDIOC_QUERYCAP` *ioctl*. Any modulator that supports RDS will set the `V4L2_TUNER_CAP_RDS` flag in the capability field of struct `v4l2_modulator`. In order to enable the RDS transmission one must set the `V4L2_TUNER_SUB_RDS` bit in the `txsubchans` field of struct `v4l2_modulator`. If the driver only passes RDS blocks without interpreting the data the `V4L2_TUNER_CAP_RDS_BLOCK_IO` flag has to be set. If the tuner is capable of handling RDS entities like program identification codes and radio text, the flag `V4L2_TUNER_CAP_RDS_CONTROLS` should be set, see *Writing RDS data* and *FM Transmitter Control Reference*.

Reading RDS data

RDS data can be read from the radio device with the `read()` function. The data is packed in groups of three bytes.

Writing RDS data

RDS data can be written to the radio device with the `write()` function. The data is packed in groups of three bytes, as follows:

RDS datastructures

v4l2_rds_data

Table 109: struct v4l2_rds_data

__u8	lsb	Least Significant Byte of RDS Block
__u8	msb	Most Significant Byte of RDS Block
__u8	block	Block description

Table 110: Block description

Bits 0-2	Block (aka offset) of the received data.
Bits 3-5	Deprecated. Currently identical to bits 0-2. Do not use these bits.
Bit 6	Corrected bit. Indicates that an error was corrected for this data block.
Bit 7	Error bit. Indicates that an uncorrectable error occurred during reception of this block.

Table 111: Block defines

V4L2_RDS_BLOCK_MSK		7	Mask for bits 0-2 to get the block ID.
V4L2_RDS_BLOCK_A		0	Block A.
V4L2_RDS_BLOCK_B		1	Block B.
V4L2_RDS_BLOCK_C		2	Block C.
V4L2_RDS_BLOCK_D		3	Block D.
V4L2_RDS_BLOCK_C_ALT		4	Block C'.
V4L2_RDS_BLOCK_INVALID	read-only	7	An invalid block.
V4L2_RDS_BLOCK_CORRECTED	read-only	0x40	A bit error was detected but corrected.
V4L2_RDS_BLOCK_ERROR	read-only	0x80	An uncorrectable error occurred.

3.2.4.10 Software Defined Radio Interface (SDR)

SDR is an abbreviation of Software Defined Radio, the radio device which uses application software for modulation or demodulation. This interface is intended for controlling and data streaming of such devices.

SDR devices are accessed through character device special files named `/dev/swradio0` to `/dev/swradio255` with major number 81 and dynamically allocated minor numbers 0 to 255.

Querying Capabilities

Devices supporting the SDR receiver interface set the `V4L2_CAP_SDR_CAPTURE` and `V4L2_CAP_TUNER` flag in the `capabilities` field of struct `v4l2_capability` returned by the `ioctl VIDIOC_QUERYCAP` ioctl. That flag means the device has an Analog to Digital Converter (ADC), which is a mandatory element for the SDR receiver.

Devices supporting the SDR transmitter interface set the `V4L2_CAP_SDR_OUTPUT` and `V4L2_CAP_MODULATOR` flag in the `capabilities` field of struct `v4l2_capability` returned by the `ioctl VIDIOC_QUERYCAP` ioctl. That flag means the device has an Digital to Analog Converter (DAC), which is a mandatory element for the SDR transmitter.

At least one of the read/write, streaming or asynchronous I/O methods must be supported.

Supplemental Functions

SDR devices can support *controls*, and must support the *Tuners and Modulators* ioctls. Tuner ioctls are used for setting the ADC/DAC sampling rate (sampling frequency) and the possible radio frequency (RF).

The `V4L2_TUNER_SDR` tuner type is used for setting SDR device ADC/DAC frequency, and the `V4L2_TUNER_RF` tuner type is used for setting radio frequency. The tuner index of the RF tuner (if any) must always follow the SDR tuner index. Normally the SDR tuner is #0 and the RF tuner is #1.

The `ioctl VIDIOC_S_HW_FREQ_SEEK` ioctl is not supported.

Data Format Negotiation

The SDR device uses the *Data Formats* ioctls to select the capture and output format. Both the sampling resolution and the data streaming format are bound to that selectable format. In addition to the basic *Data Formats* ioctls, the `ioctl VIDIOC_ENUM_FMT` ioctl must be supported as well.

To use the *Data Formats* ioctls applications set the type field of a struct `v4l2_format` to `V4L2_BUF_TYPE_SDR_CAPTURE` or `V4L2_BUF_TYPE_SDR_OUTPUT` and use the struct `v4l2_sdr_format` `sdr` member of the `fmt` union as needed per the desired operation. Currently there are two fields, `pixelformat` and `buffersize`, of struct `v4l2_sdr_format` which are used. Content of the `pixelformat` is V4L2 FourCC code of the data format. The `buffersize` field is maximum buffer size in bytes required for data transfer, set by the driver in order to inform application.

`v4l2_sdr_format`

Table 112: struct v4l2_sdr_format

__u32	pixelformat	The data format or type of compression, set by the application. This is a little endian <i>four character code</i> . V4L2 defines SDR formats in <i>SDR Formats</i> .
__u32	buffersize	Maximum size in bytes required for data. Value is set by the driver.
__u8	reserved[24]	This array is reserved for future extensions. Drivers and applications must set it to zero.

An SDR device may support *read/write* and/or streaming (*memory mapping* or *user pointer*) I/O.

3.2.4.11 Touch Devices

Touch devices are accessed through character device special files named `/dev/v4l-touch0` to `/dev/v4l-touch255` with major number 81 and dynamically allocated minor numbers 0 to 255.

Overview

Sensors may be Optical, or Projected Capacitive touch (PCT).

Processing is required to analyse the raw data and produce input events. In some systems, this may be performed on the ASIC and the raw data is purely a side-channel for diagnostics or tuning. In other systems, the ASIC is a simple analogue front end device which delivers touch data at high rate, and any touch processing must be done on the host.

For capacitive touch sensing, the touchscreen is composed of an array of horizontal and vertical conductors (alternatively called rows/columns, X/Y lines, or tx/rx). Mutual Capacitance measured is at the nodes where the conductors cross. Alternatively, Self Capacitance measures the signal from each column and row independently.

A touch input may be determined by comparing the raw capacitance measurement to a no-touch reference (or “baseline”) measurement:

$$\text{Delta} = \text{Raw} - \text{Reference}$$

The reference measurement takes account of variations in the capacitance across the touch sensor matrix, for example manufacturing irregularities, environmental or edge effects.

Querying Capabilities

Devices supporting the touch interface set the `V4L2_CAP_VIDEO_CAPTURE` flag and the `V4L2_CAP_TOUCH` flag in the capabilities field of `v4l2_capability` returned by the *ioctl VIDIOC_QUERYCAP* ioctl.

At least one of the *read/write* or streaming I/O methods must be supported.

The formats supported by touch devices are documented in *Touch Formats*.

Data Format Negotiation

A touch device may support any I/O method.

3.2.4.12 Event Interface

The V4L2 event interface provides a means for a user to get immediately notified on certain conditions taking place on a device. This might include start of frame or loss of signal events, for example. Changes in the value or state of a V4L2 control can also be reported through events.

To receive events, the events the user is interested in first must be subscribed using the *ioctl VIDIOC_SUBSCRIBE_EVENT, VIDIOC_UNSUBSCRIBE_EVENT* ioctl. Once an event is subscribed, the events of subscribed types are dequeuable using the *ioctl VIDIOC_DQEVENT* ioctl. Events may be unsubscribed using *VIDIOC_UNSUBSCRIBE_EVENT* ioctl. The special event type *V4L2_EVENT_ALL* may be used to unsubscribe all the events the driver supports.

The event subscriptions and event queues are specific to file handles. Subscribing an event on one file handle does not affect other file handles.

The information on dequeuable events is obtained by using *select* or *poll* system calls on video devices. The V4L2 events use *POLLPRI* events on *poll* system call and exceptions on *select* system call.

Starting with kernel 3.1 certain guarantees can be given with regards to events:

1. Each subscribed event has its own internal dedicated event queue. This means that flooding of one event type will not interfere with other event types.
2. If the internal event queue for a particular subscribed event becomes full, then the oldest event in that queue will be dropped.
3. Where applicable, certain event types can ensure that the payload of the oldest event that is about to be dropped will be merged with the payload of the next oldest event. Thus ensuring that no information is lost, but only an intermediate step leading up to that information. See the documentation for the event you want to subscribe to whether this is applicable for that event or not.

3.2.4.13 Sub-device Interface

The complex nature of V4L2 devices, where hardware is often made of several integrated circuits that need to interact with each other in a controlled way, leads to complex V4L2 drivers. The drivers usually reflect the hardware model in software, and model the different hardware components as software blocks called sub-devices.

V4L2 sub-devices are usually kernel-only objects. If the V4L2 driver implements the media device API, they will automatically inherit from media entities. Applications will be able to enumerate the sub-devices and discover the hardware topology using the media entities, pads and links enumeration API.

In addition to make sub-devices discoverable, drivers can also choose to make them directly configurable by applications. When both the sub-device driver and the V4L2 device driver support this, sub-devices will feature a character device node on which ioctls can be called to

- query, read and write sub-devices controls

- subscribe and unsubscribe to events and retrieve them
- negotiate image formats on individual pads

Sub-device character device nodes, conventionally named `/dev/v4l-subdev*`, use major number 81.

Drivers may opt to limit the sub-device character devices to only expose operations that do not modify the device state. In such a case the sub-devices are referred to as *read-only* in the rest of this documentation, and the related restrictions are documented in individual ioctls.

Controls

Most V4L2 controls are implemented by sub-device hardware. Drivers usually merge all controls and expose them through video device nodes. Applications can control all sub-devices through a single interface.

Complex devices sometimes implement the same control in different pieces of hardware. This situation is common in embedded platforms, where both sensors and image processing hardware implement identical functions, such as contrast adjustment, white balance or faulty pixels correction. As the V4L2 controls API doesn't support several identical controls in a single device, all but one of the identical controls are hidden.

Applications can access those hidden controls through the sub-device node with the V4L2 control API described in *User Controls*. The ioctls behave identically as when issued on V4L2 device nodes, with the exception that they deal only with controls implemented in the sub-device.

Depending on the driver, those controls might also be exposed through one (or several) V4L2 device nodes.

Events

V4L2 sub-devices can notify applications of events as described in *Event Interface*. The API behaves identically as when used on V4L2 device nodes, with the exception that it only deals with events generated by the sub-device. Depending on the driver, those events might also be reported on one (or several) V4L2 device nodes.

Pad-level Formats

Warning: Pad-level formats are only applicable to very complex devices that need to expose low-level format configuration to user space. Generic V4L2 applications do *not* need to use the API described in this section.

Note: For the purpose of this section, the term *format* means the combination of media bus data format, frame width and frame height.

Image formats are typically negotiated on video capture and output devices using the format and *selection* ioctls. The driver is responsible for configuring every block in the video pipeline according to the requested format at the pipeline input and/or output.

For complex devices, such as often found in embedded systems, identical image sizes at the output of a pipeline can be achieved using different hardware configurations. One such example is shown on *Image Format Negotiation on Pipelines*, where image scaling can be performed on both the video sensor and the host image processing hardware.

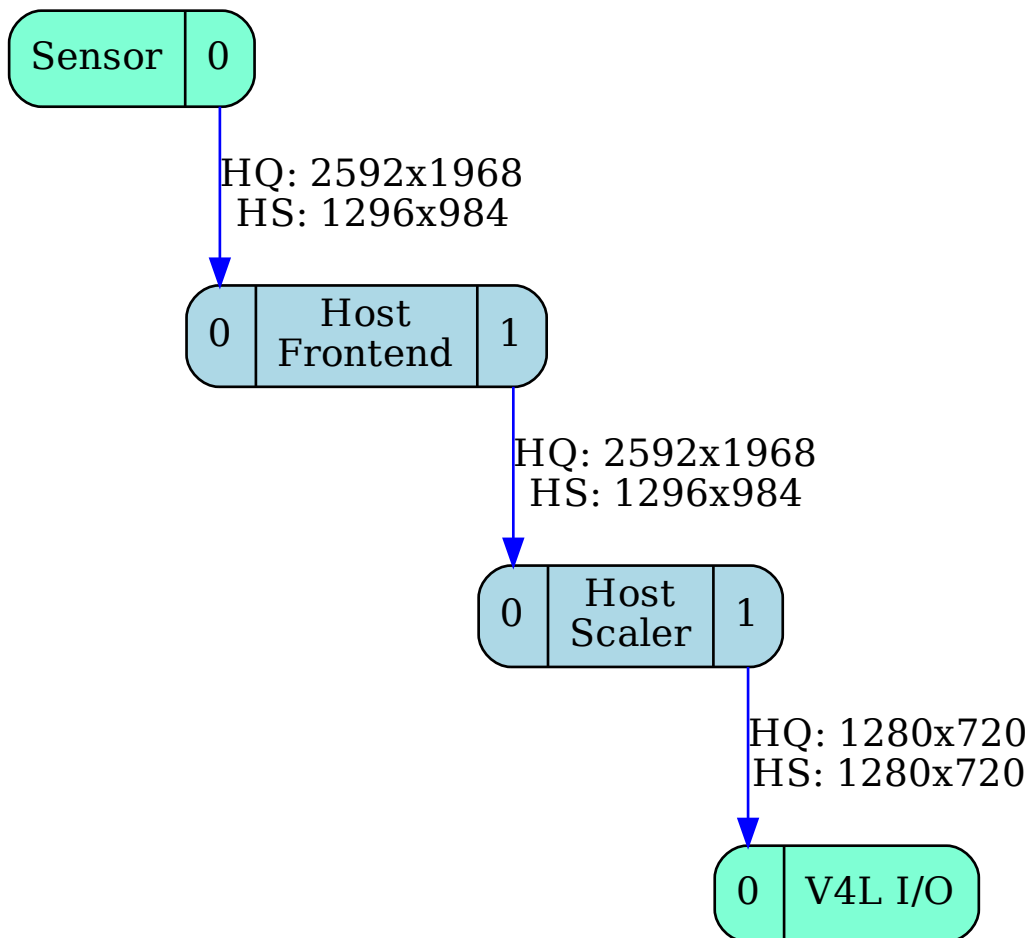


Fig. 13: Image Format Negotiation on Pipelines
High quality and high speed pipeline configuration

The sensor scaler is usually of less quality than the host scaler, but scaling on the sensor is required to achieve higher frame rates. Depending on the use case (quality vs. speed), the pipeline must be configured differently. Applications need to configure the formats at every point in the pipeline explicitly.

Drivers that implement the *media API* can expose pad-level image format configuration to applications. When they do, applications can use the `VIDIOC_SUBDEV_G_FMT` and `VIDIOC_SUBDEV_S_FMT` ioctls. to negotiate formats on a per-pad basis.

Applications are responsible for configuring coherent parameters on the whole pipeline and making sure that connected pads have compatible formats. The pipeline is checked for format mismatch at `VIDIOC_STREAMON` time, and an `EPIPE` error code is then returned if the configuration is invalid.

Pad-level image format configuration support can be tested by calling the `ioctl VIDIOC_SUBDEV_G_FMT, VIDIOC_SUBDEV_S_FMT` ioctl on pad 0. If the driver returns an `EINVAL` error code pad-level format configuration is not supported by the sub-device.

Format Negotiation

Acceptable formats on pads can (and usually do) depend on a number of external parameters, such as formats on other pads, active links, or even controls. Finding a combination of formats on all pads in a video pipeline, acceptable to both application and driver, can't rely on format enumeration only. A format negotiation mechanism is required.

Central to the format negotiation mechanism are the get/set format operations. When called with the `which` argument set to `V4L2_SUBDEV_FORMAT_TRY`, the `VIDIOC_SUBDEV_G_FMT` and `VIDIOC_SUBDEV_S_FMT` ioctls operate on a set of format parameters that are not connected to the hardware configuration. Modifying those 'try' formats leaves the device state untouched (this applies to both the software state stored in the driver and the hardware state stored in the device itself).

While not kept as part of the device state, try formats are stored in the sub-device file handles. A `VIDIOC_SUBDEV_G_FMT` call will return the last try format set *on the same sub-device file handle*. Several applications querying the same sub-device at the same time will thus not interact with each other.

To find out whether a particular format is supported by the device, applications use the `VIDIOC_SUBDEV_S_FMT` ioctl. Drivers verify and, if needed, change the requested format based on device requirements and return the possibly modified value. Applications can then choose to try a different format or accept the returned value and continue.

Formats returned by the driver during a negotiation iteration are guaranteed to be supported by the device. In particular, drivers guarantee that a returned format will not be further changed if passed to an `VIDIOC_SUBDEV_S_FMT` call as-is (as long as external parameters, such as formats on other pads or links' configuration are not changed).

Drivers automatically propagate formats inside sub-devices. When a try or active format is set on a pad, corresponding formats on other pads of the same sub-device can be modified by the driver. Drivers are free to modify formats as required by the device. However, they should comply with the following rules when possible:

- Formats should be propagated from sink pads to source pads. Modifying a format on a source pad should not modify the format on any sink pad.
- Sub-devices that scale frames using variable scaling factors should reset the scale factors to default values when sink pads formats are modified. If the 1:1 scaling ratio is supported, this means that source pads formats should be reset to the sink pads formats.

Formats are not propagated across links, as that would involve propagating them from one sub-device file handle to another. Applications must then take care to configure both ends of every link explicitly with compatible formats. Identical formats on the two ends of a link are guaranteed to be compatible. Drivers are free to accept different formats matching device requirements as being compatible.

Sample Pipeline Configuration shows a sample configuration sequence for the pipeline described in *Image Format Negotiation on Pipelines* (table columns list entity names and pad numbers).

Table 113: Sample Pipeline Configuration

	Sensor/0 format	Frontend/0 format	Frontend/1 format	Scaler/0 format	Scaler/0 compose selection rectangle	Scaler/1 format
Initial state	2048x1536 SGRBG8_1X8	(default)	(default)	(default)	(default)	(default)
Configure frontend sink format	2048x1536 SGRBG8_1X8	2048x1536 SGRBG8_1X8	2046x1534 SGRBG8_1X8	(default)	(default)	(default)
Configure scaler sink format	2048x1536 SGRBG8_1X8	2048x1536 SGRBG8_1X8	2046x1534 SGRBG8_1X8	2046x1534 SGRBG8_1X8	0,0/2046x1534	2046x1534 SGRBG8_1X8
Configure scaler sink compose selection	2048x1536 SGRBG8_1X8	2048x1536 SGRBG8_1X8	2046x1534 SGRBG8_1X8	2046x1534 SGRBG8_1X8	0,0/1280x960	1280x960 SGRBG8_1X8

1. Initial state. The sensor source pad format is set to its native 3MP size and V4L2_MBUS_FMT_SGRBG8_1X8 media bus code. Formats on the host frontend and scaler sink and source pads have the default values, as well as the compose rectangle on the scaler's sink pad.
2. The application configures the frontend sink pad format's size to 2048x1536 and its media bus code to V4L2_MBUS_FMT_SGRBG_1X8. The driver propagates the format to the frontend source pad.
3. The application configures the scaler sink pad format's size to 2046x1534 and the media bus code to V4L2_MBUS_FMT_SGRBG_1X8 to match the frontend source size and media bus code. The media bus code on the sink pad is set to V4L2_MBUS_FMT_SGRBG_1X8. The driver propagates the size to the compose selection rectangle on the scaler's sink pad, and the format to the scaler source pad.
4. The application configures the size of the compose selection rectangle of the scaler's sink pad 1280x960. The driver propagates the size to the scaler's source pad format.

When satisfied with the try results, applications can set the active formats by setting the which argument to V4L2_SUBDEV_FORMAT_ACTIVE. Active formats are changed exactly as try formats by drivers. To avoid modifying the hardware state during format negotiation, applications should negotiate try formats first and then modify the active settings using the try formats returned during the last negotiation iteration. This guarantees that the active format will be applied as-is by the driver without being modified.

Selections: cropping, scaling and composition

Many sub-devices support cropping frames on their input or output pads (or possible even on both). Cropping is used to select the area of interest in an image, typically on an image sensor or a video decoder. It can also be used as part of digital zoom implementations to select the area of the image that will be scaled up.

Crop settings are defined by a crop rectangle and represented in a struct `v4l2_rect` by the coordinates of the top left corner and the rectangle size. Both the coordinates and sizes are expressed in pixels.

As for pad formats, drivers store try and active rectangles for the selection targets *Common selection definitions*.

On sink pads, cropping is applied relative to the current pad format. The pad format represents the image size as received by the sub-device from the previous block in the pipeline, and the crop rectangle represents the sub-image that will be transmitted further inside the sub-device for processing.

The scaling operation changes the size of the image by scaling it to new dimensions. The scaling ratio isn't specified explicitly, but is implied from the original and scaled image sizes. Both sizes are represented by struct `v4l2_rect`.

Scaling support is optional. When supported by a subdev, the crop rectangle on the subdev's sink pad is scaled to the size configured using the `VIDIOC_SUBDEV_S_SELECTION` IOCTL using `V4L2_SEL_TGT_COMPOSE` selection target on the same pad. If the subdev supports scaling but not composing, the top and left values are not used and must always be set to zero.

On source pads, cropping is similar to sink pads, with the exception that the source size from which the cropping is performed, is the `COMPOSE` rectangle on the sink pad. In both sink and source pads, the crop rectangle must be entirely contained inside the source image size for the crop operation.

The drivers should always use the closest possible rectangle the user requests on all selection targets, unless specifically told otherwise. `V4L2_SEL_FLAG_GE` and `V4L2_SEL_FLAG_LE` flags may be used to round the image size either up or down. *Selection flags*

Types of selection targets

Actual targets

Actual targets (without a postfix) reflect the actual hardware configuration at any point of time. There is a `BOUNDS` target corresponding to every actual target.

BOUNDS targets

`BOUNDS` targets is the smallest rectangle that contains all valid actual rectangles. It may not be possible to set the actual rectangle as large as the `BOUNDS` rectangle, however. This may be because e.g. a sensor's pixel array is not rectangular but cross-shaped or round. The maximum size may also be smaller than the `BOUNDS` rectangle.

Order of configuration and format propagation

Inside subdevs, the order of image processing steps will always be from the sink pad towards the source pad. This is also reflected in the order in which the configuration must be performed by the user: the changes made will be propagated to any subsequent stages. If this behaviour is not desired, the user must set `V4L2_SEL_FLAG_KEEP_CONFIG` flag. This flag causes no propagation of the changes are allowed in any circumstances. This may also cause the accessed rectangle to be adjusted by the driver, depending on the properties of the underlying hardware.

The coordinates to a step always refer to the actual size of the previous step. The exception to this rule is the sink compose rectangle, which refers to the sink compose bounds rectangle — if it is supported by the hardware.

1. Sink pad format. The user configures the sink pad format. This format defines the parameters of the image the entity receives through the pad for further processing.
2. Sink pad actual crop selection. The sink pad crop defines the crop performed to the sink pad format.
3. Sink pad actual compose selection. The size of the sink pad compose rectangle defines the scaling ratio compared to the size of the sink pad crop rectangle. The location of the compose rectangle specifies the location of the actual sink compose rectangle in the sink compose bounds rectangle.
4. Source pad actual crop selection. Crop on the source pad defines crop performed to the image in the sink compose bounds rectangle.
5. Source pad format. The source pad format defines the output pixel format of the subdev, as well as the other parameters with the exception of the image width and height. Width and height are defined by the size of the source pad actual crop selection.

Accessing any of the above rectangles not supported by the subdev will return `EINVAL`. Any rectangle referring to a previous unsupported rectangle coordinates will instead refer to the previous supported rectangle. For example, if sink crop is not supported, the compose selection will refer to the sink pad format dimensions instead.

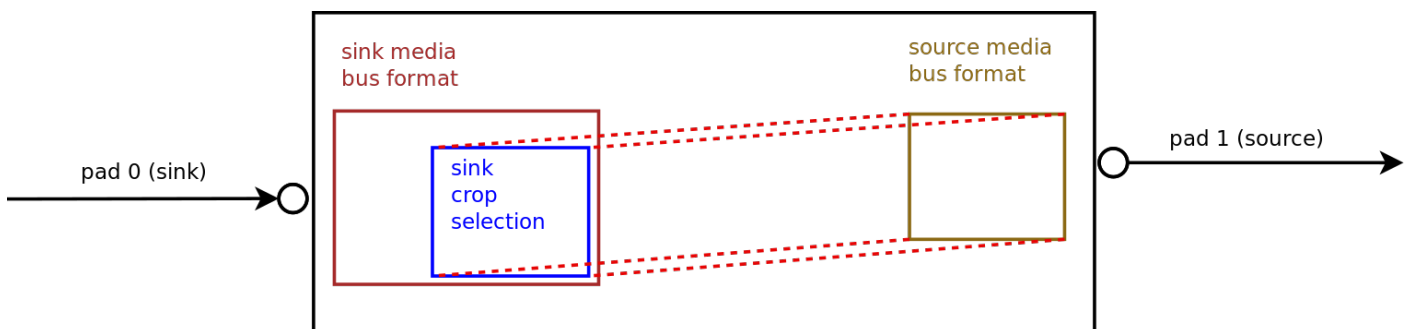


Fig. 14: **Figure 4.5. Image processing in subdevs: simple crop example**

In the above example, the subdev supports cropping on its sink pad. To configure it, the user sets the media bus format on the subdev's sink pad. Now the actual crop rectangle can be set on the sink pad — the location and size of this rectangle reflect the location and size of a rectangle to be cropped from the sink format. The size of the sink crop rectangle will also be the size of the format of the subdev's source pad.

In this example, the subdev is capable of first cropping, then scaling and finally cropping for two source pads individually from the resulting scaled image. The location of the scaled image in the cropped image is ignored in sink compose target. Both of the locations of the source crop rectangles refer to the sink scaling rectangle, independently cropping an area at location specified by the source crop rectangle from it.

The subdev driver supports two sink pads and two source pads. The images from both of the sink pads are individually cropped, then scaled and further composed on the composition bounds rectangle. From that, two independent streams are cropped and sent out of the subdev from the source pads.

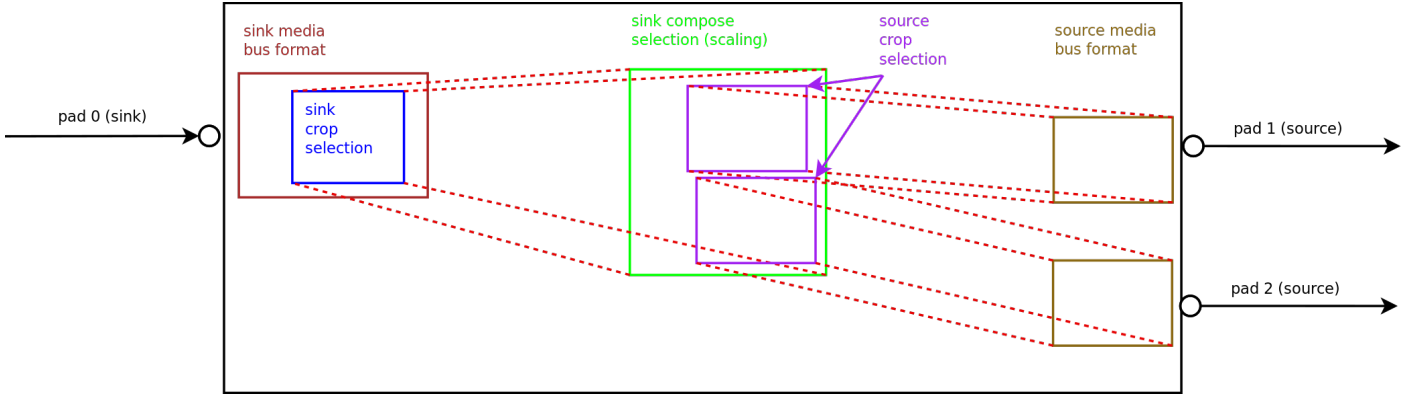


Fig. 15: **Figure 4.6. Image processing in subdevs: scaling with multiple sources**

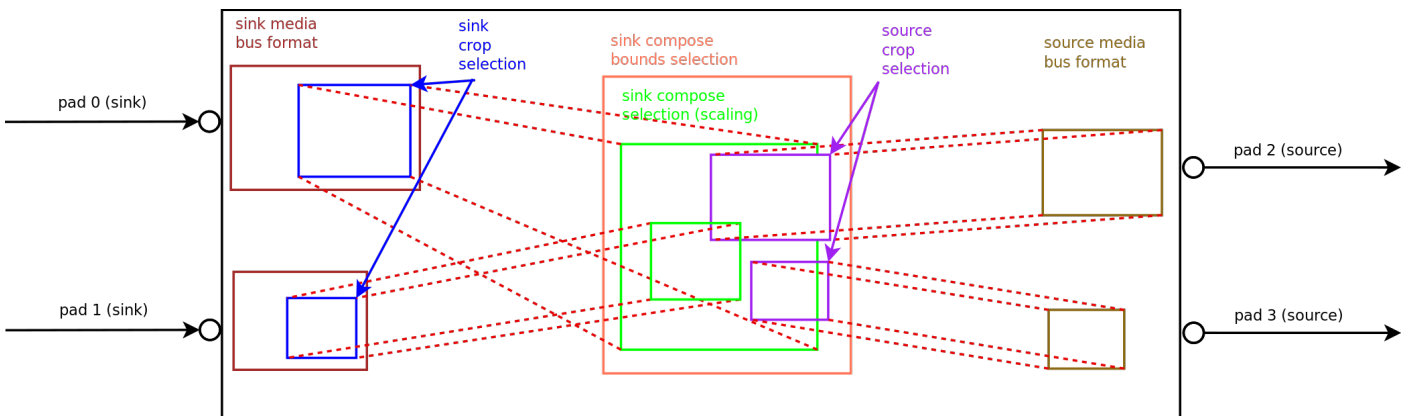


Fig. 16: **Figure 4.7. Image processing in subdevs: scaling and composition with multiple sinks and sources**

Media Bus Formats

v4l2_mbus_framefmt

Table 114: struct v4l2_mbus_framefmt

__u32	width	Image width in pixels.
__u32	height	Image height in pixels. If <code>field</code> is one of <code>V4L2_FIELD_TOP</code> , <code>V4L2_FIELD_BOTTOM</code> or <code>V4L2_FIELD_ALTERNATE</code> then height refers to the number of lines in the field, otherwise it refers to the number of lines in the frame (which is twice the field height for interlaced formats).
__u32	code	Format code, from enum v4l2_mbus_pixelcode .
__u32	field	Field order, from enum v4l2_field . See Field Order for details.
__u32	colorspace	Image colorspace, from enum v4l2_colorspace . Must be set by the driver for subdevices. If the application sets the flag <code>V4L2_MBUS_FRAMEFMT_SET_CSC</code> then the application can set this field on the source pad to request a specific colorspace for the media bus data. If the driver cannot handle the requested conversion, it will return another supported colorspace. The driver indicates that colorspace conversion is supported by setting the flag <code>V4L2_SUBDEV_MBUS_CODE_CSC_COLORSPACE</code> in the corresponding struct <code>v4l2_subdev_mbus_code_enum</code> during enumeration. See Subdev Media Bus Code Enumerate Flags .
union {	(anonymous)	
__u16	ycbcr_enc	Y'CbCr encoding, from enum v4l2_ycbcr_encoding . This information supplements the colorspace and must be set by the driver for subdevices, see Colorspaces . If the application sets the flag <code>V4L2_MBUS_FRAMEFMT_SET_CSC</code> then the application can set this field on a source pad to request a specific Y'CbCr encoding for the media bus data. If the driver cannot handle the requested conversion, it will return another supported encoding. This field is ignored for HSV media bus formats. The driver indicates that <code>ycbcr_enc</code> conversion is supported by setting the flag <code>V4L2_SUBDEV_MBUS_CODE_CSC_YCBCR_ENC</code> in the corresponding struct <code>v4l2_subdev_mbus_code_enum</code> during enumeration. See Subdev Media Bus Code Enumerate Flags .

Continued on next page

Table 114 - continued from previous page

__u16	hsv_enc	HSV encoding, from enum v4l2_hsv_encoding . This information supplements the colorspace and must be set by the driver for subdevices, see Colorspaces . If the application sets the flag V4L2_MBUS_FRAMEFMT_SET_CSC then the application can set this field on a source pad to request a specific HSV encoding for the media bus data. If the driver cannot handle the requested conversion, it will return another supported encoding. This field is ignored for Y'CbCr media bus formats. The driver indicates that hsv_enc conversion is supported by setting the flag V4L2_SUBDEV_MBUS_CODE_CSC_HSV_ENC in the corresponding struct v4l2_subdev_mbus_code_enum during enumeration. See Subdev Media Bus Code Enumerate Flags .
}		
__u16	quantization	Quantization range, from enum v4l2_quantization . This information supplements the colorspace and must be set by the driver for subdevices, see Colorspaces . If the application sets the flag V4L2_MBUS_FRAMEFMT_SET_CSC then the application can set this field on a source pad to request a specific quantization for the media bus data. If the driver cannot handle the requested conversion, it will return another supported quantization. The driver indicates that quantization conversion is supported by setting the flag V4L2_SUBDEV_MBUS_CODE_CSC_QUANTIZATION in the corresponding struct v4l2_subdev_mbus_code_enum during enumeration. See Subdev Media Bus Code Enumerate Flags .
__u16	xfer_func	Transfer function, from enum v4l2_xfer_func . This information supplements the colorspace and must be set by the driver for subdevices, see Colorspaces . If the application sets the flag V4L2_MBUS_FRAMEFMT_SET_CSC then the application can set this field on a source pad to request a specific transfer function for the media bus data. If the driver cannot handle the requested conversion, it will return another supported transfer function. The driver indicates that the transfer function conversion is supported by setting the flag V4L2_SUBDEV_MBUS_CODE_CSC_XFER_FUNC in the corresponding struct v4l2_subdev_mbus_code_enum during enumeration. See Subdev Media Bus Code Enumerate Flags .
__u16	flags	flags See: :ref:v4l2-mbus-framefmt-flags
__u16	reserved[10]	Reserved for future extensions. Applications and drivers must set the array to zero.

Table 115: v4l2_mbus_framefmt Flags

V4L2_MBUS_FRAMEFMT_SET_CSC	0x0001	Set by the application. It is only used for source pads and is ignored for sink pads. If set, then request the subdevice to do colorspace conversion from the received colorspace to the requested colorspace values. If the colorimetry field (colorspace, xfer_func, ycbcr_enc, hsv_enc or quantization) is set to *_DEFAULT, then that colorimetry setting will remain unchanged from what was received. So in order to change the quantization, only the quantization field shall be set to non default value (V4L2_QUANTIZATION_FULL_RANGE or V4L2_QUANTIZATION_LIM_RANGE) and all other colorimetry fields shall be set to *_DEFAULT. To check which conversions are supported by the hardware for the current media bus frame format, see <i>Subdev Media Bus Code Enumerate Flags</i> .
----------------------------	--------	---

Media Bus Pixel Codes

The media bus pixel codes describe image formats as flowing over physical buses (both between separate physical components and inside SoC devices). This should not be confused with the V4L2 pixel formats that describe, using four character codes, image formats as stored in memory.

While there is a relationship between image formats on buses and image formats in memory (a raw Bayer image won't be magically converted to JPEG just by storing it to memory), there is no one-to-one correspondence between them.

The media bus pixel codes document parallel formats. Should the pixel data be transported over a serial bus, the media bus pixel code that describes a parallel format that transfers a sample on a single clock cycle is used. For instance, both MEDIA_BUS_FMT_BGR888_1X24 and MEDIA_BUS_FMT_BGR888_3X8 are used on parallel busses for transferring an 8 bits per sample BGR data, whereas on serial busses the data in this format is only referred to using MEDIA_BUS_FMT_BGR888_1X24. This is because there is effectively only a single way to transport that format on the serial busses.

Packed RGB Formats

Those formats transfer pixel data as red, green and blue components. The format code is made of the following information.

- The red, green and blue components order code, as encoded in a pixel sample. Possible values are RGB and BGR.
- The number of bits per component, for each component. The values can be different for all components. Common values are 555 and 565.

- The number of bus samples per pixel. Pixels that are wider than the bus width must be transferred in multiple samples. Common values are 1 and 2.
- The bus width.
- For formats where the total number of bits per pixel is smaller than the number of bus samples per pixel times the bus width, a padding value stating if the bytes are padded in their most high order bits (PADHI) or low order bits (PADLO). A “C” prefix is used for component-wise padding in the most high order bits (CPADHI) or low order bits (CPADLO) of each separate component.
- For formats where the number of bus samples per pixel is larger than 1, an endianness value stating if the pixel is transferred MSB first (BE) or LSB first (LE).

For instance, a format where pixels are encoded as 5-bits red, 5-bits green and 5-bit blue values padded on the high bit, transferred as 2 8-bit samples per pixel with the most significant bits (padding, red and half of the green value) transferred first will be named MEDIA_BUS_FMT_RGB555_2X8_PADHI_BE.

The following tables list existing packed RGB formats.

Table 116: RGB formats

Identifier	Code	Data organization																																							
		Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0							
MEDIA_BUS_FMT_RGB444_1X12	0x1016																										r ₃	r ₂	r ₁	r ₀	g ₃	g ₂	g ₁	g ₀	b ₃	b ₂	b ₁	b ₀			
MEDIA_BUS_FMT_RGB444_2X8_PADHI_BE	0x1001																																					r ₃	r ₂	r ₁	r ₀
MEDIA_BUS_FMT_RGB444_2X8_PADHI_LE	0x1002																																								
MEDIA_BUS_FMT_RGB555_2X8_PADHI_BE	0x1003																																								
MEDIA_BUS_FMT_RGB555_2X8_PADHI_LE	0x1004																																								
MEDIA_BUS_FMT_RGB565_1X16	0x1017																																								
MEDIA_BUS_FMT_BGR565_2X8_BE	0x1005																																								
MEDIA_BUS_FMT_BGR565_2X8_LE	0x1006																																								
MEDIA_BUS_FMT_RGB565_2X8_BE	0x1007																																								
MEDIA_BUS_FMT_RGB565_2X8_LE	0x1008																																								
MEDIA_BUS_FMT_RGB666_1X18	0x1009																																								
MEDIA_BUS_FMT_RGB888_1X24	0x100e																																								
MEDIA_BUS_FMT_RGB666_1X24_CPADHI	0x1015																																								
MEDIA_BUS_FMT_BGR888_1X24	0x1013																																								
MEDIA_BUS_FMT_BGR888_3X8	0x101b																																								
MEDIA_BUS_FMT_GBR888_1X24	0x1014																																								
MEDIA_BUS_FMT_RGB888_1X24	0x100a																																								
MEDIA_BUS_FMT_RGB888_2X12_BE	0x100b																																								
MEDIA_BUS_FMT_RGB888_2X12_LE	0x100c																																								
MEDIA_BUS_FMT_RGB888_3X8	0x101c																																								
MEDIA_BUS_FMT_ARGB888_1X32	0x100d																																								
MEDIA_BUS_FMT_RGB888_1X32_PADHI	0x100f																																								

Continued on next page

Bayer Formats

Those formats transfer pixel data as red, green and blue components. The format code is made of the following information.

- The red, green and blue components order code, as encoded in a pixel sample. The possible values are shown in *Figure 4.8 Bayer Patterns*.
- The number of bits per pixel component. All components are transferred on the same number of bits. Common values are 8, 10 and 12.
- The compression (optional). If the pixel components are ALAW- or DPCM-compressed, a mention of the compression scheme and the number of bits per compressed pixel component.
- The number of bus samples per pixel. Pixels that are wider than the bus width must be transferred in multiple samples. Common values are 1 and 2.
- The bus width.
- For formats where the total number of bits per pixel is smaller than the number of bus samples per pixel times the bus width, a padding value stating if the bytes are padded in their most high order bits (PADHI) or low order bits (PADLO).
- For formats where the number of bus samples per pixel is larger than 1, an endianness value stating if the pixel is transferred MSB first (BE) or LSB first (LE).

For instance, a format with uncompressed 10-bit Bayer components arranged in a red, green, blue pattern transferred as 2 8-bit samples per pixel with the least significant bits transferred first will be named MEDIA_BUS_FMT_SRGG8_2X8_PADHI_LE.

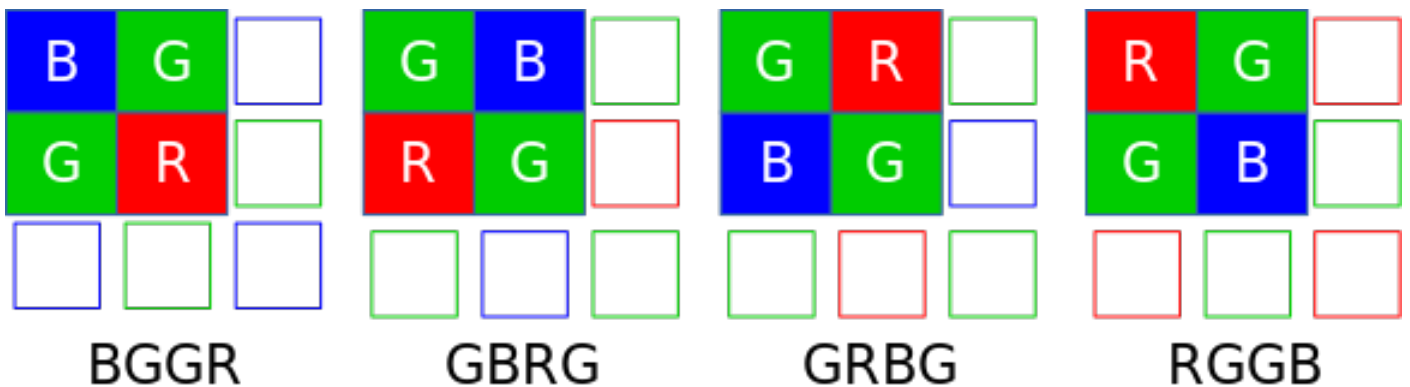


Fig. 17: Figure 4.8 Bayer Patterns

The following table lists existing packed Bayer formats. The data organization is given as an example for the first pixel only.

Table 120: Bayer Formats

Identifier	Code	Data organization																	
		Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
MEDIA_BUS_FMT_SBGG8_1X8	0x3001											b ₇	b ₆	b ₅	b ₄	b ₃	b ₂	b ₁	b ₀
MEDIA_BUS_FMT_SGBRG8_1X8	0x3013											g ₇	g ₆	g ₅	g ₄	g ₃	g ₂	g ₁	g ₀
MEDIA_BUS_FMT_SGRBG8_1X8	0x3002											g ₇	g ₆	g ₅	g ₄	g ₃	g ₂	g ₁	g ₀
MEDIA_BUS_FMT_SRGG8_1X8	0x3014											r ₇	r ₆	r ₅	r ₄	r ₃	r ₂	r ₁	r ₀
MEDIA_BUS_FMT_SBGG10_ALAW8_1X8	0x3015											b ₇	b ₆	b ₅	b ₄	b ₃	b ₂	b ₁	b ₀

Continued on next page

Table 124: HSV/HSL formats

Identifier	Code	Data organization																															
		Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1
MEDIA_BUS_FMT_AHSV8888_1X32	0x6001	a7	a6	a5	a4	a3	a2	a1	a0	h7	h6	h5	h4	h3	h2	h1	h0	s7	s6	s5	s4	s3	s2	s1	s0	v7	v6	v5	v4	v3	v2	v1	v0

JPEG Compressed Formats

Those data formats consist of an ordered sequence of 8-bit bytes obtained from JPEG compression process. Additionally to the `_JPEG` postfix the format code is made of the following information.

- The number of bus samples per entropy encoded byte.
- The bus width.

For instance, for a JPEG baseline process and an 8-bit bus width the format will be named `MEDIA_BUS_FMT_JPEG_1X8`.

The following table lists existing JPEG compressed formats.

Table 125: JPEG Formats

Identifier	Code	Remarks
<code>MEDIA_BUS_FMT_JPEG_1X8</code>	0x4001	Besides of its usage for the parallel bus this format is recommended for transmission of JPEG data over MIPI CSI bus using the User Defined 8-bit Data types.

Vendor and Device Specific Formats

This section lists complex data formats that are either vendor or device specific.

The following table lists the existing vendor and device specific formats.

Table 126: Vendor and device specific formats

Identifier	Code	Comments
<code>MEDIA_BUS_FMT_S5C_UYVY_JPEG_1X8</code>	0x5001	Interleaved raw UYVY and JPEG image format with embedded meta-data used by Samsung S3C73MX camera sensors.

Metadata Formats

This section lists all metadata formats.

The following table lists the existing metadata formats.

Table 127: Metadata formats

Identifier	Code	Comments
MEDIA_BUS_FMT_METADATA_FIXED	0x7001	This format should be used when the same driver handles both sides of the link and the bus format is a fixed metadata format that is not configurable from userspace. Width and height will be set to 0 for this format.

3.2.4.14 Metadata Interface

Metadata refers to any non-image data that supplements video frames with additional information. This may include statistics computed over the image, frame capture parameters supplied by the image source or device specific parameters for specifying how the device processes images. This interface is intended for transfer of metadata between the userspace and the hardware and control of that operation.

The metadata interface is implemented on video device nodes. The device can be dedicated to metadata or can support both video and metadata as specified in its reported capabilities.

Querying Capabilities

Device nodes supporting the metadata capture interface set the `V4L2_CAP_META_CAPTURE` flag in the `device_caps` field of the `v4l2_capability` structure returned by the `VIDIOC_QUERYCAP()` ioctl. That flag means the device can capture metadata to memory. Similarly, device nodes supporting metadata output interface set the `V4L2_CAP_META_OUTPUT` flag in the `device_caps` field of `v4l2_capability` structure. That flag means the device can read metadata from memory.

At least one of the read/write or streaming I/O methods must be supported.

Data Format Negotiation

The metadata device uses the *Data Formats* ioctls to select the capture format. The metadata buffer content format is bound to that selected format. In addition to the basic *Data Formats* ioctls, the `VIDIOC_ENUM_FMT()` ioctl must be supported as well.

To use the *Data Formats* ioctls applications set the `type` field of the `v4l2_format` structure to `V4L2_BUF_TYPE_META_CAPTURE` or to `V4L2_BUF_TYPE_META_OUTPUT` and use the `v4l2_meta_format` meta member of the `fmt` union as needed per the desired operation. Both drivers and applications must set the remainder of the `v4l2_format` structure to 0.

`v4l2_meta_format`

Table 128: struct `v4l2_meta_format`

<code>__u32</code>	<code>dataformat</code>	The data format, set by the application. This is a little endian <i>four character code</i> . V4L2 defines metadata formats in <i>Metadata Formats</i> .
<code>__u32</code>	<code>buffersize</code>	Maximum buffer size in bytes required for data. The value is set by the driver.

3.2.5 Libv4l Userspace Library

3.2.5.1 Introduction

libv4l is a collection of libraries which adds a thin abstraction layer on top of video4linux2 devices. The purpose of this (thin) layer is to make it easy for application writers to support a wide variety of devices without having to write separate code for different devices in the same class.

An example of using libv4l is provided by *v4l2grab*.

libv4l consists of 3 different libraries:

libv4lconvert

libv4lconvert is a library that converts several different pixelformats found in V4L2 drivers into a few common RGB and YUY formats.

It currently accepts the following V4L2 driver formats: *V4L2_PIX_FMT_BGR24*,
V4L2_PIX_FMT_HM12, *V4L2_PIX_FMT_JPEG*, *V4L2_PIX_FMT_MJPEG*,
V4L2_PIX_FMT_MR97310A, *V4L2_PIX_FMT_OV511*, *V4L2_PIX_FMT_OV518*,
V4L2_PIX_FMT_PAC207, *V4L2_PIX_FMT_PJPG*, *V4L2_PIX_FMT_RGB24*,
V4L2_PIX_FMT_SBGGR8, *V4L2_PIX_FMT_SGBRG8*, *V4L2_PIX_FMT_SGRBG8*,
V4L2_PIX_FMT_SN9C10X, *V4L2_PIX_FMT_SN9C20X_I420*, *V4L2_PIX_FMT_SPCA501*,
V4L2_PIX_FMT_SPCA505, *V4L2_PIX_FMT_SPCA508*, *V4L2_PIX_FMT_SPCA561*,
V4L2_PIX_FMT_SQ905C, *V4L2_PIX_FMT_SRGGB8*, *V4L2_PIX_FMT_UYVY*,
V4L2_PIX_FMT_YUV420, *V4L2_PIX_FMT_YUYV*, *V4L2_PIX_FMT_YVU420*, and
V4L2_PIX_FMT_YVYU.

Later on libv4lconvert was expanded to also be able to do various video processing functions to improve webcam video quality. The video processing is split in to 2 parts: libv4lconvert/control and libv4lconvert/processing.

The control part is used to offer video controls which can be used to control the video processing functions made available by libv4lconvert/processing. These controls are stored application wide (until reboot) by using a persistent shared memory object.

libv4lconvert/processing offers the actual video processing functionality.

libv4l1

This library offers functions that can be used to quickly make v4l1 applications work with v4l2 devices. These functions work exactly like the normal open/close/etc, except that libv4l1 does full emulation of the v4l1 api on top of v4l2 drivers, in case of v4l1 drivers it will just pass calls through.

Since those functions are emulations of the old V4L1 API, it shouldn't be used for new applications.

libv4l2

This library should be used for all modern V4L2 applications.

It provides handles to call V4L2 open/ioctl/close/poll methods. Instead of just providing the raw output of the device, it enhances the calls in the sense that it will use libv4lconvert to provide more video formats and to enhance the image quality.

In most cases, libv4l2 just passes the calls directly through to the v4l2 driver, intercepting the calls to `VIDIOC_TRY_FMT`, `VIDIOC_G_FMT`, `VIDIOC_S_FMT`, `VIDIOC_ENUM_FRAMESIZES` and `VIDIOC_ENUM_FRAMEINTERVALS` in order to emulate the formats `V4L2_PIX_FMT_BGR24`, `V4L2_PIX_FMT_RGB24`, `V4L2_PIX_FMT_YUV420`, and `V4L2_PIX_FMT_YVU420`, if they aren't available in the driver. `VIDIOC_ENUM_FMT` keeps enumerating the hardware supported formats, plus the emulated formats offered by libv4l at the end.

Libv4l device control functions

The common file operation methods are provided by libv4l.

Those functions operate just like the gcc function `dup()` and V4L2 functions `open()`, `close()`, `ioctl()`, `read()`, `mmap()` and `munmap()`:

`int v4l2_open(const char *file, int oflag, ...)`
operates like the `open()` function.

`int v4l2_close(int fd)`
operates like the `close()` function.

`int v4l2_dup(int fd)`
operates like the libc `dup()` function, duplicating a file handler.

`int v4l2_ioctl(int fd, unsigned long int request, ...)`
operates like the `ioctl()` function.

`int v4l2_read(int fd, void* buffer, size_t n)`
operates like the `read()` function.

`void v4l2_mmap(void *start, size_t length, int prot, int flags, int fd, int64_t offset)`
operates like the `mmap()` function.

`int v4l2_munmap(void *_start, size_t length);`
operates like the `munmap()` function.

Those functions provide additional control:

`int v4l2_fd_open(int fd, int v4l2_flags)`
opens an already opened `fd` for further use through `v4l2lib` and possibly modify `libv4l2`'s default behavior through the `v4l2_flags` argument. Currently, `v4l2_flags` can be `V4L2_DISABLE_CONVERSION`, to disable format conversion.

`int v4l2_set_control(int fd, int cid, int value)`
This function takes a value of 0 - 65535, and then scales that range to the actual range of the given `v4l` control id, and then if the `cid` exists and is not locked sets the `cid` to the scaled value.

int **v4l2_get_control**(int *fd*, int *cid*)

This function returns a value of 0 - 65535, scaled to from the actual range of the given v4l control id. when the cid does not exist, could not be accessed for some reason, or some error occurred 0 is returned.

v4l1compat.so wrapper library

This library intercepts calls to `open()`, `close()`, `ioctl()`, `mmap()` and `munmap()` operations and redirects them to the `libv4l` counterparts, by using `LD_PRELOAD=/usr/lib/v4l1compat.so`. It also emulates V4L1 calls via V4L2 API.

It allows usage of binary legacy applications that still don't use `libv4l`.

3.2.6 Changes

The following chapters document the evolution of the V4L2 API, errata or extensions. They are also intended to help application and driver writers to port or update their code.

3.2.6.1 Differences between V4L and V4L2

The Video For Linux API was first introduced in Linux 2.1 to unify and replace various TV and radio device related interfaces, developed independently by driver writers in prior years. Starting with Linux 2.5 the much improved V4L2 API replaces the V4L API. The support for the old V4L calls were removed from Kernel, but the library *Libv4l Userspace Library* supports the conversion of a V4L API system call into a V4L2 one.

Opening and Closing Devices

For compatibility reasons the character device file names recommended for V4L2 video capture, overlay, radio and raw vbi capture devices did not change from those used by V4L. They are listed in *Interfaces* and below in *V4L Device Types, Names and Numbers*.

The teletext devices (minor range 192-223) have been removed in V4L2 and no longer exist. There is no hardware available anymore for handling pure teletext. Instead raw or sliced VBI is used.

The V4L `videodev` module automatically assigns minor numbers to drivers in load order, depending on the registered device type. We recommend that V4L2 drivers by default register devices with the same numbers, but the system administrator can assign arbitrary minor numbers using driver module options. The major device number remains 81.

Table 129: V4L Device Types, Names and Numbers

Device Type	File Name	Minor Numbers
Video capture and overlay	<code>/dev/video</code> and <code>/dev/bttv0</code> ¹ , <code>/dev/video0</code> to <code>/dev/video63</code>	0-63
Radio receiver	<code>/dev/radio</code> ² , <code>/dev/radio0</code> to <code>/dev/radio63</code>	64-127
Raw VBI capture	<code>/dev/vbi</code> , <code>/dev/vbi0</code> to <code>/dev/vbi31</code>	224-255

V4L prohibits (or used to prohibit) multiple opens of a device file. V4L2 drivers *may* support multiple opens, see *Opening and Closing Devices* for details and consequences.

V4L drivers respond to V4L2 ioctls with an EINVAL error code.

Querying Capabilities

The V4L VIDIOCGCAP ioctl is equivalent to V4L2's *ioctl VIDIOC_QUERYCAP*.

The name field in struct `video_capability` became `card` in struct `v4l2_capability`, `type` was replaced by `capabilities`. Note V4L2 does not distinguish between device types like this, better think of basic video input, video output and radio devices supporting a set of related functions like video capturing, video overlay and VBI capturing. See *Opening and Closing Devices* for an introduction.

struct <code>video_capability</code> type	struct <code>v4l2_capability</code> capabilities flags	Purpose
VID_TYPE_CAPTURE	V4L2_CAP_VIDEO_CAPTURE	The <i>video capture</i> interface is supported.
VID_TYPE_TUNER	V4L2_CAP_TUNER	The device has a <i>tuner or modulator</i> .
VID_TYPE_TELETEXT	V4L2_CAP_VBI_CAPTURE	The <i>raw VBI capture</i> interface is supported.
VID_TYPE_OVERLAY	V4L2_CAP_VIDEO_OVERLAY	The <i>video overlay</i> interface is supported.
VID_TYPE_CHROMAKEY	V4L2_FBUF_CAP_CHROMAKEY in field capability of struct <code>v4l2_framebuffer</code>	Whether chromakey overlay is supported. For more information on overlay see <i>Video Overlay Interface</i> .
VID_TYPE_CLIPPING	V4L2_FBUF_CAP_LIST_CLIPPING and V4L2_FBUF_CAP_BITMAP_CLIPPING in field capability of struct <code>v4l2_framebuffer</code>	Whether clipping the overlaid image is supported, see <i>Video Overlay Interface</i> .
VID_TYPE_FRAMERAM	V4L2_FBUF_CAP_EXTERNOVERLAY <i>not set</i> in field capability of struct <code>v4l2_framebuffer</code>	Whether overlay overwrites frame buffer memory, see <i>Video Overlay Interface</i> .
VID_TYPE_SCALES	-	This flag indicates if the hardware can scale images. The V4L2 API implies the scale factor by setting the cropping dimensions and image size with the <i>VIDIOC_S_CROP</i> and <i>VIDIOC_S_FMT</i> ioctl, respectively. The driver returns the closest sizes possible. For more information on cropping and scaling see <i>Image Cropping, Insertion and Scaling - the CROP API</i> .

Continued on next page

¹ According to Documentation/admin-guide/devices.rst these should be symbolic links to `/dev/video0`. Note the original bttv interface is not compatible with V4L or V4L2.

² According to Documentation/admin-guide/devices.rst a symbolic link to `/dev/radio0`.

Table 130 - continued from previous page

struct video_capability type	struct v4l2_capability capabilities flags	Purpose
VID_TYPE_MONOCHROME	-	Applications can enumerate the supported image formats with the <i>ioctl VIDIOC_ENUM_FMT</i> ioctl to determine if the device supports grey scale capturing only. For more information on image formats see <i>Image Formats</i> .
VID_TYPE_SUBCAPTURE	-	Applications can call the <i>VIDIOC_G_CROP</i> ioctl to determine if the device supports capturing a subsection of the full picture (“cropping” in V4L2). If not, the ioctl returns the EINVAL error code. For more information on cropping and scaling see <i>Image Cropping, Insertion and Scaling - the CROP API</i> .
VID_TYPE_MPEG_DECODER	-	Applications can enumerate the supported image formats with the <i>ioctl VIDIOC_ENUM_FMT</i> ioctl to determine if the device supports MPEG streams.
VID_TYPE_MPEG_ENCODER	-	See above.
VID_TYPE_MJPEG_DECODER	-	See above.
VID_TYPE_MJPEG_ENCODER	-	See above.

The audios field was replaced by capabilities flag V4L2_CAP_AUDIO, indicating *if* the device has any audio inputs or outputs. To determine their number applications can enumerate audio inputs with the *VIDIOC_G_AUDIO* ioctl. The audio ioctls are described in *Audio Inputs and Outputs*.

The maxwidth, maxheight, minwidth and minheight fields were removed. Calling the *VIDIOC_S_FMT* or *VIDIOC_TRY_FMT* ioctl with the desired dimensions returns the closest size possible, taking into account the current video standard, cropping and scaling limitations.

Video Sources

V4L provides the VIDIOCGCHAN and VIDIOCSCCHAN ioctl using struct video_channel to enumerate the video inputs of a V4L device. The equivalent V4L2 ioctls are *ioctl VIDIOC_ENUMINPUT*, *VIDIOC_G_INPUT* and *VIDIOC_S_INPUT* using struct v4l2_input as discussed in *Video Inputs and Outputs*.

The channel field counting inputs was renamed to index, the video input types were renamed as follows:

struct video_channel type	struct v4l2_input type
VIDEO_TYPE_TV	V4L2_INPUT_TYPE_TUNER
VIDEO_TYPE_CAMERA	V4L2_INPUT_TYPE_CAMERA

Unlike the `tuners` field expressing the number of tuners of this input, V4L2 assumes each video input is connected to at most one tuner. However a tuner can have more than one input, i. e. RF connectors, and a device can have multiple tuners. The index number of the tuner associated with the input, if any, is stored in field `tuner` of struct `v4l2_input`. Enumeration of tuners is discussed in *Tuners and Modulators*.

The redundant `VIDEO_VC_TUNER` flag was dropped. Video inputs associated with a tuner are of type `V4L2_INPUT_TYPE_TUNER`. The `VIDEO_VC_AUDIO` flag was replaced by the `audioset` field. V4L2 considers devices with up to 32 audio inputs. Each set bit in the `audioset` field represents one audio input this video input combines with. For information about audio inputs and how to switch between them see *Audio Inputs and Outputs*.

The `norm` field describing the supported video standards was replaced by `std`. The V4L specification mentions a flag `VIDEO_VC_NORM` indicating whether the standard can be changed. This flag was a later addition together with the `norm` field and has been removed in the meantime. V4L2 has a similar, albeit more comprehensive approach to video standards, see *Video Standards* for more information.

Tuning

The V4L `VIDIOCGTUNER` and `VIDIOCSTUNER` ioctl and struct `video_tuner` can be used to enumerate the tuners of a V4L TV or radio device. The equivalent V4L2 ioctls are `VIDIOC_G_TUNER` and `VIDIOC_S_TUNER` using struct `v4l2_tuner`. Tuners are covered in *Tuners and Modulators*.

The `tuner` field counting tuners was renamed to `index`. The fields `name`, `rangelow` and `rangehigh` remained unchanged.

The `VIDEO_TUNER_PAL`, `VIDEO_TUNER_NTSC` and `VIDEO_TUNER_SECAM` flags indicating the supported video standards were dropped. This information is now contained in the associated struct `v4l2_input`. No replacement exists for the `VIDEO_TUNER_NORM` flag indicating whether the video standard can be switched. The `mode` field to select a different video standard was replaced by a whole new set of ioctls and structures described in *Video Standards*. Due to its ubiquity it should be mentioned the BTTV driver supports several standards in addition to the regular `VIDEO_MODE_PAL` (0), `VIDEO_MODE_NTSC`, `VIDEO_MODE_SECAM` and `VIDEO_MODE_AUTO` (3). Namely N/PAL Argentina, M/PAL, N/PAL, and NTSC Japan with numbers 3-6 (sic).

The `VIDEO_TUNER_STEREO_ON` flag indicating stereo reception became `V4L2_TUNER_SUB_STEREO` in field `rxsubchans`. This field also permits the detection of monaural and bilingual audio, see the definition of struct `v4l2_tuner` for details. Presently no replacement exists for the `VIDEO_TUNER_RDS_ON` and `VIDEO_TUNER_MBS_ON` flags.

The `VIDEO_TUNER_LOW` flag was renamed to `V4L2_TUNER_CAP_LOW` in the struct `v4l2_tuner` capability field.

The `VIDIOCGFREQ` and `VIDIOC_SFREQ` ioctl to change the tuner frequency were renamed to `VIDIOC_G_FREQUENCY` and `VIDIOC_S_FREQUENCY`. They take a pointer to a struct `v4l2_frequency` instead of an unsigned long integer.

Image Properties

V4L2 has no equivalent of the `VIDIOCGPICT` and `VIDIOCSPICT` `ioctl` and `struct video_picture`. The following fields were replaced by V4L2 controls accessible with the *ioctls* `VIDIOC_QUERYCTRL`, `VIDIOC_QUERY_EXT_CTRL` and `VIDIOC_QUERYMENU`, `VIDIOC_G_CTRL` and `VIDIOC_S_CTRL` `ioctls`:

<code>struct video_picture</code>	V4L2 Control ID
<code>brightness</code>	<code>V4L2_CID_BRIGHTNESS</code>
<code>hue</code>	<code>V4L2_CID_HUE</code>
<code>colour</code>	<code>V4L2_CID_SATURATION</code>
<code>contrast</code>	<code>V4L2_CID_CONTRAST</code>
<code>whiteness</code>	<code>V4L2_CID_WHITENESS</code>

The V4L picture controls are assumed to range from 0 to 65535 with no particular reset value. The V4L2 API permits arbitrary limits and defaults which can be queried with the *ioctls* `VIDIOC_QUERYCTRL`, `VIDIOC_QUERY_EXT_CTRL` and `VIDIOC_QUERYMENU` `ioctl`. For general information about controls see *User Controls*.

The depth (average number of bits per pixel) of a video image is implied by the selected image format. V4L2 does not explicitly provide such information assuming applications recognizing the format are aware of the image depth and others need not know. The palette field moved into the struct `v4l2_pix_format`:

<code>struct video_picture palette</code>	<code>struct v4l2_pix_format pixfmt</code>
<code>VIDEO_PALETTE_GREY</code>	<code>V4L2_PIX_FMT_GREY</code>
<code>VIDEO_PALETTE_HI240</code>	<code>V4L2_PIX_FMT_HI240</code> ³
<code>VIDEO_PALETTE_RGB565</code>	<code>V4L2_PIX_FMT_RGB565</code>
<code>VIDEO_PALETTE_RGB555</code>	<code>V4L2_PIX_FMT_RGB555</code>
<code>VIDEO_PALETTE_RGB24</code>	<code>V4L2_PIX_FMT_BGR24</code>
<code>VIDEO_PALETTE_RGB32</code>	<code>V4L2_PIX_FMT_BGR32</code> ⁴
<code>VIDEO_PALETTE_YUV422</code>	<code>V4L2_PIX_FMT_YUYV</code>
<code>VIDEO_PALETTE_YUYV</code> ⁵	<code>V4L2_PIX_FMT_YUYV</code>
<code>VIDEO_PALETTE_UYVY</code>	<code>V4L2_PIX_FMT_UYVY</code>
<code>VIDEO_PALETTE_YUV420</code>	None
<code>VIDEO_PALETTE_YUV411</code>	<code>V4L2_PIX_FMT_Y41P</code> ⁶
<code>VIDEO_PALETTE_RAW</code>	None ⁷
<code>VIDEO_PALETTE_YUV422P</code>	<code>V4L2_PIX_FMT_YUV422P</code>
<code>VIDEO_PALETTE_YUV411P</code>	<code>V4L2_PIX_FMT_YUV411P</code> ⁸
<code>VIDEO_PALETTE_YUV420P</code>	<code>V4L2_PIX_FMT_YVU420</code>
<code>VIDEO_PALETTE_YUV410P</code>	<code>V4L2_PIX_FMT_YVU410</code>

V4L2 image formats are defined in *Image Formats*. The image format can be selected with the `VIDIOC_S_FMT` `ioctl`.

³ This is a custom format used by the BTTV driver, not one of the V4L2 standard formats.

⁴ Presumably all V4L RGB formats are little-endian, although some drivers might interpret them according to machine endianness. V4L2 defines little-endian, big-endian and red/blue swapped variants. For details see *RGB Formats*.

⁵ `VIDEO_PALETTE_YUV422` and `VIDEO_PALETTE_YUYV` are the same formats. Some V4L drivers respond to one,

Audio

The `VIDIOCGAUDIO` and `VIDIOC_S_AUDIO` ioctl and struct `video_audio` are used to enumerate the audio inputs of a V4L device. The equivalent V4L2 ioctls are `VIDIOC_G_AUDIO` and `VIDIOC_S_AUDIO` using struct `v4l2_audio` as discussed in *Audio Inputs and Outputs*.

The audio “channel number” field counting audio inputs was renamed to `index`.

On `VIDIOC_S_AUDIO` the `mode` field selects *one* of the `VIDEO_SOUND_MONO`, `VIDEO_SOUND_STEREO`, `VIDEO_SOUND_LANG1` or `VIDEO_SOUND_LANG2` audio demodulation modes. When the current audio standard is BTSC `VIDEO_SOUND_LANG2` refers to SAP and `VIDEO_SOUND_LANG1` is meaningless. Also undocumented in the V4L specification, there is no way to query the selected mode. On `VIDIOCGAUDIO` the driver returns the *actually received* audio programmes in this field. In the V4L2 API this information is stored in the struct `v4l2_tuner` `rxsubchans` and `audmode` fields, respectively. See *Tuners and Modulators* for more information on tuners. Related to audio modes struct `v4l2_audio` also reports if this is a mono or stereo input, regardless if the source is a tuner.

The following fields were replaced by V4L2 controls accessible with the ioctls `VIDIOC_QUERYCTRL`, `VIDIOC_QUERY_EXT_CTRL` and `VIDIOC_QUERYMENU`, `VIDIOC_G_CTRL` and `VIDIOC_S_CTRL` ioctls:

struct <code>video_audio</code>	V4L2 Control ID
<code>volume</code>	<code>V4L2_CID_AUDIO_VOLUME</code>
<code>bass</code>	<code>V4L2_CID_AUDIO_BASS</code>
<code>treble</code>	<code>V4L2_CID_AUDIO_TREBLE</code>
<code>balance</code>	<code>V4L2_CID_AUDIO_BALANCE</code>

To determine which of these controls are supported by a driver V4L provides the flags `VIDEO_AUDIO_VOLUME`, `VIDEO_AUDIO_BASS`, `VIDEO_AUDIO_TREBLE` and `VIDEO_AUDIO_BALANCE`. In the V4L2 API the ioctls `VIDIOC_QUERYCTRL`, `VIDIOC_QUERY_EXT_CTRL` and `VIDIOC_QUERYMENU` ioctl reports if the respective control is supported. Accordingly the `VIDEO_AUDIO_MUTABLE` and `VIDEO_AUDIO_MUTE` flags were replaced by the boolean `V4L2_CID_AUDIO_MUTE` control.

All V4L2 controls have a `step` attribute replacing the struct `video_audio` `step` field. The V4L audio controls are assumed to range from 0 to 65535 with no particular reset value. The V4L2 API permits arbitrary limits and defaults which can be queried with the ioctls `VIDIOC_QUERYCTRL`, `VIDIOC_QUERY_EXT_CTRL` and `VIDIOC_QUERYMENU` ioctl. For general information about controls see *User Controls*.

some to the other.

⁶ Not to be confused with `V4L2_PIX_FMT_YUV411P`, which is a planar format.

⁷ V4L explains this as: “RAW capture (BT848)”

⁸ Not to be confused with `V4L2_PIX_FMT_Y41P`, which is a packed format.

Frame Buffer Overlay

The V4L2 ioctls equivalent to VIDIOCGBUF and VIDIOCSBUF are *VIDIOC_G_FBUF* and *VIDIOC_S_FBUF*. The base field of struct `video_buffer` remained unchanged, except V4L2 defines a flag to indicate non-destructive overlays instead of a NULL pointer. All other fields moved into the struct *v4l2_pix_format* `fmt` substructure of struct `v4l2_framebuffer`. The `depth` field was replaced by `pixelformat`. See *RGB Formats* for a list of RGB formats and their respective color depths.

Instead of the special ioctls VIDIOCGWIN and VIDIOCSWIN V4L2 uses the general-purpose data format negotiation ioctls *VIDIOC_G_FMT* and *VIDIOC_S_FMT*. They take a pointer to a struct `v4l2_format` as argument. Here the `win` member of the `fmt` union is used, a struct `v4l2_window`.

The `x`, `y`, `width` and `height` fields of struct `video_window` moved into struct `v4l2_rect` substructure `w` of struct `v4l2_window`. The `chromakey`, `clips`, and `clipcount` fields remained unchanged. Struct `video_clip` was renamed to struct `v4l2_clip`, also containing a struct `v4l2_rect`, but the semantics are still the same.

The VIDEO_WINDOW_INTERLACE flag was dropped. Instead applications must set the `field` field to V4L2_FIELD_ANY or V4L2_FIELD_INTERLACED. The VIDEO_WINDOW_CHROMAKEY flag moved into struct `v4l2_framebuffer`, under the new name V4L2_FBUF_FLAG_CHROMAKEY.

In V4L, storing a bitmap pointer in `clips` and setting `clipcount` to VIDEO_CLIP_BITMAP (-1) requests bitmap clipping, using a fixed size bitmap of 1024 × 625 bits. Struct `v4l2_window` has a separate bitmap pointer field for this purpose and the bitmap size is determined by `w.width` and `w.height`.

The VIDIOCCAPTURE ioctl to enable or disable overlay was renamed to *ioctl VIDIOC_OVERLAY*.

Cropping

To capture only a subsection of the full picture V4L defines the VIDIOCGCAPTURE and VIDIOCSCAPTURE ioctls using struct `video_capture`. The equivalent V4L2 ioctls are *VIDIOC_G_CROP* and *VIDIOC_S_CROP* using struct `v4l2_crop`, and the related *ioctl VIDIOC_CROPCAP* ioctl. This is a rather complex matter, see *Image Cropping, Insertion and Scaling - the CROP API* for details.

The `x`, `y`, `width` and `height` fields moved into struct `v4l2_rect` substructure `c` of struct `v4l2_crop`. The `decimation` field was dropped. In the V4L2 API the scaling factor is implied by the size of the cropping rectangle and the size of the captured or overlaid image.

The VIDEO_CAPTURE_ODD and VIDEO_CAPTURE_EVEN flags to capture only the odd or even field, respectively, were replaced by V4L2_FIELD_TOP and V4L2_FIELD_BOTTOM in the `field` named field of struct *v4l2_pix_format* and struct `v4l2_window`. These structures are used to select a capture or overlay format with the *VIDIOC_S_FMT* ioctl.

Reading Images, Memory Mapping

Capturing using the read method

There is no essential difference between reading images from a V4L or V4L2 device using the `read()` function, however V4L2 drivers are not required to support this I/O method. Applications can determine if the function is available with the *ioctl* `VIDIOC_QUERYCAP` *ioctl*. All V4L2 devices exchanging data with applications must support the `select()` and `poll()` functions.

To select an image format and size, V4L provides the `VIDIOCSPICT` and `VIDIOC_SWIN` *ioctls*. V4L2 uses the general-purpose data format negotiation *ioctls* `VIDIOC_G_FMT` and `VIDIOC_S_FMT`. They take a pointer to a struct `v4l2_format` as argument, here the struct `v4l2_pix_format` named `pix` of its `fmt` union is used.

For more information about the V4L2 read interface see *Read/Write*.

Capturing using memory mapping

Applications can read from V4L devices by mapping buffers in device memory, or more often just buffers allocated in DMA-able system memory, into their address space. This avoids the data copying overhead of the read method. V4L2 supports memory mapping as well, with a few differences.

V4L	V4L2
	The image format must be selected before buffers are allocated, with the <code>VIDIOC_S_FMT</code> ioctl. When no format is selected the driver may use the last, possibly by another application requested format.
Applications cannot change the number of buffers. The it is built into the driver, unless it has a module option to change the number when the driver module is loaded.	The <code>ioctl VIDIOC_REQBUFS</code> ioctl allocates the desired number of buffers, this is a required step in the initialization sequence.
Drivers map all buffers as one contiguous range of memory. The <code>VIDIOCGMBUF</code> ioctl is available to query the number of buffers, the offset of each buffer from the start of the virtual file, and the overall amount of memory used, which can be used as arguments for the <code>mmap()</code> function.	Buffers are individually mapped. The offset and size of each buffer can be determined with the <code>ioctl VIDIOC_QUERYBUF</code> ioctl.
The <code>VIDIOCMCAPTURE</code> ioctl prepares a buffer for capturing. It also determines the image format for this buffer. The ioctl returns immediately, eventually with an <code>EAGAIN</code> error code if no video signal had been detected. When the driver supports more than one buffer applications can call the ioctl multiple times and thus have multiple outstanding capture requests. The <code>VIDIOCSYNC</code> ioctl suspends execution until a particular buffer has been filled.	Drivers maintain an incoming and outgoing queue. <code>ioctl VIDIOC_QBUF</code> , <code>VIDIOC_DQBUF</code> enqueues any empty buffer into the incoming queue. Filled buffers are dequeued from the outgoing queue with the <code>VIDIOC_DQBUF</code> ioctl. To wait until filled buffers become available this function, <code>select()</code> or <code>poll()</code> can be used. The <code>ioctl VIDIOC_STREAMON</code> , <code>VIDIOC_STREAMOFF</code> ioctl must be called once after enqueueing one or more buffers to start capturing. Its counterpart <code>VIDIOC_STREAMOFF</code> stops capturing and dequeues all buffers from both queues. Applications can query the signal status, if known, with the <code>ioctl VIDIOC_ENUMINPUT</code> ioctl.

For a more in-depth discussion of memory mapping and examples, see *Streaming I/O (Memory Mapping)*.

Reading Raw VBI Data

Originally the V4L API did not specify a raw VBI capture interface, only the device file `/dev/vbi` was reserved for this purpose. The only driver supporting this interface was the BTTV driver, de-facto defining the V4L VBI interface. Reading from the device yields a raw VBI image with the following parameters:

struct v4l2_vbi_format	V4L, BTTV driver
sampling_rate	28636363 Hz NTSC (or any other 525-line standard); 35468950 Hz PAL and SECAM (625-line standards)
offset	?
samples_per_line	2048
sample_format	V4L2_PIX_FMT_GREY. The last four bytes (a machine endianness integer) contain a frame counter.
start[]	10, 273 NTSC; 22, 335 PAL and SECAM
count[]	16, 16 ⁹
flags	0

Undocumented in the V4L specification, in Linux 2.3 the VIDIOCGVBIFMT and VIDIOCSVBIFMT ioctls using struct vbi_format were added to determine the VBI image parameters. These ioctls are only partially compatible with the V4L2 VBI interface specified in *Raw VBI Data Interface*.

An offset field does not exist, sample_format is supposed to be VIDEO_PALETTE_RAW, equivalent to V4L2_PIX_FMT_GREY. The remaining fields are probably equivalent to struct v4l2_vbi_format.

Apparently only the Zoran (ZR 36120) driver implements these ioctls. The semantics differ from those specified for V4L2 in two ways. The parameters are reset on open() and VIDIOCSVBIFMT always returns an EINVAL error code if the parameters are invalid.

Miscellaneous

V4L2 has no equivalent of the VIDIOCGUNIT ioctl. Applications can find the VBI device associated with a video capture device (or vice versa) by reopening the device and requesting VBI data. For details see *Opening and Closing Devices*.

No replacement exists for VIDIOCKEY, and the V4L functions for microcode programming. A new interface for MPEG compression and playback devices is documented in *Extended Controls API*.

3.2.6.2 Changes of the V4L2 API

Soon after the V4L API was added to the kernel it was criticised as too inflexible. In August 1998 Bill Dirks proposed a number of improvements and began to work on documentation, example drivers and applications. With the help of other volunteers this eventually became the V4L2 API, not just an extension but a replacement for the V4L API. However it took another four years and two stable kernel releases until the new API was finally accepted for inclusion into the kernel in its present form.

⁹ Old driver versions used different values, eventually the custom BTTV_VBISIZE ioctl was added to query the correct values.

Early Versions

1998-08-20: First version.

1998-08-27: The `select()` function was introduced.

1998-09-10: New video standard interface.

1998-09-18: The `VIDIOC_NONCAP` ioctl was replaced by the otherwise meaningless `O_TRUNC` `open()` flag, and the aliases `O_NONCAP` and `O_NOIO` were defined. Applications can set this flag if they intend to access controls only, as opposed to capture applications which need exclusive access. The `VIDEO_STD_XXX` identifiers are now ordinals instead of flags, and the `video_std_construct()` helper function takes id and transmission arguments.

1998-09-28: Revamped video standard. Made video controls individually enumerable.

1998-10-02: The `id` field was removed from struct `video_standard` and the color subcarrier fields were renamed. The *ioctl* `VIDIOC_QUERYSTD`, `VIDIOC_SUBDEV_QUERYSTD` ioctl was renamed to *ioctl* `VIDIOC_ENUMSTD`, `VIDIOC_SUBDEV_ENUMSTD`, `VIDIOC_G_INPUT` to *ioctl* `VIDIOC_ENUMINPUT`. A first draft of the Codec API was released.

1998-11-08: Many minor changes. Most symbols have been renamed. Some material changes to struct `v4l2_capability`.

1998-11-12: The read/write direction of some ioctls was misdefined.

1998-11-14: `V4L2_PIX_FMT_RGB24` changed to `V4L2_PIX_FMT_BGR24`, and `V4L2_PIX_FMT_RGB32` changed to `V4L2_PIX_FMT_BGR32`. Audio controls are now accessible with the *VIDIOC_G_CTRL* and *VIDIOC_S_CTRL* ioctls under names starting with `V4L2_CID_AUDIO`. The `V4L2_MAJOR` define was removed from `videodev.h` since it was only used once in the `videodev` kernel module. The YUV422 and YUV411 planar image formats were added.

1998-11-28: A few ioctl symbols changed. Interfaces for codecs and video output devices were added.

1999-01-14: A raw VBI capture interface was added.

1999-01-19: The `VIDIOC_NEXTBUF` ioctl was removed.

V4L2 Version 0.16 1999-01-31

1999-01-27: There is now one QBUF ioctl, `VIDIOC_QWBUF` and `VIDIOC_QRBUF` are gone. `VIDIOC_QBUF` takes a `v4l2_buffer` as a parameter. Added digital zoom (cropping) controls.

V4L2 Version 0.18 1999-03-16

Added a `v4l` to V4L2 ioctl compatibility layer to `videodev.c`. Driver writers, this changes how you implement your ioctl handler. See the Driver Writer's Guide. Added some more control id codes.

V4L2 Version 0.19 1999-06-05

1999-03-18: Fill in the category and catname fields of `v4l2_queryctrl` objects before passing them to the driver. Required a minor change to the `VIDIOC_QUERYCTRL` handlers in the sample drivers.

1999-03-31: Better compatibility for v4l memory capture ioctls. Requires changes to drivers to fully support new compatibility features, see Driver Writer's Guide and `v4l2cap.c`. Added new control IDs: `V4L2_CID_HFLIP`, `_VFLIP`. Changed `V4L2_PIX_FMT_YUV422P` to `_YUV422P`, and `_YUV411P` to `_YUV411P`.

1999-04-04: Added a few more control IDs.

1999-04-07: Added the button control type.

1999-05-02: Fixed a typo in `videodev.h`, and added the `V4L2_CTRL_FLAG_GRAYED` (later `V4L2_CTRL_FLAG_GRABBED`) flag.

1999-05-20: Definition of `VIDIOC_G_CTRL` was wrong causing a malfunction of this ioctl.

1999-06-05: Changed the value of `V4L2_CID_WHITENESS`.

V4L2 Version 0.20 (1999-09-10)

Version 0.20 introduced a number of changes which were *not backward compatible* with 0.19 and earlier versions. Purpose of these changes was to simplify the API, while making it more extensible and following common Linux driver API conventions.

1. Some typos in `V4L2_FMT_FLAG` symbols were fixed. `struct v4l2_clip` was changed for compatibility with v4l. (1999-08-30)
2. `V4L2_TUNER_SUB_LANG1` was added. (1999-09-05)
3. All `ioctl()` commands that used an integer argument now take a pointer to an integer. Where it makes sense, ioctls will return the actual new value in the integer pointed to by the argument, a common convention in the V4L2 API. The affected ioctls are: `VIDIOC_PREVIEW`, `VIDIOC_STREAMON`, `VIDIOC_STREAMOFF`, `VIDIOC_S_FREQ`, `VIDIOC_S_INPUT`, `VIDIOC_S_OUTPUT`, `VIDIOC_S_EFFECT`. For example

```
err = ioctl (fd, VIDIOC_XXX, V4L2_XXX);
```

becomes

```
int a = V4L2_XXX; err = ioctl(fd, VIDIOC_XXX, &a);
```

4. All the different get- and set-format commands were swept into one `VIDIOC_G_FMT` and `VIDIOC_S_FMT` ioctl taking a union and a type field selecting the union member as parameter. Purpose is to simplify the API by eliminating several ioctls and to allow new and driver private data streams without adding new ioctls.

This change obsoletes the following ioctls: `VIDIOC_S_INFMT`, `VIDIOC_G_INFMT`, `VIDIOC_S_OUTFMT`, `VIDIOC_G_OUTFMT`, `VIDIOC_S_VBIFMT` and `VIDIOC_G_VBIFMT`. The image format struct `v4l2_format` was renamed to `struct v4l2_pix_format`, while struct `v4l2_format` is now the envelopping structure for all format negotiations.

5. Similar to the changes above, the `VIDIOC_G_PARM` and `VIDIOC_S_PARM` ioctls were merged with `VIDIOC_G_OUTPARAM` and `VIDIOC_S_OUTPARAM`. A type field in the new struct `v4l2_streamparm` selects the respective union member.

This change obsoletes the `VIDIOC_G_OUTPARAM` and `VIDIOC_S_OUTPARAM` ioctls.

6. Control enumeration was simplified, and two new control flags were introduced and one dropped. The `catname` field was replaced by a `group` field.

Drivers can now flag unsupported and temporarily unavailable controls with `V4L2_CTRL_FLAG_DISABLED` and `V4L2_CTRL_FLAG_GRABBED` respectively. The `group` name indicates a possibly narrower classification than the `category`. In other words, there may be multiple groups within a category. Controls within a group would typically be drawn within a group box. Controls in different categories might have a greater separation, or may even appear in separate windows.

7. The struct `v4l2_buffer` `timestamp` was changed to a 64 bit integer, containing the sampling or output time of the frame in nanoseconds. Additionally timestamps will be in absolute system time, not starting from zero at the beginning of a stream. The data type name for timestamps is `stamp_t`, defined as a signed 64-bit integer. Output devices should not send a buffer out until the time in the `timestamp` field has arrived. I would like to follow SGI's lead, and adopt a multimedia timestamping system like their UST (Unadjusted System Time). See http://web.archive.org/web/*/http://reality.sgi.com/cpi-razzi_engr/lg/time/intro.html. UST uses timestamps that are 64-bit signed integers (not struct `timeval`'s) and given in nanosecond units. The UST clock starts at zero when the system is booted and runs continuously and uniformly. It takes a little over 292 years for UST to overflow. There is no way to set the UST clock. The regular Linux time-of-day clock can be changed periodically, which would cause errors if it were being used for timestamping a multimedia stream. A real UST style clock will require some support in the kernel that is not there yet. But in anticipation, I will change the `timestamp` field to a 64-bit integer, and I will change the `v4l2_masterclock_gettime()` function (used only by drivers) to return a 64-bit integer.
8. A `sequence` field was added to struct `v4l2_buffer`. The `sequence` field counts captured frames, it is ignored by output devices. When a capture driver drops a frame, the `sequence` number of that frame is skipped.

V4L2 Version 0.20 incremental changes

1999-12-23: In struct `v4l2_vbi_format` the `reserved1` field became `offset`. Previously drivers were required to clear the `reserved1` field.

2000-01-13: The `V4L2_FMT_FLAG_NOT_INTERLACED` flag was added.

2000-07-31: The `linux/poll.h` header is now included by `videodev.h` for compatibility with the original `videodev.h` file.

2000-11-20: `V4L2_TYPE_VBI_OUTPUT` and `V4L2_PIX_FMT_Y41P` were added.

2000-11-25: `V4L2_TYPE_VBI_INPUT` was added.

2000-12-04: A couple typos in symbol names were fixed.

2001-01-18: To avoid namespace conflicts the `fourcc` macro defined in the `videodev.h` header file was renamed to `v4l2_fourcc`.

2001-01-25: A possible driver-level compatibility problem between the `videodev.h` file in Linux 2.4.0 and the `videodev.h` file included in the `videodevX` patch was fixed. Users of an earlier version of `videodevX` on Linux 2.4.0 should recompile their V4L and V4L2 drivers.

2001-01-26: A possible kernel-level incompatibility between the `videodev.h` file in the `videodevX` patch and the `videodev.h` file in Linux 2.2.x with `devfs` patches applied was fixed.

2001-03-02: Certain V4L ioctls which pass data in both direction although they are defined with read-only parameter, did not work correctly through the backward compatibility layer. [Solution?]

2001-04-13: Big endian 16-bit RGB formats were added.

2001-09-17: New YUV formats and the `VIDIOC_G_FREQUENCY` and `VIDIOC_S_FREQUENCY` ioctls were added. (The old `VIDIOC_G_FREQ` and `VIDIOC_S_FREQ` ioctls did not take multiple tuners into account.)

2000-09-18: `V4L2_BUF_TYPE_VBI` was added. This may *break compatibility* as the `VIDIOC_G_FMT` and `VIDIOC_S_FMT` ioctls may fail now if the struct `v4l2_fmt` type field does not contain `V4L2_BUF_TYPE_VBI`. In the documentation of the struct `v4l2_vbi_format`, the offset field the ambiguous phrase “rising edge” was changed to “leading edge”.

V4L2 Version 0.20 2000-11-23

A number of changes were made to the raw VBI interface.

1. Figures clarifying the line numbering scheme were added to the V4L2 API specification. The `start[0]` and `start[1]` fields no longer count line numbers beginning at zero. Rationale: a) The previous definition was unclear. b) The `start[]` values are ordinal numbers. c) There is no point in inventing a new line numbering scheme. We now use line number as defined by ITU-R, period. Compatibility: Add one to the start values. Applications depending on the previous semantics may not function correctly.
2. The restriction “`count[0] > 0` and `count[1] > 0`” has been relaxed to “(`count[0] + count[1]`) > 0”. Rationale: Drivers may allocate resources at scan line granularity and some data services are transmitted only on the first field. The comment that both count values will usually be equal is misleading and pointless and has been removed. This change *breaks compatibility* with earlier versions: Drivers may return `EINVAL`, applications may not function correctly.
3. Drivers are again permitted to return negative (unknown) start values as proposed earlier. Why this feature was dropped is unclear. This change may *break compatibility* with applications depending on the start values being positive. The use of `EBUSY` and `EINVAL` error codes with the `VIDIOC_S_FMT` ioctl was clarified. The `EBUSY` error code was finally documented, and the `reserved2` field which was previously mentioned only in the `videodev.h` header file.
4. New buffer types `V4L2_TYPE_VBI_INPUT` and `V4L2_TYPE_VBI_OUTPUT` were added. The former is an alias for the old `V4L2_TYPE_VBI`, the latter was missing in the `videodev.h` file.

V4L2 Version 0.20 2002-07-25

Added sliced VBI interface proposal.

V4L2 in Linux 2.5.46, 2002-10

Around October-November 2002, prior to an announced feature freeze of Linux 2.5, the API was revised, drawing from experience with V4L2 0.20. This unnamed version was finally merged into Linux 2.5.46.

1. As specified in *Related Devices*, drivers must make related device functions available under all minor device numbers.
2. The `open()` function requires access mode `O_RDWR` regardless of the device type. All V4L2 drivers exchanging data with applications must support the `O_NONBLOCK` flag. The `O_NOIO` flag, a V4L2 symbol which aliased the meaningless `O_TRUNC` to indicate accesses without data exchange (panel applications) was dropped. Drivers must stay in “panel mode” until the application attempts to initiate a data exchange, see *Opening and Closing Devices*.
3. The struct `v4l2_capability` changed dramatically. Note that also the size of the structure changed, which is encoded in the `ioctl` request code, thus older V4L2 devices will respond with an `EINVAL` error code to the new `ioctl VIDIOC_QUERYCAP` `ioctl`.

There are new fields to identify the driver, a new RDS device function `V4L2_CAP_RDS_CAPTURE`, the `V4L2_CAP_AUDIO` flag indicates if the device has any audio connectors, another I/O capability `V4L2_CAP_ASYNCIO` can be flagged. In response to these changes the `type` field became a bit set and was merged into the `flags` field. `V4L2_FLAG_TUNER` was renamed to `V4L2_CAP_TUNER`, `V4L2_CAP_VIDEO_OVERLAY` replaced `V4L2_FLAG_PREVIEW` and `V4L2_CAP_VBI_CAPTURE` and `V4L2_CAP_VBI_OUTPUT` replaced `V4L2_FLAG_DATA_SERVICE`. `V4L2_FLAG_READ` and `V4L2_FLAG_WRITE` were merged into `V4L2_CAP_READWRITE`.

The redundant fields `inputs`, `outputs` and `audios` were removed. These properties can be determined as described in *Video Inputs and Outputs* and *Audio Inputs and Outputs*.

The somewhat volatile and therefore barely useful fields `maxwidth`, `maxheight`, `minwidth`, `minheight`, `maxframerate` were removed. This information is available as described in *Data Formats* and *Video Standards*.

`V4L2_FLAG_SELECT` was removed. We believe the `select()` function is important enough to require support of it in all V4L2 drivers exchanging data with applications. The redundant `V4L2_FLAG_MONOCHROME` flag was removed, this information is available as described in *Data Formats*.

4. In struct `v4l2_input` the `assoc_audio` field and the `capability` field and its only flag `V4L2_INPUT_CAP_AUDIO` was replaced by the new `audioset` field. Instead of linking one video input to one audio input this field reports all audio inputs this video input combines with.

New fields are `tuner` (reversing the former link from tuners to video inputs), `std` and `status`.

Accordingly struct `v4l2_output` lost its `capability` and `assoc_audio` fields. `audioset`, `modulator` and `std` were added instead.

5. The struct `v4l2_audio` field `audio` was renamed to `index`, for consistency with other structures. A new capability flag `V4L2_AUDCAP_STEREO` was added to indicate if the audio input in question supports stereo sound. `V4L2_AUDCAP_EFFECTS` and the corresponding `V4L2_AUDMODE` flags were removed. This can be easily implemented using controls. (However the same applies to AVL which is still there.)

Again for consistency the struct `v4l2_audioout` field `audio` was renamed to `index`.

6. The struct `v4l2_tuner` input field was replaced by an `index` field, permitting devices with multiple tuners. The link between video inputs and tuners is now reversed, inputs point to their tuner. The `std` substructure became a simple set (more about this below) and moved into struct `v4l2_input`. A `type` field was added.

Accordingly in struct `v4l2_modulator` the output was replaced by an `index` field.

In struct `v4l2_frequency` the `port` field was replaced by a `tuner` field containing the respective tuner or modulator index number. A `tuner` type field was added and the reserved field became larger for future extensions (satellite tuners in particular).

7. The idea of completely transparent video standards was dropped. Experience showed that applications must be able to work with video standards beyond presenting the user a menu. Instead of enumerating supported standards with an `ioctl` applications can now refer to standards by `v4l2_std_id` and symbols defined in the `videodev2.h` header file. For details see *Video Standards*. The `VIDIOC_G_STD` and `VIDIOC_S_STD` now take a pointer to this type as argument. `ioctl VIDIOC_QUERYSTD, VIDIOC_SUBDEV_QUERYSTD` was added to autodetect the received standard, if the hardware has this capability. In struct `v4l2_standard` an `index` field was added for `ioctl VIDIOC_ENUMSTD, VIDIOC_SUBDEV_ENUMSTD`. A `v4l2_std_id` field named `id` was added as machine readable identifier, also replacing the `transmission` field. The misleading `framerate` field was renamed to `frameperiod`. The now obsolete `colorstandard` information, originally needed to distinguish between variations of standards, were removed.

Struct `v4l2_enumstd` ceased to be. `ioctl VIDIOC_ENUMSTD, VIDIOC_SUBDEV_ENUMSTD` now takes a pointer to a struct `v4l2_standard` directly. The information which standards are supported by a particular video input or output moved into struct `v4l2_input` and struct `v4l2_output` fields named `std`, respectively.

8. The struct `v4l2_queryctrl` fields `category` and `group` did not catch on and/or were not implemented as expected and therefore removed.
9. The `VIDIOC_TRY_FMT` `ioctl` was added to negotiate data formats as with `VIDIOC_S_FMT`, but without the overhead of programming the hardware and regardless of I/O in progress.

In struct `v4l2_format` the `fmt` union was extended to contain struct `v4l2_window`. All image format negotiations are now possible with `VIDIOC_G_FMT, VIDIOC_S_FMT` and `VIDIOC_TRY_FMT`; `ioctl`. The `VIDIOC_G_WIN` and `VIDIOC_S_WIN` `ioctls` to prepare for a video overlay were removed. The `type` field changed to type enum `v4l2_buf_type` and the buffer type names changed as follows.

Old defines	enum v4l2_buf_type
V4L2_BUF_TYPE_CAPTURE	V4L2_BUF_TYPE_VIDEO_CAPTURE
V4L2_BUF_TYPE_CODECCIN	Omitted for now
V4L2_BUF_TYPE_CODECCOUT	Omitted for now
V4L2_BUF_TYPE_EFFECTSIN	Omitted for now
V4L2_BUF_TYPE_EFFECTSIN2	Omitted for now
V4L2_BUF_TYPE_EFFECTSOUT	Omitted for now
V4L2_BUF_TYPE_VIDEO0OUT	V4L2_BUF_TYPE_VIDEO_OUTPUT
-	V4L2_BUF_TYPE_VIDEO_OVERLAY
-	V4L2_BUF_TYPE_VBI_CAPTURE
-	V4L2_BUF_TYPE_VBI_OUTPUT
-	V4L2_BUF_TYPE_SLICED_VBI_CAPTURE
-	V4L2_BUF_TYPE_SLICED_VBI_OUTPUT
V4L2_BUF_TYPE_PRIVATE_BASE	V4L2_BUF_TYPE_PRIVATE (but this is deprecated)

10. In struct `v4l2_fmtdesc` a enum `v4l2_buf_type` field named `type` was added as in struct `v4l2_format`. The `VIDIOC_ENUM_FBUFFMT` ioctl is no longer needed and was removed. These calls can be replaced by `ioctl VIDIOC_ENUM_FMT` with type `V4L2_BUF_TYPE_VIDEO_OVERLAY`.
11. In `struct v4l2_pix_format` the `depth` field was removed, assuming applications which recognize the format by its four-character-code already know the color depth, and others do not care about it. The same rationale lead to the removal of the `V4L2_FMT_FLAG_COMPRESSED` flag. The `V4L2_FMT_FLAG_SWCONVECOMPRESSED` flag was removed because drivers are not supposed to convert images in kernel space. A user library of conversion functions should be provided instead. The `V4L2_FMT_FLAG_BYTESPERLINE` flag was redundant. Applications can set the `bytesperline` field to zero to get a reasonable default. Since the remaining flags were replaced as well, the `flags` field itself was removed.

The interlace flags were replaced by a `enum v4l2_field` value in a newly added `field` field.

Old flag	enum v4l2_field
V4L2_FMT_FLAG_NOT_INTERLACED	?
V4L2_FMT_FLAG_INTERLACED = V4L2_FMT_FLAG_COMBINED	V4L2_FIELD_INTERLACED
V4L2_FMT_FLAG_TOPFIELD = V4L2_FMT_FLAG_ODDFIELD	V4L2_FIELD_TOP
V4L2_FMT_FLAG_BOTFIELD = V4L2_FMT_FLAG_EVENFIELD	V4L2_FIELD_BOTTOM
-	V4L2_FIELD_SEQ_TB
-	V4L2_FIELD_SEQ_BT
-	V4L2_FIELD_ALTERNATE

The color space flags were replaced by a `enum v4l2_colorspace` value in a newly added `colorspace` field, where one of `V4L2_COLORSPACE_SMPTE170M`, `V4L2_COLORSPACE_BT878`, `V4L2_COLORSPACE_470_SYSTEM_M` or `V4L2_COLORSPACE_470_SYSTEM_BG` replaces `V4L2_FMT_CS_601YUV`.

12. In struct `v4l2_requestbuffers` the `type` field was properly defined as enum `v4l2_buf_type`. Buffer types changed as mentioned above. A new memory field of type enum `v4l2_memory` was added to distinguish between I/O methods using buffers allocated by the driver or the application. See *Input/Output* for details.

13. In struct `v4l2_buffer` the `type` field was properly defined as enum `v4l2_buf_type`. Buffer types changed as mentioned above. A `field` field of type *enum v4l2_field* was added to indicate if a buffer contains a top or bottom field. The old `field` flags were removed. Since no unadjusted system time clock was added to the kernel as planned, the `timestamp` field changed back from type `stamp_t`, an unsigned 64 bit integer expressing the sample time in nanoseconds, to struct `timeval`. With the addition of a second memory mapping method the `offset` field moved into union `m`, and a new memory field of type enum `v4l2_memory` was added to distinguish between I/O methods. See *Input/Output* for details.

The `V4L2_BUF_REQ_CONTIG` flag was used by the V4L compatibility layer, after changes to this code it was no longer needed. The `V4L2_BUF_ATTR_DEVICEMEM` flag would indicate if the buffer was indeed allocated in device memory rather than DMA-able system memory. It was barely useful and so was removed.

14. In struct `v4l2_framebuffer` the `base[3]` array anticipating double- and triple-buffering in off-screen video memory, however without defining a synchronization mechanism, was replaced by a single pointer. The `V4L2_FBUF_CAP_SCALEUP` and `V4L2_FBUF_CAP_SCALEDOWN` flags were removed. Applications can determine this capability more accurately using the new cropping and scaling interface. The `V4L2_FBUF_CAP_CLIPPING` flag was replaced by `V4L2_FBUF_CAP_LIST_CLIPPING` and `V4L2_FBUF_CAP_BITMAP_CLIPPING`.
15. In struct `v4l2_clip` the `x`, `y`, `width` and `height` field moved into a `c` substructure of type struct `v4l2_rect`. The `x` and `y` fields were renamed to `left` and `top`, i. e. offsets to a context dependent origin.
16. In struct `v4l2_window` the `x`, `y`, `width` and `height` field moved into a `w` substructure as above. A `field` field of type *enum v4l2_field* was added to distinguish between field and frame (interlaced) overlay.
17. The digital zoom interface, including struct `v4l2_zoomcap`, struct `v4l2_zoom`, `V4L2_ZOOM_NONCAP` and `V4L2_ZOOM_WHILESTREAMING` was replaced by a new cropping and scaling interface. The previously unused struct `v4l2_cropcap` and struct `v4l2_crop` were redefined for this purpose. See *Image Cropping, Insertion and Scaling - the CROP API* for details.
18. In struct `v4l2_vbi_format` the `SAMPLE_FORMAT` field now contains a four-character-code as used to identify video image formats and `V4L2_PIX_FMT_GREY` replaces the `V4L2_VBI_SF_UBYTE` define. The reserved field was extended.
19. In struct `v4l2_captureparm` the type of the `timeperframe` field changed from unsigned long to struct `v4l2_fract`. This allows the accurate expression of multiples of the NTSC-M frame rate $30000 / 1001$. A new field `readbuffers` was added to control the driver behaviour in read I/O mode.

Similar changes were made to struct `v4l2_outputparm`.

20. The struct `v4l2_performance` and `VIDIOC_G_PERF` ioctl were dropped. Except when using the *read/write I/O method*, which is limited anyway, this information is already available to applications.
21. The example transformation from RGB to YCbCr color space in the old V4L2 documentation was inaccurate, this has been corrected in *Image Formats*.

V4L2 2003-06-19

1. A new capability flag `V4L2_CAP_RADIO` was added for radio devices. Prior to this change radio devices would identify solely by having exactly one tuner whose type field reads `V4L2_TUNER_RADIO`.
2. An optional driver access priority mechanism was added, see *Application Priority* for details.
3. The audio input and output interface was found to be incomplete.

Previously the `VIDIOC_G_AUDIO` ioctl would enumerate the available audio inputs. An ioctl to determine the current audio input, if more than one combines with the current video input, did not exist. So `VIDIOC_G_AUDIO` was renamed to `VIDIOC_G_AUDIO_OLD`, this ioctl was removed on Kernel 2.6.39. The *ioctl* `VIDIOC_ENUMAUDIO` ioctl was added to enumerate audio inputs, while `VIDIOC_G_AUDIO` now reports the current audio input.

The same changes were made to `VIDIOC_G_AUDOUT` and `VIDIOC_ENUMAUDOUT`.

Until further the “videodev” module will automatically translate between the old and new ioctls, but drivers and applications must be updated to successfully compile again.

4. The *ioctl* `VIDIOC_OVERLAY` ioctl was incorrectly defined with write-read parameter. It was changed to write-only, while the write-read version was renamed to `VIDIOC_OVERLAY_OLD`. The old ioctl was removed on Kernel 2.6.39. Until further the “videodev” kernel module will automatically translate to the new version, so drivers must be recompiled, but not applications.
5. *Video Overlay Interface* incorrectly stated that clipping rectangles define regions where the video can be seen. Correct is that clipping rectangles define regions where *no* video shall be displayed and so the graphics surface can be seen.
6. The `VIDIOC_S_PARM` and `VIDIOC_S_CTRL` ioctls were defined with write-only parameter, inconsistent with other ioctls modifying their argument. They were changed to write-read, while a `_OLD` suffix was added to the write-only versions. The old ioctls were removed on Kernel 2.6.39. Drivers and applications assuming a constant parameter need an update.

V4L2 2003-11-05

1. In *RGB Formats* the following pixel formats were incorrectly transferred from Bill Dirks’ V4L2 specification. Descriptions below refer to bytes in memory, in ascending address order.

Symbol	In this document prior to revision 0.5	Corrected
<code>V4L2_PIX_FMT_RGB24</code>	B, G, R	R, G, B
<code>V4L2_PIX_FMT_BGR24</code>	R, G, B	B, G, R
<code>V4L2_PIX_FMT_RGB32</code>	B, G, R, X	R, G, B, X
<code>V4L2_PIX_FMT_BGR32</code>	R, G, B, X	B, G, R, X

The `V4L2_PIX_FMT_BGR24` example was always correct.

In *Image Properties* the mapping of the V4L `VIDEO_PALETTE_RGB24` and `VIDEO_PALETTE_RGB32` formats to V4L2 pixel formats was accordingly corrected.

2. Unrelated to the fixes above, drivers may still interpret some V4L2 RGB pixel formats differently. These issues have yet to be addressed, for details see *RGB Formats*.

V4L2 in Linux 2.6.6, 2004-05-09

1. The *ioctl VIDIOC_CROPCAP* ioctl was incorrectly defined with read-only parameter. It is now defined as write-read ioctl, while the read-only version was renamed to VIDIOC_CROPCAP_OLD. The old ioctl was removed on Kernel 2.6.39.

V4L2 in Linux 2.6.8

1. A new field input (former reserved[0]) was added to the struct v4l2_buffer. Purpose of this field is to alternate between video inputs (e. g. cameras) in step with the video capturing process. This function must be enabled with the new V4L2_BUF_FLAG_INPUT flag. The flags field is no longer read-only.

V4L2 spec erratum 2004-08-01

1. The return value of the *V4L2 open()* function was incorrectly documented.
2. Audio output ioctls end in -AUDOUT, not -AUDIOOUT.
3. In the Current Audio Input example the VIDIOC_G_AUDIO ioctl took the wrong argument.
4. The documentation of the *ioctl VIDIOC_QBUF, VIDIOC_DQBUF* and *VIDIOC_DQBUF* ioctls did not mention the struct v4l2_buffer memory field. It was also missing from examples. Also on the VIDIOC_DQBUF page the EIO error code was not documented.

V4L2 in Linux 2.6.14

1. A new sliced VBI interface was added. It is documented in *Sliced VBI Data Interface* and replaces the interface first proposed in V4L2 specification 0.8.

V4L2 in Linux 2.6.15

1. The *ioctl VIDIOC_LOG_STATUS* ioctl was added.
2. New video standards V4L2_STD_NTSC_443, V4L2_STD_SECAM_LC, V4L2_STD_SECAM_DK (a set of SECAM D, K and K1), and V4L2_STD_ATSC (a set of V4L2_STD_ATSC_8_VSB and V4L2_STD_ATSC_16_VSB) were defined. Note the V4L2_STD_525_60 set now includes V4L2_STD_NTSC_443. See also *typedef v4l2_std_id*.
3. The VIDIOC_G_COMP and VIDIOC_S_COMP ioctl were renamed to VIDIOC_G_MPEGCOMP and VIDIOC_S_MPEGCOMP respectively. Their argument was replaced by a struct v4l2_mpeg_compression pointer. (The VIDIOC_G_MPEGCOMP and VIDIOC_S_MPEGCOMP ioctls where removed in Linux 2.6.25.)

V4L2 spec erratum 2005-11-27

The capture example in *Video Capture Example* called the `VIDIOC_S_CROP` ioctl without checking if cropping is supported. In the video standard selection example in *Video Standards* the `VIDIOC_S_STD` call used the wrong argument type.

V4L2 spec erratum 2006-01-10

1. The `V4L2_IN_ST_COLOR_KILL` flag in struct `v4l2_input` not only indicates if the color killer is enabled, but also if it is active. (The color killer disables color decoding when it detects no color in the video signal to improve the image quality.)
2. `VIDIOC_S_PARM` is a write-read ioctl, not write-only as stated on its reference page. The ioctl changed in 2003 as noted above.

V4L2 spec erratum 2006-02-03

1. In struct `v4l2_captureparm` and struct `v4l2_outputparm` the `timeperframe` field gives the time in seconds, not microseconds.

V4L2 spec erratum 2006-02-04

1. The `clips` field in struct `v4l2_window` must point to an array of struct `v4l2_clip`, not a linked list, because drivers ignore the struct `v4l2_clip`. `next` pointer.

V4L2 in Linux 2.6.17

1. New video standard macros were added: `V4L2_STD_NTSC_M_KR` (NTSC M South Korea), and the sets `V4L2_STD_MN`, `V4L2_STD_B`, `V4L2_STD_GH` and `V4L2_STD_DK`. The `V4L2_STD_NTSC` and `V4L2_STD_SECAM` sets now include `V4L2_STD_NTSC_M_KR` and `V4L2_STD_SECAM_LC` respectively.
2. A new `V4L2_TUNER_MODE_LANG1_LANG2` was defined to record both languages of a bilingual program. The use of `V4L2_TUNER_MODE_STEREO` for this purpose is deprecated now. See the `VIDIOC_G_TUNER` section for details.

V4L2 spec erratum 2006-09-23 (Draft 0.15)

1. In various places `V4L2_BUF_TYPE_SLICED_VBI_CAPTURE` and `V4L2_BUF_TYPE_SLICED_VBI_OUTPUT` of the sliced VBI interface were not mentioned along with other buffer types.
2. In `VIDIOC_G_AUDIO` it was clarified that the struct `v4l2_audio` mode field is a flags field.
3. ioctl `VIDIOC_QUERYCAP` did not mention the sliced VBI and radio capability flags.
4. In `VIDIOC_G_FREQUENCY` it was clarified that applications must initialize the tuner type field of struct `v4l2_frequency` before calling `VIDIOC_S_FREQUENCY`.

5. The reserved array in struct `v4l2_requestbuffers` has 2 elements, not 32.
6. In *Video Output Interface* and *Raw VBI Data Interface* the device file names `/dev/vout` which never caught on were replaced by `/dev/video`.
7. With Linux 2.6.15 the possible range for VBI device minor numbers was extended from 224-239 to 224-255. Accordingly device file names `/dev/vbi0` to `/dev/vbi31` are possible now.

V4L2 in Linux 2.6.18

1. New ioctls `VIDIOC_G_EXT_CTRL`s, `VIDIOC_S_EXT_CTRL`s and `VIDIOC_TRY_EXT_CTRL`s were added, a flag to skip unsupported controls with ioctls `VIDIOC_QUERYCTRL`, `VIDIOC_QUERY_EXT_CTRL` and `VIDIOC_QUERYMENU`, new control types `V4L2_CTRL_TYPE_INTEGER64` and `V4L2_CTRL_TYPE_CTRL_CLASS` (enum `v4l2_ctrl_type`), and new control flags `V4L2_CTRL_FLAG_READ_ONLY`, `V4L2_CTRL_FLAG_UPDATE`, `V4L2_CTRL_FLAG_INACTIVE` and `V4L2_CTRL_FLAG_SLIDER` (*Control Flags*). See *Extended Controls API* for details.

V4L2 in Linux 2.6.19

1. In struct `v4l2_sliced_vbi_cap` a buffer type field was added replacing a reserved field. Note on architectures where the size of enum types differs from int types the size of the structure changed. The `VIDIOC_G_SLICED_VBI_CAP` ioctl was redefined from being read-only to write-read. Applications must initialize the type field and clear the reserved fields now. These changes may *break the compatibility* with older drivers and applications.
2. The ioctls `ioctl VIDIOC_ENUM_FRAMESIZES` and `ioctl VIDIOC_ENUM_FRAMEINTERVALS` were added.
3. A new pixel format `V4L2_PIX_FMT_RGB444` (*RGB Formats*) was added.

V4L2 spec erratum 2006-10-12 (Draft 0.17)

1. `V4L2_PIX_FMT_HM12` (*Reserved Image Formats*) is a YUV 4:2:0, not 4:2:2 format.

V4L2 in Linux 2.6.21

1. The `videodev2.h` header file is now dual licensed under GNU General Public License version two or later, and under a 3-clause BSD-style license.

V4L2 in Linux 2.6.22

1. Two new field orders `V4L2_FIELD_INTERLACED_TB` and `V4L2_FIELD_INTERLACED_BT` were added. See *enum v4l2_field* for details.
2. Three new clipping/blending methods with a global or straight or inverted local alpha value were added to the video overlay interface. See the description of the *VIDIOC_G_FBUF* and *VIDIOC_S_FBUF* ioctls for details.

A new `global_alpha` field was added to struct `v4l2_window`, extending the structure. This may **break compatibility** with applications using a struct `v4l2_window` directly. However the *VIDIOC_G/S/TRY_FMT* ioctls, which take a pointer to a struct `v4l2_format` parent structure with padding bytes at the end, are not affected.

3. The format of the chromakey field in struct `v4l2_window` changed from “host order RGB32” to a pixel value in the same format as the framebuffer. This may **break compatibility** with existing applications. Drivers supporting the “host order RGB32” format are not known.

V4L2 in Linux 2.6.24

1. The pixel formats `V4L2_PIX_FMT_PAL8`, `V4L2_PIX_FMT_YUV444`, `V4L2_PIX_FMT_YUV555`, `V4L2_PIX_FMT_YUV565` and `V4L2_PIX_FMT_YUV32` were added.

V4L2 in Linux 2.6.25

1. The pixel formats *V4L2_PIX_FMT_Y16* and *V4L2_PIX_FMT_SBGGR16* were added.
2. New *controls* `V4L2_CID_POWER_LINE_FREQUENCY`, `V4L2_CID_HUE_AUTO`, `V4L2_CID_WHITE_BALANCE_TEMPERATURE`, `V4L2_CID_SHARPNESS` and `V4L2_CID_BACKLIGHT_COMPENSATION` were added. The controls `V4L2_CID_BLACK_Level`, `V4L2_CID_WHITENESS`, `V4L2_CID_HCENTER` and `V4L2_CID_VCENTER` were deprecated.
3. A *Camera controls class* was added, with the new controls `V4L2_CID_EXPOSURE_AUTO`, `V4L2_CID_EXPOSURE_ABSOLUTE`, `V4L2_CID_EXPOSURE_AUTO_PRIORITY`, `V4L2_CID_PAN_RELATIVE`, `V4L2_CID_TILT_RELATIVE`, `V4L2_CID_PAN_RESET`, `V4L2_CID_TILT_RESET`, `V4L2_CID_PAN_ABSOLUTE`, `V4L2_CID_TILT_ABSOLUTE`, `V4L2_CID_FOCUS_ABSOLUTE`, `V4L2_CID_FOCUS_RELATIVE` and `V4L2_CID_FOCUS_AUTO`.
4. The *VIDIOC_G_MPEGCOMP* and *VIDIOC_S_MPEGCOMP* ioctls, which were superseded by the *extended controls* interface in Linux 2.6.18, were finally removed from the `videodev2.h` header file.

V4L2 in Linux 2.6.26

1. The pixel formats `V4L2_PIX_FMT_Y16` and `V4L2_PIX_FMT_SBGGR16` were added.
2. Added user controls `V4L2_CID_CHROMA_AGC` and `V4L2_CID_COLOR_KILLER`.

V4L2 in Linux 2.6.27

1. The `ioctl VIDIOC_S_HW_FREQ_SEEK` ioctl and the `V4L2_CAP_HW_FREQ_SEEK` capability were added.
2. The pixel formats `V4L2_PIX_FMT_YVYU`, `V4L2_PIX_FMT_PCA501`, `V4L2_PIX_FMT_PCA505`, `V4L2_PIX_FMT_PCA508`, `V4L2_PIX_FMT_PCA561`, `V4L2_PIX_FMT_SGBRG8`, `V4L2_PIX_FMT_PAC207` and `V4L2_PIX_FMT_PJPG` were added.

V4L2 in Linux 2.6.28

1. Added `V4L2_MPEG_AUDIO_ENCODING_AAC` and `V4L2_MPEG_AUDIO_ENCODING_AC3` MPEG audio encodings.
2. Added `V4L2_MPEG_VIDEO_ENCODING_MPEG_4_AVC` MPEG video encoding.
3. The pixel formats `V4L2_PIX_FMT_SGRBG10` and `V4L2_PIX_FMT_SGRBG10DPCM8` were added.

V4L2 in Linux 2.6.29

1. The `VIDIOC_G_CHIP_IDENT` ioctl was renamed to `VIDIOC_G_CHIP_IDENT_OLD` and `VIDIOC_DBG_G_CHIP_IDENT` was introduced in its place. The old struct `v4l2_chip_ident` was renamed to struct `v4l2_chip_ident_old`.
2. The pixel formats `V4L2_PIX_FMT_VYUY`, `V4L2_PIX_FMT_NV16` and `V4L2_PIX_FMT_NV61` were added.
3. Added camera controls `V4L2_CID_ZOOM_ABSOLUTE`, `V4L2_CID_ZOOM_RELATIVE`, `V4L2_CID_ZOOM_CONTINUOUS` and `V4L2_CID_PRIVACY`.

V4L2 in Linux 2.6.30

1. New control flag `V4L2_CTRL_FLAG_WRITE_ONLY` was added.
2. New control `V4L2_CID_COLORFX` was added.

V4L2 in Linux 2.6.32

1. In order to be easier to compare a V4L2 API and a kernel version, now V4L2 API is numbered using the Linux Kernel version numeration.
2. Finalized the RDS capture API. See [RDS Interface](#) for more information.
3. Added new capabilities for modulators and RDS encoders.
4. Add description for `libv4l` API.
5. Added support for string controls via new type `V4L2_CTRL_TYPE_STRING`.
6. Added `V4L2_CID_BAND_STOP_FILTER` documentation.
7. Added FM Modulator (FM TX) Extended Control Class: `V4L2_CTRL_CLASS_FM_TX` and their Control IDs.

8. Added FM Receiver (FM RX) Extended Control Class: `V4L2_CTRL_CLASS_FM_RX` and their Control IDs.
9. Added Remote Controller chapter, describing the default Remote Controller mapping for media devices.

V4L2 in Linux 2.6.33

1. Added support for Digital Video timings in order to support HDTV receivers and transmitters.

V4L2 in Linux 2.6.34

1. Added `V4L2_CID_IRIS_ABSOLUTE` and `V4L2_CID_IRIS_RELATIVE` controls to the *Camera controls class*.

V4L2 in Linux 2.6.37

1. Remove the `vtx` (videotext/teletext) API. This API was no longer used and no hardware exists to verify the API. Nor were any userspace applications found that used it. It was originally scheduled for removal in 2.6.35.

V4L2 in Linux 2.6.39

1. The old `VIDIOC*_OLD` symbols and V4L1 support were removed.
2. Multi-planar API added. Does not affect the compatibility of current drivers and applications. See *multi-planar API* for details.

V4L2 in Linux 3.1

1. `VIDIOC_QUERYCAP` now returns a per-subsystem version instead of a per-driver one. Standardize an error code for invalid ioctl.
Added `V4L2_CTRL_TYPE_BITMASK`.

V4L2 in Linux 3.2

1. `V4L2_CTRL_FLAG_VOLATILE` was added to signal volatile controls to userspace.
2. Add selection API for extended control over cropping and composing. Does not affect the compatibility of current drivers and applications. See *selection API* for details.

V4L2 in Linux 3.3

1. Added V4L2_CID_ALPHA_COMPONENT control to the *User controls class*.
2. Added the device_caps field to struct v4l2_capabilities and added the new V4L2_CAP_DEVICE_CAPS capability.

V4L2 in Linux 3.4

1. Added *JPEG compression control class*.
2. Extended the DV Timings API: *ioctl VIDIOC_ENUM_DV_TIMINGS*, *VIDIOC_SUBDEV_ENUM_DV_TIMINGS*, *ioctl VIDIOC_QUERY_DV_TIMINGS* and *ioctl VIDIOC_DV_TIMINGS_CAP*, *VIDIOC_SUBDEV_DV_TIMINGS_CAP*.

V4L2 in Linux 3.5

1. Added integer menus, the new type will be V4L2_CTRL_TYPE_INTEGER_MENU.
2. Added selection API for V4L2 subdev interface: *ioctl VIDIOC_SUBDEV_G_SELECTION*, *VIDIOC_SUBDEV_S_SELECTION* and *VIDIOC_SUBDEV_S_SELECTION*.
3. Added V4L2_COLORFX_ANTIQUA, V4L2_COLORFX_ART_FREEZE, V4L2_COLORFX_AQUA, V4L2_COLORFX_SILHOUETTE, V4L2_COLORFX_SOLARIZATION, V4L2_COLORFX_VIVID and V4L2_COLORFX_ARBITRARY_CBCR menu items to the V4L2_CID_COLORFX control.
4. Added V4L2_CID_COLORFX_CBCR control.
5. Added camera controls V4L2_CID_AUTO_EXPOSURE_BIAS, V4L2_CID_AUTO_N_PRESET_WHITE_BALANCE, V4L2_CID_IMAGE_STABILIZATION, V4L2_CID_ISO_SENSITIVITY, V4L2_CID_ISO_SENSITIVITY_AUTO, V4L2_CID_EXPOSURE_METERING, V4L2_CID_SCENE_MODE, V4L2_CID_3A_LOCK, V4L2_CID_AUTO_FOCUS_START, V4L2_CID_AUTO_FOCUS_STOP, V4L2_CID_AUTO_FOCUS_STATUS and V4L2_CID_AUTO_FOCUS_RANGE.

V4L2 in Linux 3.6

1. Replaced input in struct v4l2_buffer by reserved2 and removed V4L2_BUF_FLAG_INPUT.
2. Added V4L2_CAP_VIDEO_M2M and V4L2_CAP_VIDEO_M2M_MPLANE capabilities.
3. Added support for frequency band enumerations: *ioctl VIDIOC_ENUM_FREQ_BANDS*.

V4L2 in Linux 3.9

1. Added timestamp types to flags field in struct `v4l2_buffer`. See *Buffer Flags*.
2. Added `V4L2_EVENT_CTRL_CH_RANGE` control event changes flag. See *Control Changes*.

V4L2 in Linux 3.10

1. Removed obsolete and unused `DV_PRESET` ioctls `VIDIOC_G_DV_PRESET`, `VIDIOC_S_DV_PRESET`, `VIDIOC_QUERY_DV_PRESET` and `VIDIOC_ENUM_DV_PRESET`. Remove the related `v4l2_input/output` capability flags `V4L2_IN_CAP_PRESETS` and `V4L2_OUT_CAP_PRESETS`.
2. Added new debugging ioctl *ioctl* `VIDIOC_DBG_G_CHIP_INFO`.

V4L2 in Linux 3.11

1. Remove obsolete `VIDIOC_DBG_G_CHIP_IDENT` ioctl.

V4L2 in Linux 3.14

1. In struct `v4l2_rect`, the type of width and height fields changed from `_s32` to `_u32`.

V4L2 in Linux 3.15

1. Added Software Defined Radio (SDR) Interface.

V4L2 in Linux 3.16

1. Added event `V4L2_EVENT_SOURCE_CHANGE`.

V4L2 in Linux 3.17

1. Extended *struct* `v4l2_pix_format`. Added format flags.
2. Added compound control types and `VIDIOC_QUERY_EXT_CTRL`.

V4L2 in Linux 3.18

1. Added `V4L2_CID_PAN_SPEED` and `V4L2_CID_TILT_SPEED` camera controls.

V4L2 in Linux 3.19

1. Rewrote `Colorspace` chapter, added new `enum v4l2_ycbcr_encoding` and `enum v4l2_quantization` fields to `struct v4l2_pix_format`, `struct v4l2_pix_format_mplane` and `struct v4l2_mbus_framefmt`.

V4L2 in Linux 4.4

1. Renamed `V4L2_TUNER_ADC` to `V4L2_TUNER_SDR`. The use of `V4L2_TUNER_ADC` is deprecated now.
2. Added `V4L2_CID_RF_TUNER_RF_GAIN` RF Tuner control.
3. Added transmitter support for Software Defined Radio (SDR) Interface.

Relation of V4L2 to other Linux multimedia APIs

X Video Extension

The X Video Extension (abbreviated XVideo or just Xv) is an extension of the X Window system, implemented for example by the XFree86 project. Its scope is similar to V4L2, an API to video capture and output devices for X clients. Xv allows applications to display live video in a window, send window contents to a TV output, and capture or output still images in XPixmap¹. With their implementation XFree86 makes the extension available across many operating systems and architectures.

Because the driver is embedded into the X server Xv has a number of advantages over the V4L2 *video overlay interface*. The driver can easily determine the overlay target, i. e. visible graphics memory or off-screen buffers for a destructive overlay. It can program the RAMDAC for a non-destructive overlay, scaling or color-keying, or the clipping functions of the video capture hardware, always in sync with drawing operations or windows moving or changing their stacking order.

To combine the advantages of Xv and V4L a special Xv driver exists in XFree86 and XOrg, just programming any overlay capable Video4Linux device it finds. To enable it `/etc/X11/XF86Config` must contain these lines:

```
Section "Module"
    Load "v4l"
EndSection
```

As of XFree86 4.2 this driver still supports only V4L ioctls, however it should work just fine with all V4L2 devices through the V4L2 backward-compatibility layer. Since V4L2 permits multiple opens it is possible (if supported by the V4L2 driver) to capture video while an X client requested video overlay. Restrictions of simultaneous capturing and overlay are discussed in *Video Overlay Interface* apply.

Only marginally related to V4L2, XFree86 extended Xv to support hardware YUV to RGB conversion and scaling for faster video playback, and added an interface to MPEG-2 decoding hardware. This API is useful to display images captured with V4L2 devices.

¹ This is not implemented in XFree86.

Digital Video

V4L2 does not support digital terrestrial, cable or satellite broadcast. A separate project aiming at digital receivers exists. You can find its homepage at <https://linuxtv.org>. The Linux DVB API has no connection to the V4L2 API except that drivers for hybrid hardware may support both.

Audio Interfaces

[to do - OSS/ALSA]

Experimental API Elements

The following V4L2 API elements are currently experimental and may change in the future.

- `ioctl VIDIOC_DBG_G_REGISTER, VIDIOC_DBG_S_REGISTER` and `VIDIOC_DBG_S_REGISTER` ioctls.
- `ioctl VIDIOC_DBG_G_CHIP_INFO` ioctl.

Obsolete API Elements

The following V4L2 API elements were superseded by new interfaces and should not be implemented in new drivers.

- `VIDIOC_G_MPEGCOMP` and `VIDIOC_S_MPEGCOMP` ioctls. Use Extended Controls, *Extended Controls API*.
- `VIDIOC_G_DV_PRESET, VIDIOC_S_DV_PRESET, VIDIOC_ENUM_DV_PRESETS` and `VIDIOC_QUERY_DV_PRESET` ioctls. Use the DV Timings API (*Digital Video (DV) Timings*).
- `VIDIOC_SUBDEV_G_CROP` and `VIDIOC_SUBDEV_S_CROP` ioctls. Use `VIDIOC_SUBDEV_G_SELECTION` and `VIDIOC_SUBDEV_S_SELECTION, ioctl VIDIOC_SUBDEV_G_SELECTION, VIDIOC_SUBDEV_S_SELECTION`.

3.2.7 Function Reference

3.2.7.1 V4L2 close()

Name

v4l2-close - Close a V4L2 device

Synopsis

```
#include <unistd.h>
```

```
int close(int fd)
```

Arguments

fd File descriptor returned by `open()`.

Description

Closes the device. Any I/O in progress is terminated and resources associated with the file descriptor are freed. However data format parameters, current input or output, control values or other properties remain unchanged.

Return Value

The function returns 0 on success, -1 on failure and the `errno` is set appropriately. Possible error codes:

EBADF `fd` is not a valid open file descriptor.

3.2.7.2 V4L2 ioctl()

Name

v4l2-ioctl - Program a V4L2 device

Synopsis

```
#include <sys/ioctl.h>
```

```
int ioctl(int fd, int request, void *argp)
```

Arguments

fd File descriptor returned by `open()`.

request V4L2 ioctl request code as defined in the `videodev2.h` header file, for example `VIDIOC_QUERYCAP`.

argp Pointer to a function parameter, usually a structure.

Description

The *ioctl()* function is used to program V4L2 devices. The argument *fd* must be an open file descriptor. An *ioctl* request has encoded in it whether the argument is an input, output or read/write parameter, and the size of the argument *argp* in bytes. Macros and defines specifying V4L2 *ioctl* requests are located in the *videodev2.h* header file. Applications should use their own copy, not include the version in the kernel sources on the system they compile on. All V4L2 *ioctl* requests, their respective function and parameters are specified in *Function Reference*.

Return Value

On success 0 is returned, on error -1 and the *errno* variable is set appropriately. The generic error codes are described at the *Generic Error Codes* chapter.

When an *ioctl* that takes an output or read/write parameter fails, the parameter remains unmodified.

3.2.7.3 ioctl VIDIOC_CREATE_BUFS

Name

VIDIOC_CREATE_BUFS - Create buffers for Memory Mapped or User Pointer or DMA Buffer I/O

Synopsis

VIDIOC_CREATE_BUFS

```
int ioctl(int fd, VIDIOC_CREATE_BUFS, struct v4l2_create_buffers *argp)
```

Arguments

fd File descriptor returned by *open()*.

argp Pointer to struct *v4l2_create_buffers*.

Description

This *ioctl* is used to create buffers for *memory mapped* or *user pointer* or *DMA buffer* I/O. It can be used as an alternative or in addition to the *ioctl VIDIOC_REQBUFS* *ioctl*, when a tighter control over buffers is required. This *ioctl* can be called multiple times to create buffers of different sizes.

To allocate the device buffers applications must initialize the relevant fields of the struct *v4l2_create_buffers* structure. The *count* field must be set to the number of requested buffers, the *memory* field specifies the requested I/O method and the *reserved* array must be zeroed.

The format field specifies the image format that the buffers must be able to handle. The application has to fill in this struct `v4l2_format`. Usually this will be done using the `VIDIOC_TRY_FMT` or `VIDIOC_G_FMT` ioctls to ensure that the requested format is supported by the driver. Based on the format's `type` field the requested buffer size (for single-planar) or plane sizes (for multi-planar formats) will be used for the allocated buffers. The driver may return an error if the size(s) are not supported by the hardware (usually because they are too small).

The buffers created by this ioctl will have as minimum size the size defined by the `format.pix.sizeimage` field (or the corresponding fields for other format types). Usually if the `format.pix.sizeimage` field is less than the minimum required for the given format, then an error will be returned since drivers will typically not allow this. If it is larger, then the value will be used as-is. In other words, the driver may reject the requested size, but if it is accepted the driver will use it unchanged.

When the ioctl is called with a pointer to this structure the driver will attempt to allocate up to the requested number of buffers and store the actual number allocated and the starting index in the `count` and the `index` fields respectively. On return `count` can be smaller than the number requested.

v4l2_create_buffers

Table 131: struct v4l2_create_buffers

__u32	index	The starting buffer index, returned by the driver.
__u32	count	The number of buffers requested or granted. If count == 0, then <i>ioctl VIDIOC_CREATE_BUFS</i> will set index to the current number of created buffers, and it will check the validity of memory and format.type. If those are invalid -1 is returned and errno is set to EINVAL error code, otherwise <i>ioctl VIDIOC_CREATE_BUFS</i> returns 0. It will never set errno to EBUSY error code in this particular case.
__u32	memory	Applications set this field to V4L2_MEMORY_MMAP, V4L2_MEMORY_DMABUF or V4L2_MEMORY_USERPTR. See v4l2_memory
struct v4l2_format	format	Filled in by the application, preserved by the driver.
__u32	capabilities	Set by the driver. If 0, then the driver doesn't support capabilities. In that case all you know is that the driver is guaranteed to support V4L2_MEMORY_MMAP and <i>might</i> support other v4l2_memory types. It will not support any other capabilities. See here for a list of the capabilities. If you want to just query the capabilities without making any other changes, then set count to 0, memory to V4L2_MEMORY_MMAP and format.type to the buffer type.
__u32	flags	Specifies additional buffer management attributes. See Memory Consistency Flags .
__u32	reserved[6]	A place holder for future extensions. Drivers and applications must set the array to zero.

Return Value

On success 0 is returned, on error -1 and the errno variable is set appropriately. The generic error codes are described at the [Generic Error Codes](#) chapter.

ENOMEM No memory to allocate buffers for *memory mapped* I/O.

EINVAL The buffer type (format.type field), requested I/O method (memory) or format (format field) is not valid.

3.2.7.4 ioctl VIDIOC_CROPCAP

Name

VIDIOC_CROPCAP - Information about the video cropping and scaling abilities

Synopsis

VIDIOC_CROPCAP

```
int ioctl(int fd, VIDIOC_CROPCAP, struct v4l2_cropcap *argp)
```

Arguments

fd File descriptor returned by `open()`.

argp Pointer to struct `v4l2_cropcap`.

Description

Applications use this function to query the cropping limits, the pixel aspect of images and to calculate scale factors. They set the `type` field of a `v4l2_cropcap` structure to the respective buffer (stream) type and call the *ioctl VIDIOC_CROPCAP* ioctl with a pointer to this structure. Drivers fill the rest of the structure. The results are constant except when switching the video standard. Remember this switch can occur implicit when switching the video input or output.

This ioctl must be implemented for video capture or output devices that support cropping and/or scaling and/or have non-square pixels, and for overlay devices.

v4l2_cropcap

Table 132: struct v4l2_cropcap

__u32	type	Type of the data stream, set by the application. Only these types are valid here: V4L2_BUF_TYPE_VIDEO_CAPTURE, V4L2_BUF_TYPE_VIDEO_CAPTURE_MPLANE, V4L2_BUF_TYPE_VIDEO_OUTPUT, V4L2_BUF_TYPE_VIDEO_OUTPUT_MPLANE and V4L2_BUF_TYPE_VIDEO_OVERLAY. See <code>v4l2_buf_type</code> and the note below.
struct <i>v4l2_rect</i>	bounds	Defines the window within capturing or output is possible, this may exclude for example the horizontal and vertical blanking areas. The cropping rectangle cannot exceed these limits. Width and height are defined in pixels, the driver writer is free to choose origin and units of the coordinate system in the analog domain.
struct <i>v4l2_rect</i>	defrect	Default cropping rectangle, it shall cover the “whole picture”. Assuming pixel aspect 1/1 this could be for example a 640 × 480 rectangle for NTSC, a 768 × 576 rectangle for PAL and SECAM centered over the active picture area. The same co-ordinate system as for bounds is used.
struct <i>v4l2_fract</i>	pixelaspect	This is the pixel aspect (y / x) when no scaling is applied, the ratio of the actual sampling frequency and the frequency required to get square pixels. When cropping coordinates refer to square pixels, the driver sets pixelaspect to 1/1. Other common values are 54/59 for PAL and SECAM, 11/10 for NTSC sampled according to [ITU BT.601] .

Note: Unfortunately in the case of multiplanar buffer types (V4L2_BUF_TYPE_VIDEO_CAPTURE_MPLANE and V4L2_BUF_TYPE_VIDEO_OUTPUT_MPLANE) this API was messed up with regards to how the `v4l2_cropcap` type field should be filled in. Some drivers only accepted the `_MPLANE` buffer type while other drivers only accepted a non-multiplanar buffer type (i.e. without the `_MPLANE` at the end).

Starting with kernel 4.13 both variations are allowed.

Table 133: struct v4l2_rect

__s32	left	Horizontal offset of the top, left corner of the rectangle, in pixels.
__s32	top	Vertical offset of the top, left corner of the rectangle, in pixels.
__u32	width	Width of the rectangle, in pixels.
__u32	height	Height of the rectangle, in pixels.

Return Value

On success 0 is returned, on error -1 and the `errno` variable is set appropriately. The generic error codes are described at the *Generic Error Codes* chapter.

EINVAL The struct `v4l2_cropcap` type is invalid.

ENODATA Cropping is not supported for this input or output.

3.2.7.5 ioctl VIDIOC_DBG_G_CHIP_INFO

Name

VIDIOC_DBG_G_CHIP_INFO - Identify the chips on a TV card

Synopsis

VIDIOC_DBG_G_CHIP_INFO

```
int ioctl(int fd, VIDIOC_DBG_G_CHIP_INFO, struct v4l2_dbg_chip_info *argp)
```

Arguments

fd File descriptor returned by `open()`.

argp Pointer to struct `v4l2_dbg_chip_info`.

Description

Note: This is an *Experimental API Elements* interface and may change in the future.

For driver debugging purposes this `ioctl` allows test applications to query the driver about the chips present on the TV card. Regular applications must not use it. When you found a chip specific bug, please contact the linux-media mailing list (<https://linuxtv.org/lists.php>) so it can be fixed.

Additionally the Linux kernel must be compiled with the `CONFIG_VIDEO_ADV_DEBUG` option to enable this `ioctl`.

To query the driver applications must initialize the `match.type` and `match.addr` or `match.name` fields of a struct `v4l2_dbg_chip_info` and call `ioctl VIDIOC_DBG_G_CHIP_INFO` with a pointer to this structure. On success the driver stores information about the selected chip in the `name` and `flags` fields.

When `match.type` is `V4L2_CHIP_MATCH_BRIDGE`, `match.addr` selects the `n`th bridge ‘chip’ on the TV card. You can enumerate all chips by starting at zero and incrementing `match.addr` by one until `ioctl VIDIOC_DBG_G_CHIP_INFO` fails with an `EINVAL` error code. The number zero always selects the bridge chip itself, e. g. the chip connected to the PCI or USB bus. Non-zero numbers identify specific parts of the bridge chip such as an AC97 register block.

When `match.type` is `V4L2_CHIP_MATCH_SUBDEV`, `match.addr` selects the `n`th sub-device. This allows you to enumerate over all sub-devices.

On success, the `name` field will contain a chip name and the `flags` field will contain `V4L2_CHIP_FL_READABLE` if the driver supports reading registers from the device or `V4L2_CHIP_FL_WRITABLE` if the driver supports writing registers to the device.

We recommended the `v4l2-dbg` utility over calling this `ioctl` directly. It is available from the LinuxTV `v4l-dvb` repository; see <https://linuxtv.org/repo/> for access instructions.

Table 134: struct `v4l2_dbg_match`

<code>__u32</code>	<code>type</code>	See <i>Chip Match Types</i> for a list of possible types.
<code>union {</code>	(anonymous)	
<code>__u32</code>	<code>addr</code>	Match a chip by this number, interpreted according to the <code>type</code> field.
<code>char</code>	<code>name[32]</code>	Match a chip by this name, interpreted according to the <code>type</code> field. Currently unused.
<code>}</code>		

`v4l2_dbg_chip_info`

Table 135: struct `v4l2_dbg_chip_info`

struct <code>v4l2_dbg_match</code>	<code>match</code>	How to match the chip, see <i>struct v4l2_dbg_match</i> .
<code>char</code>	<code>name[32]</code>	The name of the chip.
<code>__u32</code>	<code>flags</code>	Set by the driver. If <code>V4L2_CHIP_FL_READABLE</code> is set, then the driver supports reading registers from the device. If <code>V4L2_CHIP_FL_WRITABLE</code> is set, then it supports writing registers.
<code>__u32</code>	<code>reserved[8]</code>	Reserved fields, both application and driver must set these to 0.

Table 136: Chip Match Types

V4L2_CHIP_MATCH_BRIDGE	0	Match the nth chip on the card, zero for the bridge chip. Does not match sub-devices.
V4L2_CHIP_MATCH_SUBDEV	4	Match the nth sub-device.

Return Value

On success 0 is returned, on error -1 and the `errno` variable is set appropriately. The generic error codes are described at the [Generic Error Codes](#) chapter.

EINVAL The `match_type` is invalid or no device could be matched.

3.2.7.6 ioctl VIDIOC_DBG_G_REGISTER, VIDIOC_DBG_S_REGISTER

Name

VIDIOC_DBG_G_REGISTER - VIDIOC_DBG_S_REGISTER - Read or write hardware registers

Synopsis

VIDIOC_DBG_G_REGISTER

```
int ioctl(int fd, VIDIOC_DBG_G_REGISTER, struct v4l2_dbg_register *argp)
```

VIDIOC_DBG_S_REGISTER

```
int ioctl(int fd, VIDIOC_DBG_S_REGISTER, const struct v4l2_dbg_register *argp)
```

Arguments

fd File descriptor returned by `open()`.

argp Pointer to struct `v4l2_dbg_register`.

Description

Note: This is an *Experimental API Elements* interface and may change in the future.

For driver debugging purposes these ioctls allow test applications to access hardware registers directly. Regular applications must not use them.

Since writing or even reading registers can jeopardize the system security, its stability and damage the hardware, both ioctls require superuser privileges. Additionally the Linux kernel must be compiled with the `CONFIG_VIDEO_ADV_DEBUG` option to enable these ioctls.

To write a register applications must initialize all fields of a struct `v4l2_dbg_register` except for `size` and call `VIDIOC_DBG_S_REGISTER` with a pointer to this structure. The `match.type` and

`match.addr` or `match.name` fields select a chip on the TV card, the `reg` field specifies a register number and the `val` field the value to be written into the register.

To read a register applications must initialize the `match.type`, `match.addr` or `match.name` and `reg` fields, and call `VIDIOC_DBG_G_REGISTER` with a pointer to this structure. On success the driver stores the register value in the `val` field and the size (in bytes) of the value in `size`.

When `match.type` is `V4L2_CHIP_MATCH_BRIDGE`, `match.addr` selects the `nth` non-sub-device chip on the TV card. The number zero always selects the host chip, e. g. the chip connected to the PCI or USB bus. You can find out which chips are present with the *ioctl* `VIDIOC_DBG_G_CHIP_INFO` *ioctl*.

When `match.type` is `V4L2_CHIP_MATCH_SUBDEV`, `match.addr` selects the `nth` sub-device.

These *ioctls* are optional, not all drivers may support them. However when a driver supports these *ioctls* it must also support *ioctl* `VIDIOC_DBG_G_CHIP_INFO`. Conversely it may support `VIDIOC_DBG_G_CHIP_INFO` but not these *ioctls*.

`VIDIOC_DBG_G_REGISTER` and `VIDIOC_DBG_S_REGISTER` were introduced in Linux 2.6.21, but their API was changed to the one described here in kernel 2.6.29.

We recommended the `v4l2-dbg` utility over calling these *ioctls* directly. It is available from the LinuxTV `v4l-dvb` repository; see <https://linuxtv.org/repo/> for access instructions.

v4l2_dbg_match

Table 137: struct `v4l2_dbg_match`

<code>__u32</code>	<code>type</code>	See <i>Chip Match Types</i> for a list of possible types.
<code>union {</code>	(anonymous)	
<code>__u32</code>	<code>addr</code>	Match a chip by this number, interpreted according to the <code>type</code> field.
<code>char</code>	<code>name[32]</code>	Match a chip by this name, interpreted according to the <code>type</code> field. Currently unused.
<code>}</code>		

v4l2_dbg_register

Table 138: struct `v4l2_dbg_register`

<code>struct v4l2_dbg_match</code>	<code>match</code>	How to match the chip, see <code>v4l2_dbg_match</code> .
<code>__u32</code>	<code>size</code>	The register size in bytes.
<code>__u64</code>	<code>reg</code>	A register number.
<code>__u64</code>	<code>val</code>	The value read from, or to be written into the register.

Table 139: Chip Match Types

V4L2_CHIP_MATCH_BRIDGE	0	Match the nth chip on the card, zero for the bridge chip. Does not match sub-devices.
V4L2_CHIP_MATCH_SUBDEV	4	Match the nth sub-device.

Return Value

On success 0 is returned, on error -1 and the `errno` variable is set appropriately. The generic error codes are described at the [Generic Error Codes](#) chapter.

EPERM Insufficient permissions. Root privileges are required to execute these ioctls.

3.2.7.7 ioctl VIDIOC_DECODER_CMD, VIDIOC_TRY_DECODER_CMD

Name

VIDIOC_DECODER_CMD - VIDIOC_TRY_DECODER_CMD - Execute an decoder command

Synopsis

VIDIOC_DECODER_CMD

```
int ioctl(int fd, VIDIOC_DECODER_CMD, struct v4l2_decoder_cmd *argp)
```

VIDIOC_TRY_DECODER_CMD

```
int ioctl(int fd, VIDIOC_TRY_DECODER_CMD, struct v4l2_decoder_cmd *argp)
```

Arguments

fd File descriptor returned by `open()`.

argp pointer to struct `v4l2_decoder_cmd`.

Description

These ioctls control an audio/video (usually MPEG-) decoder. `VIDIOC_DECODER_CMD` sends a command to the decoder, `VIDIOC_TRY_DECODER_CMD` can be used to try a command without actually executing it. To send a command applications must initialize all fields of a struct `v4l2_decoder_cmd` and call `VIDIOC_DECODER_CMD` or `VIDIOC_TRY_DECODER_CMD` with a pointer to this structure.

The `cmd` field must contain the command code. Some commands use the `flags` field for additional information.

A `write()` or *ioctl* `VIDIOC_STREAMON`, `VIDIOC_STREAMOFF` call sends an implicit `START` command to the decoder if it has not been started yet. Applies to both queues of mem2mem decoders.

A `close()` or `VIDIOC_STREAMOFF` call of a streaming file descriptor sends an implicit immediate STOP command to the decoder, and all buffered data is discarded. Applies to both queues of mem2mem decoders.

In principle, these ioctls are optional, not all drivers may support them. They were introduced in Linux 3.3. They are, however, mandatory for stateful mem2mem decoders (as further documented in *Memory-to-Memory Stateful Video Decoder Interface*).

v4l2_decoder_cmd

Table 140: struct v4l2_decoder_cmd

__u32	cmd		The decoder command, see <i>Decoder Commands</i> .
__u32	flags		Flags to go with the command. If no flags are defined for this command, drivers and applications must set this field to zero.
union {	(anonymous)		
struct	start		Structure containing additional data for the V4L2_DEC_CMD_START command.
	__s32	speed	Playback speed and direction. The playback speed is defined as speed/1000 of the normal speed. So 1000 is normal playback. Negative numbers denote reverse playback, so -1000 does reverse playback at normal speed. Speeds -1, 0 and 1 have special meanings: speed 0 is shorthand for 1000 (normal playback). A speed of 1 steps just one frame forward, a speed of -1 steps just one frame back.
	__u32	format	Format restrictions. This field is set by the driver, not the application. Possible values are V4L2_DEC_START_FMT_NONE if there are no format restrictions or V4L2_DEC_START_FMT_GOP if the decoder operates on full GOPs (<i>Group Of Pictures</i>). This is usually the case for reverse playback: the decoder needs full GOPs, which it can then play in reverse order. So to implement reverse playback the application must feed the decoder the last GOP in the video file, then the GOP before that, etc. etc.
struct	stop		Structure containing additional data for the V4L2_DEC_CMD_STOP command.
	__u64	pts	Stop playback at this pts or immediately if the playback is already past that timestamp. Leave to 0 if you want to stop after the last frame was decoded.
struct	raw		
	__u32	data[16]	Reserved for future extensions. Drivers and applications must set the array to zero.
}			

Table 141: Decoder Commands

V4L2_DEC_CMD_START	0	<p>Start the decoder. When the decoder is already running or paused, this command will just change the playback speed. That means that calling V4L2_DEC_CMD_START when the decoder was paused will <i>not</i> resume the decoder. You have to explicitly call V4L2_DEC_CMD_RESUME for that. This command has one flag: V4L2_DEC_CMD_START_MUTE_AUDIO. If set, then audio will be muted when playing back at a non-standard speed.</p> <p>For a device implementing the <i>Memory-to-Memory Stateful Video Decoder Interface</i>, once the drain sequence is initiated with the V4L2_DEC_CMD_STOP command, it must be driven to completion before this command can be invoked. Any attempt to invoke the command while the drain sequence is in progress will trigger an EBUSY error code. The command may be also used to restart the decoder in case of an implicit stop initiated by the decoder itself, without the V4L2_DEC_CMD_STOP being called explicitly. See <i>Memory-to-Memory Stateful Video Decoder Interface</i> for more details.</p>
V4L2_DEC_CMD_STOP	1	<p>Stop the decoder. When the decoder is already stopped, this command does nothing. This command has two flags: if V4L2_DEC_CMD_STOP_TO_BLACK is set, then the decoder will set the picture to black after it stopped decoding. Otherwise the last image will repeat. If V4L2_DEC_CMD_STOP_IMMEDIATELY is set, then the decoder stops immediately (ignoring the pts value), otherwise it will keep decoding until timestamp \geq pts or until the last of the pending data from its internal buffers was decoded.</p> <p>For a device implementing the <i>Memory-to-Memory Stateful Video Decoder Interface</i>, the command will initiate the drain sequence as documented in <i>Memory-to-Memory Stateful Video Decoder Interface</i>. No flags or other arguments are accepted in this case. Any attempt to invoke the command again before the sequence completes will trigger an EBUSY error code.</p>
V4L2_DEC_CMD_PAUSE	2	<p>Pause the decoder. When the decoder has not been started yet, the driver will return an EPERM error code. When the decoder is already paused, this command does nothing. This command has one flag: if V4L2_DEC_CMD_PAUSE_TO_BLACK is set, then set the decoder output to black when paused.</p>
V4L2_DEC_CMD_RESUME	3	<p>Resume decoding after a PAUSE command. When the decoder has not been started yet, the driver will return an EPERM error code. When the decoder is already running, this command does nothing. No flags are defined for this command.</p>

Continued on next page

Table 141 - continued from previous page

V4L2_DEC_CMD_FLUSH	4	Flush any held capture buffers. Only valid for stateless decoders. This command is typically used when the application reached the end of the stream and the last output buffer had the V4L2_BUF_FLAG_M2M_HOLD_CAPTURE_BUF flag set. This would prevent dequeuing the capture buffer containing the last decoded frame. So this command can be used to explicitly flush that final decoded frame. This command does nothing if there are no held capture buffers.
--------------------	---	---

Return Value

On success 0 is returned, on error -1 and the `errno` variable is set appropriately. The generic error codes are described at the [Generic Error Codes](#) chapter.

EBUSY A drain sequence of a device implementing the *Memory-to-Memory Stateful Video Decoder Interface* is still in progress. It is not allowed to issue another decoder command until it completes.

EINVAL The `cmd` field is invalid.

EPERM The application sent a PAUSE or RESUME command when the decoder was not running.

3.2.7.8 ioctl VIDIOC_DQEVENT

Name

VIDIOC_DQEVENT - Dequeue event

Synopsis

VIDIOC_DQEVENT

```
int ioctl(int fd, VIDIOC_DQEVENT, struct v4l2_event *argp)
```

Arguments

fd File descriptor returned by `open()`.

argp Pointer to struct `v4l2_event`.

Description

Dequeue an event from a video device. No input is required for this ioctl. All the fields of the struct `v4l2_event` structure are filled by the driver. The file handle will also receive exceptions which the application may get by e.g. using the select system call.

`v4l2_event`

Table 142: struct `v4l2_event`

<code>__u32</code>	<code>type</code>	Type of the event, see <i>Event Types</i> .
<code>union {</code>	<code>u</code>	
<code>struct v4l2_event_vsync</code>	<code>vsync</code>	Event data for event <code>V4L2_EVENT_VSYNC</code> .
<code>struct v4l2_event_ctrl</code>	<code>ctrl</code>	Event data for event <code>V4L2_EVENT_CTRL</code> .
<code>struct v4l2_event_frame_sync</code>	<code>frame_sync</code>	Event data for event <code>V4L2_EVENT_FRAME_SYNC</code> .
<code>struct v4l2_event_motion_det</code>	<code>motion_det</code>	Event data for event <code>V4L2_EVENT_MOTION_DET</code> .
<code>struct v4l2_event_src_change</code>	<code>src_change</code>	Event data for event <code>V4L2_EVENT_SOURCE_CHANGE</code> .
<code>__u8</code>	<code>data[64]</code>	Event data. Defined by the event type. The union should be used to define easily accessible type for events.
<code>}</code>		
<code>__u32</code>	<code>pending</code>	Number of pending events excluding this one.
<code>__u32</code>	<code>sequence</code>	Event sequence number. The sequence number is incremented for every subscribed event that takes place. If sequence numbers are not contiguous it means that events have been lost.
<code>struct timespec</code>	<code>timestamp</code>	Event timestamp. The timestamp has been taken from the <code>CLOCK_MONOTONIC</code> clock. To access the same clock outside <code>V4L2</code> , use <code>clock_gettime()</code> .
<code>u32</code>	<code>id</code>	The ID associated with the event source. If the event does not have an associated ID (this depends on the event type), then this is 0.
<code>__u32</code>	<code>reserved[8]</code>	Reserved for future extensions. Drivers must set the array to zero.

Table 143: Event Types

<code>V4L2_EVENT_ALL</code>	0	All events. <code>V4L2_EVENT_ALL</code> is valid only for <code>VIDIOC_UNSUBSCRIBE_EVENT</code> for unsubscribing all events at once.
<code>V4L2_EVENT_VSYNC</code>	1	This event is triggered on the vertical sync. This event has a struct <code>v4l2_event_vsync</code> associated with it.

Continued on next page

Table 143 - continued from previous page

V4L2_EVENT_EOS	2	This event is triggered when the end of a stream is reached. This is typically used with MPEG decoders to report to the application when the last of the MPEG stream has been decoded.
V4L2_EVENT_CTRL	3	This event requires that the id matches the control ID from which you want to receive events. This event is triggered if the control's value changes, if a button control is pressed or if the control's flags change. This event has a struct <code>v4l2_event_ctrl</code> associated with it. This struct contains much of the same information as struct <code>v4l2_queryctrl</code> and struct <code>v4l2_control</code> . If the event is generated due to a call to <code>VIDIOC_S_CTRL</code> or <code>VIDIOC_S_EXT_CTRL</code> s, then the event will <i>not</i> be sent to the file handle that called the ioctl function. This prevents nasty feedback loops. If you <i>do</i> want to get the event, then set the <code>V4L2_EVENT_SUB_FL_ALLOW_FEEDBACK</code> flag. This event type will ensure that no information is lost when more events are raised than there is room internally. In that case the struct <code>v4l2_event_ctrl</code> of the second-oldest event is kept, but the changes field of the second-oldest event is ORed with the changes field of the oldest event.
V4L2_EVENT_FRAME_SYNC	4	Triggered immediately when the reception of a frame has begun. This event has a struct <code>v4l2_event_frame_sync</code> associated with it. If the hardware needs to be stopped in the case of a buffer underrun it might not be able to generate this event. In such cases the <code>frame_sequence</code> field in struct <code>v4l2_event_frame_sync</code> will not be incremented. This causes two consecutive frame sequence numbers to have n times frame interval in between them.

Continued on next page

Table 143 - continued from previous page

V4L2_EVENT_SOURCE_CHANGE	5	This event is triggered when a source parameter change is detected during runtime by the video device. It can be a runtime resolution change triggered by a video decoder or the format change happening on an input connector. This event requires that the id matches the input index (when used with a video device node) or the pad index (when used with a subdevice node) from which you want to receive events. This event has a struct <code>v4l2_event_src_change</code> associated with it. The <code>changes</code> bitfield denotes what has changed for the subscribed pad. If multiple events occurred before application could dequeue them, then the changes will have the ORed value of all the events generated.
V4L2_EVENT_MOTION_DET	6	Triggered whenever the motion detection state for one or more of the regions changes. This event has a struct <code>v4l2_event_motion_det</code> associated with it.
V4L2_EVENT_PRIVATE_START	0x08000000	Base event number for driver-private events.

v4l2_event_vsyncTable 144: struct `v4l2_event_vsync`

<code>__u8</code>	field	The upcoming field. See enum <code>v4l2_field</code> .
-------------------	-------	--

v4l2_event_ctrlTable 145: struct `v4l2_event_ctrl`

<code>__u32</code>	changes	A bitmask that tells what has changed. See Control Changes .
<code>__u32</code>	type	The type of the control. See enum <code>v4l2_ctrl_type</code> .
union {	(anonymous)	
<code>__s32</code>	value	The 32-bit value of the control for 32-bit control types. This is 0 for string controls since the value of a string cannot be passed using <code>ioctl VIDIOC_DQEVENT</code> .
<code>__s64</code>	value64	The 64-bit value of the control for 64-bit control types.
}		
<code>__u32</code>	flags	The control flags. See Control Flags .
<code>__s32</code>	minimum	The minimum value of the control. See struct <code>v4l2_queryctrl</code> .
<code>__s32</code>	maximum	The maximum value of the control. See struct <code>v4l2_queryctrl</code> .
<code>__s32</code>	step	The step value of the control. See struct <code>v4l2_queryctrl</code> .
<code>__s32</code>	default_value	The default value of the control. See struct <code>v4l2_queryctrl</code> .

v4l2_event_frame_sync

Table 146: struct v4l2_event_frame_sync

__u32	frame_sequence	The sequence number of the frame being received.
-------	----------------	--

v4l2_event_src_change

Table 147: struct v4l2_event_src_change

__u32	changes	A bitmask that tells what has changed. See Source Changes .
-------	---------	---

v4l2_event_motion_det

Table 148: struct v4l2_event_motion_det

__u32	flags	Currently only one flag is available: if V4L2_EVENT_MD_FL_HAVE_FRAME_SEQ is set, then the frame_sequence field is valid, otherwise that field should be ignored.
__u32	frame_sequence	The sequence number of the frame being received. Only valid if the V4L2_EVENT_MD_FL_HAVE_FRAME_SEQ flag was set.
__u32	region_mask	The bitmask of the regions that reported motion. There is at least one region. If this field is 0, then no motion was detected at all. If there is no V4L2_CID_DETECT_MD_REGION_GRID control (see Detect Control Reference) to assign a different region to each cell in the motion detection grid, then that all cells are automatically assigned to the default region 0.

Table 149: Control Changes

V4L2_EVENT_CTRL_CH_VALUE	0x0001	This control event was triggered because the value of the control changed. Special cases: Volatile controls do not generate this event; If a control has the V4L2_CTRL_FLAG_EXECUTE_ON_WRITE flag set, then this event is sent as well, regardless its value.
V4L2_EVENT_CTRL_CH_FLAGS	0x0002	This control event was triggered because the control flags changed.
V4L2_EVENT_CTRL_CH_RANGE	0x0004	This control event was triggered because the minimum, maximum, step or the default value of the control changed.

Table 150: Source Changes

V4L2_EVENT_SRC_CH_RESOLUTION	0x0001	<p>This event gets triggered when a resolution change is detected at an input. This can come from an input connector or from a video decoder. Applications will have to query the new resolution (if any, the signal may also have been lost).</p> <p>For stateful decoders follow the guidelines in <i>Memory-to-Memory Stateful Video Decoder Interface</i>. Video Capture devices have to query the new timings using <i>ioctl VIDIOC_QUERY_DV_TIMINGS</i> or <i>VIDIOC_QUERYSTD</i>.</p> <p><i>Important:</i> even if the new video timings appear identical to the old ones, receiving this event indicates that there was an issue with the video signal and you must stop and restart streaming (<i>VIDIOC_STREAMOFF</i> followed by <i>VIDIOC_STREAMON</i>). The reason is that many Video Capture devices are not able to recover from a temporary loss of signal and so restarting streaming I/O is required in order for the hardware to synchronize to the video signal.</p>
------------------------------	--------	--

Return Value

On success 0 is returned, on error -1 and the `errno` variable is set appropriately. The generic error codes are described at the *Generic Error Codes* chapter.

3.2.7.9 ioctl VIDIOC_DV_TIMINGS_CAP, VIDIOC_SUBDEV_DV_TIMINGS_CAP

Name

VIDIOC_DV_TIMINGS_CAP - VIDIOC_SUBDEV_DV_TIMINGS_CAP - The capabilities of the Digital Video receiver/transmitter

Synopsis

VIDIOC_DV_TIMINGS_CAP

```
int ioctl(int fd, VIDIOC_DV_TIMINGS_CAP, struct v4l2_dv_timings_cap *argp)
```

VIDIOC_SUBDEV_DV_TIMINGS_CAP

```
int ioctl(int fd, VIDIOC_SUBDEV_DV_TIMINGS_CAP, struct v4l2_dv_timings_cap *argp)
```

Arguments

fd File descriptor returned by `open()`.

argp Pointer to struct `v4l2_dv_timings_cap`.

Description

To query the capabilities of the DV receiver/transmitter applications initialize the `pad` field to 0, zero the reserved array of struct `v4l2_dv_timings_cap` and call the `VIDIOC_DV_TIMINGS_CAP` ioctl on a video node and the driver will fill in the structure.

Note: Drivers may return different values after switching the video input or output.

When implemented by the driver DV capabilities of subdevices can be queried by calling the `VIDIOC_SUBDEV_DV_TIMINGS_CAP` ioctl directly on a subdevice node. The capabilities are specific to inputs (for DV receivers) or outputs (for DV transmitters), applications must specify the desired `pad` number in the struct `v4l2_dv_timings_cap` `pad` field and zero the reserved array. Attempts to query capabilities on a `pad` that doesn't support them will return an `EINVAL` error code.

`v4l2_bt_timings_cap`

Table 151: struct `v4l2_bt_timings_cap`

<code>__u32</code>	<code>min_width</code>	Minimum width of the active video in pixels.
<code>__u32</code>	<code>max_width</code>	Maximum width of the active video in pixels.
<code>__u32</code>	<code>min_height</code>	Minimum height of the active video in lines.
<code>__u32</code>	<code>max_height</code>	Maximum height of the active video in lines.
<code>__u64</code>	<code>min_pixelclock</code>	Minimum pixelclock frequency in Hz.
<code>__u64</code>	<code>max_pixelclock</code>	Maximum pixelclock frequency in Hz.
<code>__u32</code>	<code>standards</code>	The video standard(s) supported by the hardware. See <i>DV BT Timing standards</i> for a list of standards.
<code>__u32</code>	<code>capabilities</code>	Several flags giving more information about the capabilities. See <i>DV BT Timing capabilities</i> for a description of the flags.
<code>__u32</code>	<code>reserved[16]</code>	Reserved for future extensions. Drivers must set the array to zero.

`v4l2_dv_timings_cap`

Table 152: struct v4l2_dv_timings_cap

__u32	type	Type of DV timings as listed in <i>DV Timing types</i> .
__u32	pad	Pad number as reported by the media controller API. This field is only used when operating on a subdevice node. When operating on a video node applications must set this field to zero.
__u32	reserved[2]	Reserved for future extensions. Drivers and applications must set the array to zero.
union {	(anonymous)	
struct v4l2_bt_timings_cap	bt	BT.656/1120 timings capabilities of the hardware.
__u32	raw_data[32]	
}		

Table 153: DV BT Timing capabilities

Flag	Description
V4L2_DV_BT_CAP_INTERLACED	Interlaced formats are supported.
V4L2_DV_BT_CAP_PROGRESSIVE	Progressive formats are supported.
V4L2_DV_BT_CAP_REDUCED_BLANKING	CVT/GTF specific: the timings can make use of reduced blanking (CVT) or the 'Secondary GTF' curve (GTF).
V4L2_DV_BT_CAP_CUSTOM	Can support non-standard timings, i.e. timings not belonging to the standards set in the standards field.

Return Value

On success 0 is returned, on error -1 and the `errno` variable is set appropriately. The generic error codes are described at the *Generic Error Codes* chapter.

3.2.7.10 ioctl VIDIOC_ENCODER_CMD, VIDIOC_TRY_ENCODER_CMD

Name

VIDIOC_ENCODER_CMD - VIDIOC_TRY_ENCODER_CMD - Execute an encoder command

Synopsis

VIDIOC_ENCODER_CMD

```
int ioctl(int fd, VIDIOC_ENCODER_CMD, struct v4l2_encoder_cmd *argp)
```

VIDIOC_TRY_ENCODER_CMD

```
int ioctl(int fd, VIDIOC_TRY_ENCODER_CMD, struct v4l2_encoder_cmd *argp)
```

Arguments

fd File descriptor returned by `open()`.

argp Pointer to struct `v4l2_encoder_cmd`.

Description

These `ioctl`s control an audio/video (usually MPEG-) encoder. `VIDIOC_ENCODER_CMD` sends a command to the encoder, `VIDIOC_TRY_ENCODER_CMD` can be used to try a command without actually executing it.

To send a command applications must initialize all fields of a struct `v4l2_encoder_cmd` and call `VIDIOC_ENCODER_CMD` or `VIDIOC_TRY_ENCODER_CMD` with a pointer to this structure.

The `cmd` field must contain the command code. Some commands use the `flags` field for additional information.

After a `STOP` command, `read()` calls will read the remaining data buffered by the driver. When the buffer is empty, `read()` will return zero and the next `read()` call will restart the encoder.

A `read()` or `VIDIOC_STREAMON` call sends an implicit `START` command to the encoder if it has not been started yet. Applies to both queues of mem2mem encoders.

A `close()` or `VIDIOC_STREAMOFF` call of a streaming file descriptor sends an implicit immediate `STOP` to the encoder, and all buffered data is discarded. Applies to both queues of mem2mem encoders.

These `ioctl`s are optional, not all drivers may support them. They were introduced in Linux 2.6.21. They are, however, mandatory for stateful mem2mem encoders (as further documented in *Memory-to-Memory Stateful Video Encoder Interface*).

v4l2_encoder_cmd

Table 154: struct `v4l2_encoder_cmd`

<code>__u32</code>	<code>cmd</code>	The encoder command, see <i>Encoder Commands</i> .
<code>__u32</code>	<code>flags</code>	Flags to go with the command, see <i>Encoder Command Flags</i> . If no flags are defined for this command, drivers and applications must set this field to zero.
<code>__u32</code>	<code>data[8]</code>	Reserved for future extensions. Drivers and applications must set the array to zero.

Table 155: Encoder Commands

V4L2_ENC_CMD_START	0	<p>Start the encoder. When the encoder is already running or paused, this command does nothing. No flags are defined for this command.</p> <p>For a device implementing the <i>Memory-to-Memory Stateful Video Encoder Interface</i>, once the drain sequence is initiated with the V4L2_ENC_CMD_STOP command, it must be driven to completion before this command can be invoked. Any attempt to invoke the command while the drain sequence is in progress will trigger an EBUSY error code. See <i>Memory-to-Memory Stateful Video Encoder Interface</i> for more details.</p>
V4L2_ENC_CMD_STOP	1	<p>Stop the encoder. When the V4L2_ENC_CMD_STOP_AT_GOP_END flag is set, encoding will continue until the end of the current <i>Group Of Pictures</i>, otherwise encoding will stop immediately. When the encoder is already stopped, this command does nothing.</p> <p>For a device implementing the <i>Memory-to-Memory Stateful Video Encoder Interface</i>, the command will initiate the drain sequence as documented in <i>Memory-to-Memory Stateful Video Encoder Interface</i>. No flags or other arguments are accepted in this case. Any attempt to invoke the command again before the sequence completes will trigger an EBUSY error code.</p>
V4L2_ENC_CMD_PAUSE	2	<p>Pause the encoder. When the encoder has not been started yet, the driver will return an EPERM error code. When the encoder is already paused, this command does nothing. No flags are defined for this command.</p>
V4L2_ENC_CMD_RESUME	3	<p>Resume encoding after a PAUSE command. When the encoder has not been started yet, the driver will return an EPERM error code. When the encoder is already running, this command does nothing. No flags are defined for this command.</p>

Table 156: Encoder Command Flags

V4L2_ENC_CMD_STOP_AT_GOP_END	0x0001	<p>Stop encoding at the end of the current <i>Group Of Pictures</i>, rather than immediately. Does not apply to <i>Memory-to-Memory Stateful Video Encoder Interface</i>.</p>
------------------------------	--------	---

Return Value

On success 0 is returned, on error -1 and the `errno` variable is set appropriately. The generic error codes are described at the [Generic Error Codes](#) chapter.

EBUSY A drain sequence of a device implementing the [Memory-to-Memory Stateful Video Encoder Interface](#) is still in progress. It is not allowed to issue another encoder command until it completes.

EINVAL The `cmd` field is invalid.

EPERM The application sent a PAUSE or RESUME command when the encoder was not running.

3.2.7.11 ioctl VIDIOC_ENUMAUDIO

Name

VIDIOC_ENUMAUDIO - Enumerate audio inputs

Synopsis

VIDIOC_ENUMAUDIO

```
int ioctl(int fd, VIDIOC_ENUMAUDIO, struct v4l2_audio *argp)
```

Arguments

fd File descriptor returned by `open()`.

argp Pointer to struct `v4l2_audio`.

Description

To query the attributes of an audio input applications initialize the `index` field and zero out the reserved array of a struct `v4l2_audio` and call the `ioctl VIDIOC_ENUMAUDIO` `ioctl` with a pointer to this structure. Drivers fill the rest of the structure or return an `EINVAL` error code when the `index` is out of bounds. To enumerate all audio inputs applications shall begin at `index` zero, incrementing by one until the driver returns `EINVAL`.

See [VIDIOC_G_AUDIO](#) for a description of struct `v4l2_audio`.

Return Value

On success 0 is returned, on error -1 and the `errno` variable is set appropriately. The generic error codes are described at the [Generic Error Codes](#) chapter.

EINVAL The number of the audio input is out of bounds.

3.2.7.12 ioctl VIDIOC_ENUMAUDOUT

Name

VIDIOC_ENUMAUDOUT - Enumerate audio outputs

Synopsis

VIDIOC_ENUMAUDOUT

```
int ioctl(int fd, VIDIOC_ENUMAUDOUT, struct v4l2_audioout *argp)
```

Arguments

fd File descriptor returned by `open()`.

argp Pointer to struct `v4l2_audioout`.

Description

To query the attributes of an audio output applications initialize the `index` field and zero out the reserved array of a struct `v4l2_audioout` and call the `VIDIOC_G_AUDOUT` ioctl with a pointer to this structure. Drivers fill the rest of the structure or return an `EINVAL` error code when the `index` is out of bounds. To enumerate all audio outputs applications shall begin at index zero, incrementing by one until the driver returns `EINVAL`.

Note: Connectors on a TV card to loop back the received audio signal to a sound card are not audio outputs in this sense.

See [VIDIOC_G_AUDIOout](#) for a description of struct `v4l2_audioout`.

Return Value

On success 0 is returned, on error -1 and the `errno` variable is set appropriately. The generic error codes are described at the [Generic Error Codes](#) chapter.

EINVAL The number of the audio output is out of bounds.

3.2.7.13 `ioctl VIDIOC_ENUM_DV_TIMINGS, VIDIOC_SUBDEV_ENUM_DV_TIMINGS`

Name

`VIDIOC_ENUM_DV_TIMINGS - VIDIOC_SUBDEV_ENUM_DV_TIMINGS` - Enumerate supported Digital Video timings

Synopsis

`VIDIOC_ENUM_DV_TIMINGS`

```
int ioctl(int fd, VIDIOC_ENUM_DV_TIMINGS, struct v4l2_enum_dv_timings *argp)
```

`VIDIOC_SUBDEV_ENUM_DV_TIMINGS`

```
int ioctl(int fd, VIDIOC_SUBDEV_ENUM_DV_TIMINGS, struct v4l2_enum_dv_timings *argp)
```

Arguments

fd File descriptor returned by `open()`.

argp Pointer to `struct v4l2_enum_dv_timings`.

Description

While some DV receivers or transmitters support a wide range of timings, others support only a limited number of timings. With this `ioctl` applications can enumerate a list of known supported timings. Call `ioctl VIDIOC_DV_TIMINGS_CAP, VIDIOC_SUBDEV_DV_TIMINGS_CAP` to check if it also supports other standards or even custom timings that are not in this list.

To query the available timings, applications initialize the `index` field, set the `pad` field to 0, zero the reserved array of `struct v4l2_enum_dv_timings` and call the `VIDIOC_ENUM_DV_TIMINGS` `ioctl` on a video node with a pointer to this structure. Drivers fill the rest of the structure or return an `EINVAL` error code when the `index` is out of bounds. To enumerate all supported DV timings, applications shall begin at `index` zero, incrementing by one until the driver returns `EINVAL`.

Note: Drivers may enumerate a different set of DV timings after switching the video input or output.

When implemented by the driver DV timings of subdevices can be queried by calling the `VIDIOC_SUBDEV_ENUM_DV_TIMINGS` ioctl directly on a subdevice node. The DV timings are specific to inputs (for DV receivers) or outputs (for DV transmitters), applications must specify the desired pad number in the struct `v4l2_enum_dv_timings` pad field. Attempts to enumerate timings on a pad that doesn't support them will return an `EINVAL` error code.

`v4l2_enum_dv_timings`

Table 157: struct `v4l2_enum_dv_timings`

<code>__u32</code>	<code>index</code>	Number of the DV timings, set by the application.
<code>__u32</code>	<code>pad</code>	Pad number as reported by the media controller API. This field is only used when operating on a subdevice node. When operating on a video node applications must set this field to zero.
<code>__u32</code>	<code>reserved[2]</code>	Reserved for future extensions. Drivers and applications must set the array to zero.
struct <code>v4l2_dv_timings</code>	<code>timings</code>	The timings.

Return Value

On success 0 is returned, on error -1 and the `errno` variable is set appropriately. The generic error codes are described at the [Generic Error Codes](#) chapter.

EINVAL The struct `v4l2_enum_dv_timings` index is out of bounds or the pad number is invalid.

ENODATA Digital video presets are not supported for this input or output.

3.2.7.14 ioctl `VIDIOC_ENUM_FMT`

Name

`VIDIOC_ENUM_FMT` - Enumerate image formats

Synopsis

`VIDIOC_ENUM_FMT`

```
int ioctl(int fd, VIDIOC_ENUM_FMT, struct v4l2_fmtdesc *argp)
```

Arguments

fd File descriptor returned by `open()`.

argp Pointer to struct `v4l2_fmtdesc`.

Description

To enumerate image formats applications initialize the `type`, `mbus_code` and `index` fields of struct `v4l2_fmtdesc` and call the `ioctl VIDIOC_ENUM_FMT` ioctl with a pointer to this structure. Drivers fill the rest of the structure or return an `EINVAL` error code. All formats are enumerable by beginning at index zero and incrementing by one until `EINVAL` is returned. If applicable, drivers shall return formats in preference order, where preferred formats are returned before (that is, with lower index value) less-preferred formats.

Depending on the `V4L2_CAP_IO_MC` *capability*, the `mbus_code` field is handled differently:

- 1) `V4L2_CAP_IO_MC` is not set (also known as a 'video-node-centric' driver)

Applications shall initialize the `mbus_code` field to zero and drivers shall ignore the value of the field.

Drivers shall enumerate all image formats.

Note: After switching the input or output the list of enumerated image formats may be different.

- 2) `V4L2_CAP_IO_MC` is set (also known as an 'MC-centric' driver)

If the `mbus_code` field is zero, then all image formats shall be enumerated.

If the `mbus_code` field is initialized to a valid (non-zero) *media bus format code*, then drivers shall restrict enumeration to only the image formats that can produce (for video output devices) or be produced from (for video capture devices) that media bus code. If the `mbus_code` is unsupported by the driver, then `EINVAL` shall be returned.

Regardless of the value of the `mbus_code` field, the enumerated image formats shall not depend on the active configuration of the video device or device pipeline.

v4l2_fmtdesc

Table 158: struct v4l2_fmtdesc

__u32	index	Number of the format in the enumeration, set by the application. This is in no way related to the pixelformat field.
__u32	type	Type of the data stream, set by the application. Only these types are valid here: V4L2_BUF_TYPE_VIDEO_CAPTURE, V4L2_BUF_TYPE_VIDEO_CAPTURE_MPLANE, V4L2_BUF_TYPE_VIDEO_OUTPUT, V4L2_BUF_TYPE_VIDEO_OUTPUT_MPLANE, V4L2_BUF_TYPE_VIDEO_OVERLAY, V4L2_BUF_TYPE_SDR_CAPTURE, V4L2_BUF_TYPE_SDR_OUTPUT, V4L2_BUF_TYPE_META_CAPTURE and V4L2_BUF_TYPE_META_OUTPUT. See v4l2_buf_type.
__u32	flags	See <i>Image Format Description Flags</i>
__u8	description[32]	Description of the format, a NUL-terminated ASCII string. This information is intended for the user, for example: "YUV 4:2:2".
__u32	pixelformat	The image format identifier. This is a four character code as computed by the v4l2_fourcc() macro:
<pre>#define v4l2_fourcc(a,b,c,d) (((__u32)(a)<<0) ((__u32)(b)<<8) ((__u32)(c)<<16) ((__u32)(d)<<24))</pre> <p>Several image formats are already defined by this specification in <i>Image Formats</i>.</p>		
<p>Attention: These codes are not the same as those used in the Windows world.</p>		
__u32	mbus_code	Media bus code restricting the enumerated formats, set by the application. Only applicable to drivers that advertise the V4L2_CAP_IO_MC <i>capability</i> , shall be 0 otherwise.
__u32	reserved[3]	Reserved for future extensions. Drivers must set the array to zero.

Table 159: Image Format Description Flags

V4L2_FMT_FLAG_COMPRESSED	0x0001	This is a compressed format.
V4L2_FMT_FLAG_EMULATED	0x0002	This format is not native to the device but emulated through software (usually libv4l2), where possible try to use a native format instead for better performance.

Continued on next page

Table 159 - continued from previous page

V4L2_FMT_FLAG_CONTINUOUS_BYTESTREAM	0x0004	The hardware decoder for this compressed bytestream format (aka coded format) is capable of parsing a continuous bytestream. Applications do not need to parse the bytestream themselves to find the boundaries between frames/fields. This flag can only be used in combination with the V4L2_FMT_FLAG_COMPRESSED flag, since this applies to compressed formats only. This flag is valid for stateful decoders only.
V4L2_FMT_FLAG_DYN_RESOLUTION	0x0008	Dynamic resolution switching is supported by the device for this compressed bytestream format (aka coded format). It will notify the user via the event V4L2_EVENT_SOURCE_CHANGE when changes in the video parameters are detected. This flag can only be used in combination with the V4L2_FMT_FLAG_COMPRESSED flag, since this applies to compressed formats only. This flag is valid for stateful codecs only.

Continued on next page

Table 159 - continued from previous page

V4L2_FMT_FLAG_ENC_CAP_FRAME_INTERVAL	0x0010	<p>The hardware encoder supports setting the CAPTURE coded frame interval separately from the OUTPUT raw frame interval. Setting the OUTPUT raw frame interval with <i>VIDIOC_S_PARM</i> also sets the CAPTURE coded frame interval to the same value. If this flag is set, then the CAPTURE coded frame interval can be set to a different value afterwards. This is typically used for offline encoding where the OUTPUT raw frame interval is used as a hint for reserving hardware encoder resources and the CAPTURE coded frame interval is the actual frame rate embedded in the encoded video stream.</p> <p>This flag can only be used in combination with the <i>V4L2_FMT_FLAG_COMPRESSED</i> flag, since this applies to compressed formats only. This flag is valid for stateful encoders only.</p>
V4L2_FMT_FLAG_CSC_COLORSPACE	0x0020	<p>The driver allows the application to try to change the default colorspace. This flag is relevant only for capture devices. The application can ask to configure the colorspace of the capture device when calling the <i>VIDIOC_S_FMT</i> ioctl with <i>V4L2_PIX_FMT_FLAG_SET_CSC</i> set.</p>
V4L2_FMT_FLAG_CSC_XFER_FUNC	0x0040	<p>The driver allows the application to try to change the default transfer function. This flag is relevant only for capture devices. The application can ask to configure the transfer function of the capture device when calling the <i>VIDIOC_S_FMT</i> ioctl with <i>V4L2_PIX_FMT_FLAG_SET_CSC</i> set.</p>

Continued on next page

Table 159 - continued from previous page

V4L2_FMT_FLAG_CSC_YCBCR_ENC	0x0080	The driver allows the application to try to change the default Y'CbCr encoding. This flag is relevant only for capture devices. The application can ask to configure the Y'CbCr encoding of the capture device when calling the <i>VIDIOC_S_FMT</i> ioctl with <i>V4L2_PIX_FMT_FLAG_SET_CSC</i> set.
V4L2_FMT_FLAG_CSC_HSV_ENC	0x0080	The driver allows the application to try to change the default HSV encoding. This flag is relevant only for capture devices. The application can ask to configure the HSV encoding of the capture device when calling the <i>VIDIOC_S_FMT</i> ioctl with <i>V4L2_PIX_FMT_FLAG_SET_CSC</i> set.
V4L2_FMT_FLAG_CSC_QUANTIZATION	0x0100	The driver allows the application to try to change the default quantization. This flag is relevant only for capture devices. The application can ask to configure the quantization of the capture device when calling the <i>VIDIOC_S_FMT</i> ioctl with <i>V4L2_PIX_FMT_FLAG_SET_CSC</i> set.

Return Value

On success 0 is returned, on error -1 and the `errno` variable is set appropriately. The generic error codes are described at the [Generic Error Codes](#) chapter.

EINVAL The struct `v4l2_fmtdesc` type is not supported or the index is out of bounds.

If `V4L2_CAP_IO_MC` is set and the specified `mbus_code` is unsupported, then also return this error code.

3.2.7.15 ioctl VIDIOC_ENUM_FRAMESIZES

Name

VIDIOC_ENUM_FRAMESIZES - Enumerate frame sizes

Synopsis

VIDIOC_ENUM_FRAMESIZES

```
int ioctl(int fd, VIDIOC_ENUM_FRAMESIZES, struct v4l2_frmsizeenum *argp)
```

Arguments

fd File descriptor returned by `open()`.

argp Pointer to `struct v4l2_frmsizeenum` that contains an index and pixel format and receives a frame width and height.

Description

This `ioctl` allows applications to enumerate all frame sizes (i. e. width and height in pixels) that the device supports for the given pixel format.

The supported pixel formats can be obtained by using the `ioctl VIDIOC_ENUM_FMT` function.

The return value and the content of the `v4l2_frmsizeenum.type` field depend on the type of frame sizes the device supports. Here are the semantics of the function for the different cases:

- **Discrete:** The function returns success if the given index value (zero-based) is valid. The application should increase the index by one for each call until `EINVAL` is returned. The `v4l2_frmsizeenum.type` field is set to `V4L2_FRMSIZE_TYPE_DISCRETE` by the driver. Of the union only the discrete member is valid.
- **Step-wise:** The function returns success if the given index value is zero and `EINVAL` for any other index value. The `v4l2_frmsizeenum.type` field is set to `V4L2_FRMSIZE_TYPE_STEPWISE` by the driver. Of the union only the stepwise member is valid.
- **Continuous:** This is a special case of the step-wise type above. The function returns success if the given index value is zero and `EINVAL` for any other index value. The `v4l2_frmsizeenum.type` field is set to `V4L2_FRMSIZE_TYPE_CONTINUOUS` by the driver. Of the union only the stepwise member is valid and the `step_width` and `step_height` values are set to 1.

When the application calls the function with index zero, it must check the `type` field to determine the type of frame size enumeration the device supports. Only for the `V4L2_FRMSIZE_TYPE_DISCRETE` type does it make sense to increase the index value to receive more frame sizes.

Note: The order in which the frame sizes are returned has no special meaning. In particular does it not say anything about potential default format sizes.

Applications can assume that the enumeration data does not change without any interaction from the application itself. This means that the enumeration data is consistent if the application does not perform any other `ioctl` calls while it runs the frame size enumeration.

Structs

In the structs below, *IN* denotes a value that has to be filled in by the application, *OUT* denotes values that the driver fills in. The application should zero out all members except for the *IN* fields.

v4l2_frmsize_discrete

Table 160: struct v4l2_frmsize_discrete

__u32	width	Width of the frame [pixel].
__u32	height	Height of the frame [pixel].

v4l2_frmsize_stepwise

Table 161: struct v4l2_frmsize_stepwise

__u32	min_width	Minimum frame width [pixel].
__u32	max_width	Maximum frame width [pixel].
__u32	step_width	Frame width step size [pixel].
__u32	min_height	Minimum frame height [pixel].
__u32	max_height	Maximum frame height [pixel].
__u32	step_height	Frame height step size [pixel].

v4l2_frmsizeenum

Table 162: struct v4l2_frmsizeenum

__u32	index	IN: Index of the given frame size in the enumeration.
__u32	pixel_format	IN: Pixel format for which the frame sizes are enumerated.
__u32	type	OUT: Frame size type the device supports.
union {	(anonymous)	OUT: Frame size with the given index.
struct v4l2_frmsize_discrete	discrete	
struct v4l2_frmsize_stepwise	stepwise	
}		
__u32	reserved[2]	Reserved space for future use. Must be zeroed by drivers and applications.

Enums

v4l2_frmsizetypes

Table 163: enum v4l2_frmsizetypes

V4L2_FRMSIZE_TYPE_DISCRETE	1	Discrete frame size.
V4L2_FRMSIZE_TYPE_CONTINUOUS	2	Continuous frame size.
V4L2_FRMSIZE_TYPE_STEPWISE	3	Step-wise defined frame size.

Return Value

On success 0 is returned, on error -1 and the `errno` variable is set appropriately. The generic error codes are described at the [Generic Error Codes](#) chapter.

3.2.7.16 `ioctl VIDIOC_ENUM_FRAMEINTERVALS`

Name

`VIDIOC_ENUM_FRAMEINTERVALS` - Enumerate frame intervals

Synopsis

`VIDIOC_ENUM_FRAMEINTERVALS`

```
int ioctl(int fd, VIDIOC_ENUM_FRAMEINTERVALS, struct v4l2_fmvalenum *argp)
```

Arguments

fd File descriptor returned by `open()`.

argp Pointer to `struct v4l2_fmvalenum` that contains a pixel format and size and receives a frame interval.

Description

This `ioctl` allows applications to enumerate all frame intervals that the device supports for the given pixel format and frame size.

The supported pixel formats and frame sizes can be obtained by using the `ioctl VIDIOC_ENUM_FMT` and `ioctl VIDIOC_ENUM_FRAMESIZES` functions.

The return value and the content of the `v4l2_fmvalenum.type` field depend on the type of frame intervals the device supports. Here are the semantics of the function for the different cases:

- **Discrete:** The function returns success if the given index value (zero-based) is valid. The application should increase the index by one for each call until `EINVAL` is returned. The `v4l2_fmvalenum.type` field is set to `V4L2_FRMIVAL_TYPE_DISCRETE` by the driver. Of the union only the *discrete* member is valid.
- **Step-wise:** The function returns success if the given index value is zero and `EINVAL` for any other index value. The `v4l2_fmvalenum.type` field is set to `V4L2_FRMIVAL_TYPE_STEPWISE` by the driver. Of the union only the *stepwise* member is valid.
- **Continuous:** This is a special case of the step-wise type above. The function returns success if the given index value is zero and `EINVAL` for any other index value. The `v4l2_fmvalenum.type` field is set to `V4L2_FRMIVAL_TYPE_CONTINUOUS` by the driver. Of the union only the *stepwise* member is valid and the *step* value is set to 1.

When the application calls the function with index zero, it must check the `type` field to determine the type of frame interval enumeration the device supports. Only for the `V4L2_FRMIVAL_TYPE_DISCRETE` type does it make sense to increase the index value to receive more frame intervals.

Note: The order in which the frame intervals are returned has no special meaning. In particular does it not say anything about potential default frame intervals.

Applications can assume that the enumeration data does not change without any interaction from the application itself. This means that the enumeration data is consistent if the application does not perform any other `ioctl` calls while it runs the frame interval enumeration.

Note: Frame intervals and frame rates: The V4L2 API uses frame intervals instead of frame rates. Given the frame interval the frame rate can be computed as follows:

$$\text{frame_rate} = 1 / \text{frame_interval}$$

Structs

In the structs below, *IN* denotes a value that has to be filled in by the application, *OUT* denotes values that the driver fills in. The application should zero out all members except for the *IN* fields.

`v4l2_frmival_stepwise`

Table 164: struct `v4l2_frmival_stepwise`

struct <code>v4l2_fract</code>	<code>min</code>	Minimum frame interval [s].
struct <code>v4l2_fract</code>	<code>max</code>	Maximum frame interval [s].
struct <code>v4l2_fract</code>	<code>step</code>	Frame interval step size [s].

`v4l2_frmivalenum`

Table 165: struct v4l2_fmvalenum

__u32	index	IN: Index of the given frame interval in the enumeration.
__u32	pixel_format	IN: Pixel format for which the frame intervals are enumerated.
__u32	width	IN: Frame width for which the frame intervals are enumerated.
__u32	height	IN: Frame height for which the frame intervals are enumerated.
__u32	type	OUT: Frame interval type the device supports.
union {	(anonymous)	OUT: Frame interval with the given index.
struct v4l2_fract	discrete	Frame interval [s].
struct v4l2_fmival_stepwise	stepwise	
}		
__u32	reserved[2]	Reserved space for future use. Must be zeroed by drivers and applications.

Enums

v4l2_fmivaltypes

Table 166: enum v4l2_fmivaltypes

V4L2_FRMIVAL_TYPE_DISCRETE	1	Discrete frame interval.
V4L2_FRMIVAL_TYPE_CONTINUOUS	2	Continuous frame interval.
V4L2_FRMIVAL_TYPE_STEPWISE	3	Step-wise defined frame interval.

Return Value

On success 0 is returned, on error -1 and the `errno` variable is set appropriately. The generic error codes are described at the [Generic Error Codes](#) chapter.

3.2.7.17 ioctl VIDIOC_ENUM_FREQ_BANDS

Name

VIDIOC_ENUM_FREQ_BANDS - Enumerate supported frequency bands

Synopsis

VIDIOC_ENUM_FREQ_BANDS

```
int ioctl(int fd, VIDIOC_ENUM_FREQ_BANDS, struct v4l2_frequency_band *argp)
```

Arguments

fd File descriptor returned by `open()`.

argp Pointer to `struct v4l2_frequency_band`.

Description

Enumerates the frequency bands that a tuner or modulator supports. To do this applications initialize the `tuner`, `type` and `index` fields, and zero out the reserved array of a `struct v4l2_frequency_band` and call the *ioctl* `VIDIOC_ENUM_FREQ_BANDS` *ioctl* with a pointer to this structure.

This *ioctl* is supported if the `V4L2_TUNER_CAP_FREQ_BANDS` capability of the corresponding tuner/modulator is set.

v4l2_frequency_band

Table 167: struct v4l2_frequency_band

__u32	tuner	The tuner or modulator index number. This is the same value as in the struct v4l2_input tuner field and the struct v4l2_tuner index field, or the struct v4l2_output modulator field and the struct v4l2_modulator index field.
__u32	type	The tuner type. This is the same value as in the struct v4l2_tuner type field. The type must be set to V4L2_TUNER_RADIO for /dev/radioX device nodes, and to V4L2_TUNER_ANALOG_TV for all others. Set this field to V4L2_TUNER_RADIO for modulators (currently only radio modulators are supported). See v4l2_tuner_type
__u32	index	Identifies the frequency band, set by the application.
__u32	capability	The tuner/modulator capability flags for this frequency band, see <i>Tuner and Modulator Capability Flags</i> . The V4L2_TUNER_CAP_LOW or V4L2_TUNER_CAP_1HZ capability must be the same for all frequency bands of the selected tuner/modulator. So either all bands have that capability set, or none of them have that capability.
__u32	rangelow	The lowest tunable frequency in units of 62.5 kHz, or if the capability flag V4L2_TUNER_CAP_LOW is set, in units of 62.5 Hz, for this frequency band. A 1 Hz unit is used when the capability flag V4L2_TUNER_CAP_1HZ is set.
__u32	rangehigh	The highest tunable frequency in units of 62.5 kHz, or if the capability flag V4L2_TUNER_CAP_LOW is set, in units of 62.5 Hz, for this frequency band. A 1 Hz unit is used when the capability flag V4L2_TUNER_CAP_1HZ is set.
__u32	modulation	The supported modulation systems of this frequency band. See <i>Band Modulation Systems</i> . Note: Currently only one modulation system per frequency band is supported. More work will need to be done if multiple modulation systems are possible. Contact the linux-media mailing list (https://linuxtv.org/lists.php) if you need such functionality.
__u32	reserved[9]	Reserved for future extensions. Applications and drivers must set the array to zero.

Table 168: Band Modulation Systems

V4L2_BAND_MODULATION_VSB	0x02	Vestigial Sideband modulation, used for analog TV.
V4L2_BAND_MODULATION_FM	0x04	Frequency Modulation, commonly used for analog radio.
V4L2_BAND_MODULATION_AM	0x08	Amplitude Modulation, commonly used for analog radio.

Return Value

On success 0 is returned, on error -1 and the `errno` variable is set appropriately. The generic error codes are described at the [Generic Error Codes](#) chapter.

EINVAL The tuner or index is out of bounds or the type field is wrong.

3.2.7.18 `ioctl VIDIOC_ENUMINPUT`

Name

`VIDIOC_ENUMINPUT` - Enumerate video inputs

Synopsis

`VIDIOC_ENUMINPUT`

```
int ioctl(int fd, VIDIOC_ENUMINPUT, struct v4l2_input *argp)
```

Arguments

fd File descriptor returned by `open()`.

argp Pointer to struct `v4l2_input`.

Description

To query the attributes of a video input applications initialize the `index` field of struct `v4l2_input` and call the `ioctl VIDIOC_ENUMINPUT` with a pointer to this structure. Drivers fill the rest of the structure or return an `EINVAL` error code when the index is out of bounds. To enumerate all inputs applications shall begin at index zero, incrementing by one until the driver returns `EINVAL`.

`v4l2_input`

Table 169: struct v4l2_input

__u32	index	Identifies the input, set by the application.
__u8	name[32]	Name of the video input, a NUL-terminated ASCII string, for example: "Vin (Composite 2)". This information is intended for the user, preferably the connector label on the device itself.
__u32	type	Type of the input, see <i>Input Types</i> .
__u32	audioset	Drivers can enumerate up to 32 video and audio inputs. This field shows which audio inputs were selectable as audio source if this was the currently selected video input. It is a bit mask. The LSB corresponds to audio input 0, the MSB to input 31. Any number of bits can be set, or none. When the driver does not enumerate audio inputs no bits must be set. Applications shall not interpret this as lack of audio support. Some drivers automatically select audio sources and do not enumerate them since there is no choice anyway. For details on audio inputs and how to select the current input see <i>Audio Inputs and Outputs</i> .
__u32	tuner	Capture devices can have zero or more tuners (RF demodulators). When the type is set to V4L2_INPUT_TYPE_TUNER this is an RF connector and this field identifies the tuner. It corresponds to struct v4l2_tuner field index. For details on tuners see <i>Tuners and Modulators</i> .
<i>v4l2_std_id</i>	std	Every video input supports one or more different video standards. This field is a set of all supported standards. For details on video standards and how to switch see <i>Video Standards</i> .
__u32	status	This field provides status information about the input. See <i>Input Status Flags</i> for flags. With the exception of the sensor orientation bits status is only valid when this is the current input.
__u32	capabilities	This field provides capabilities for the input. See <i>Input capabilities</i> for flags.
__u32	reserved[3]	Reserved for future extensions. Drivers must set the array to zero.

Table 170: Input Types

V4L2_INPUT_TYPE_TUNER	1	This input uses a tuner (RF demodulator).
V4L2_INPUT_TYPE_CAMERA	2	Any non-tuner video input, for example Composite Video, S-Video, HDMI, camera sensor. The naming as <code>_TYPE_CAMERA</code> is historical, today we would have called it <code>_TYPE_VIDEO</code> .
V4L2_INPUT_TYPE_TOUCH	3	This input is a touch device for capturing raw touch data.

Table 171: Input Status Flags

General		
V4L2_IN_ST_NO_POWER	0x00000001	Attached device is off.
V4L2_IN_ST_NO_SIGNAL	0x00000002	
V4L2_IN_ST_NO_COLOR	0x00000004	The hardware supports color decoding, but does not detect color modulation in the signal.
Sensor Orientation		
V4L2_IN_ST_HFLIP	0x00000010	The input is connected to a device that produces a signal that is flipped horizontally and does not correct this before passing the signal to userspace.
V4L2_IN_ST_VFLIP	0x00000020	The input is connected to a device that produces a signal that is flipped vertically and does not correct this before passing the signal to userspace. .. note:: A 180 degree rotation is the same as HFLIP VFLIP
Analog Video		
V4L2_IN_ST_NO_H_LOCK	0x00000100	No horizontal sync lock.
V4L2_IN_ST_COLOR_KILL	0x00000200	A color killer circuit automatically disables color decoding when it detects no color modulation. When this flag is set the color killer is enabled <i>and</i> has shut off color decoding.
V4L2_IN_ST_NO_V_LOCK	0x00000400	No vertical sync lock.
V4L2_IN_ST_NO_STD_LOCK	0x00000800	No standard format lock in case of auto-detection format by the component.
Digital Video		
V4L2_IN_ST_NO_SYNC	0x00010000	No synchronization lock.
V4L2_IN_ST_NO_EQU	0x00020000	No equalizer lock.
V4L2_IN_ST_NO_CARRIER	0x00040000	Carrier recovery failed.
VCR and Set-Top Box		
V4L2_IN_ST_MACROVISION	0x01000000	Macrovision is an analog copy prevention system mangling the video signal to confuse video recorders. When this flag is set Macrovision has been detected.
V4L2_IN_ST_NO_ACCESS	0x02000000	Conditional access denied.
V4L2_IN_ST_VTR	0x04000000	VTR time constant. [?]

Table 172: Input capabilities

V4L2_IN_CAP_DV_TIMINGS	0x00000002	This input supports setting video timings by using <code>VIDIOC_S_DV_TIMINGS</code> .
V4L2_IN_CAP_STD	0x00000004	This input supports setting the TV standard by using <code>VIDIOC_S_STD</code> .
V4L2_IN_CAP_NATIVE_SIZE	0x00000008	This input supports setting the native size using the <code>V4L2_SEL_TGT_NATIVE_SIZE</code> selection target, see <i>Common selection definitions</i> .

Return Value

On success 0 is returned, on error -1 and the `errno` variable is set appropriately. The generic error codes are described at the [Generic Error Codes](#) chapter.

EINVAL The struct `v4l2_input` index is out of bounds.

3.2.7.19 `ioctl VIDIOC_ENUMOUTPUT`

Name

`VIDIOC_ENUMOUTPUT` - Enumerate video outputs

Synopsis

`VIDIOC_ENUMOUTPUT`

```
int ioctl(int fd, VIDIOC_ENUMOUTPUT, struct v4l2_output *argp)
```

Arguments

fd File descriptor returned by `open()`.

argp Pointer to struct `v4l2_output`.

Description

To query the attributes of a video outputs applications initialize the `index` field of struct `v4l2_output` and call the `ioctl VIDIOC_ENUMOUTPUT` with a pointer to this structure. Drivers fill the rest of the structure or return an `EINVAL` error code when the index is out of bounds. To enumerate all outputs applications shall begin at index zero, incrementing by one until the driver returns `EINVAL`.

`v4l2_output`

Table 173: struct v4l2_output

__u32	index	Identifies the output, set by the application.
__u8	name[32]	Name of the video output, a NUL-terminated ASCII string, for example: "Vout". This information is intended for the user, preferably the connector label on the device itself.
__u32	type	Type of the output, see <i>Output Type</i> .
__u32	audioset	Drivers can enumerate up to 32 video and audio outputs. This field shows which audio outputs were selectable as the current output if this was the currently selected video output. It is a bit mask. The LSB corresponds to audio output 0, the MSB to output 31. Any number of bits can be set, or none. When the driver does not enumerate audio outputs no bits must be set. Applications shall not interpret this as lack of audio support. Drivers may automatically select audio outputs without enumerating them. For details on audio outputs and how to select the current output see <i>Audio Inputs and Outputs</i> .
__u32	modulator	Output devices can have zero or more RF modulators. When the type is V4L2_OUTPUT_TYPE_MODULATOR this is an RF connector and this field identifies the modulator. It corresponds to struct v4l2_modulator field index. For details on modulators see <i>Tuners and Modulators</i> .
<i>v4l2_std_id</i>	std	Every video output supports one or more different video standards. This field is a set of all supported standards. For details on video standards and how to switch see <i>Video Standards</i> .
__u32	capabilities	This field provides capabilities for the output. See <i>Output capabilities</i> for flags.
__u32	reserved[3]	Reserved for future extensions. Drivers must set the array to zero.

Table 174: Output Type

V4L2_OUTPUT_TYPE_MODULATOR	1	This output is an analog TV modulator.
V4L2_OUTPUT_TYPE_ANALOG	2	Any non-modulator video output, for example Composite Video, S-Video, HDMI. The naming as _TYPE_ANALOG is historical, today we would have called it _TYPE_VIDEO.
V4L2_OUTPUT_TYPE_ANALOGVGAOVERLAY	3	The video output will be copied to a <i>video overlay</i> .

Table 175: Output capabilities

V4L2_OUT_CAP_DV_TIMINGS	0x00000002	This output supports setting video timings by using VIDIOC_S_DV_TIMINGS.
V4L2_OUT_CAP_STD	0x00000004	This output supports setting the TV standard by using VIDIOC_S_STD.
V4L2_OUT_CAP_NATIVE_SIZE	0x00000008	This output supports setting the native size using the V4L2_SEL_TGT_NATIVE_SIZE selection target, see <i>Common selection definitions</i> .

Return Value

On success 0 is returned, on error -1 and the `errno` variable is set appropriately. The generic error codes are described at the *Generic Error Codes* chapter.

EINVAL The struct `v4l2_output` index is out of bounds.

3.2.7.20 ioctl VIDIOC_ENUMSTD, VIDIOC_SUBDEV_ENUMSTD

Name

VIDIOC_ENUMSTD - VIDIOC_SUBDEV_ENUMSTD - Enumerate supported video standards

Synopsis

VIDIOC_ENUMSTD

```
int ioctl(int fd, VIDIOC_ENUMSTD, struct v4l2_standard *argp)
```

VIDIOC_SUBDEV_ENUMSTD

```
int ioctl(int fd, VIDIOC_SUBDEV_ENUMSTD, struct v4l2_standard *argp)
```

Arguments

fd File descriptor returned by `open()`.

argp Pointer to struct `v4l2_standard`.

Description

To query the attributes of a video standard, especially a custom (driver defined) one, applications initialize the `index` field of `struct v4l2_standard` and call the `ioctl VIDIOC_ENUMSTD`, `VIDIOC_SUBDEV_ENUMSTD` `ioctl` with a pointer to this structure. Drivers fill the rest of the structure or return an `EINVAL` error code when the index is out of bounds. To enumerate all standards applications shall begin at index zero, incrementing by one until the driver returns `EINVAL`. Drivers may enumerate a different set of standards after switching the video input or output.¹

v4l2_standard

Table 176: struct v4l2_standard

<code>__u32</code>	<code>index</code>	Number of the video standard, set by the application.
<code>v4l2_std_id</code>	<code>id</code>	The bits in this field identify the standard as one of the common standards listed in <code>typedef v4l2_std_id</code> , or if bits 32 to 63 are set as custom standards. Multiple bits can be set if the hardware does not distinguish between these standards, however separate indices do not indicate the opposite. The <code>id</code> must be unique. No other enumerated <code>struct v4l2_standard</code> structure, for this input or output anyway, can contain the same set of bits.
<code>__u8</code>	<code>name[24]</code>	Name of the standard, a NUL-terminated ASCII string, for example: "PAL-B/G", "NTSC Japan". This information is intended for the user.
<code>struct v4l2_fract</code>	<code>frameperiod</code>	The frame period (not field period) is numerator / denominator. For example M/NTSC has a frame period of 1001 / 30000 seconds.
<code>__u32</code>	<code>framelines</code>	Total lines per frame including blanking, e. g. 625 for B/PAL.
<code>__u32</code>	<code>reserved[4]</code>	Reserved for future extensions. Drivers must set the array to zero.

v4l2_fract

Table 177: struct v4l2_fract

<code>__u32</code>	<code>numerator</code>	
<code>__u32</code>	<code>denominator</code>	

¹ The supported standards may overlap and we need an unambiguous set to find the current standard returned by `VIDIOC_G_STD`.

Table 178: typedef v4l2_std_id

<code>__u64</code>	<code>v4l2_std_id</code>	This type is a set, each bit representing another video standard as listed below and in <i>Video Standards (based on itu470)</i> . The 32 most significant bits are reserved for custom (driver defined) video standards.
--------------------	--------------------------	---

```
#define V4L2_STD_PAL_B      ((v4l2_std_id)0x00000001)
#define V4L2_STD_PAL_B1    ((v4l2_std_id)0x00000002)
#define V4L2_STD_PAL_G     ((v4l2_std_id)0x00000004)
#define V4L2_STD_PAL_H     ((v4l2_std_id)0x00000008)
#define V4L2_STD_PAL_I     ((v4l2_std_id)0x00000010)
#define V4L2_STD_PAL_D     ((v4l2_std_id)0x00000020)
#define V4L2_STD_PAL_D1    ((v4l2_std_id)0x00000040)
#define V4L2_STD_PAL_K     ((v4l2_std_id)0x00000080)

#define V4L2_STD_PAL_M     ((v4l2_std_id)0x00000100)
#define V4L2_STD_PAL_N     ((v4l2_std_id)0x00000200)
#define V4L2_STD_PAL_Nc    ((v4l2_std_id)0x00000400)
#define V4L2_STD_PAL_60    ((v4l2_std_id)0x00000800)
```

V4L2_STD_PAL_60 is a hybrid standard with 525 lines, 60 Hz refresh rate, and PAL color modulation with a 4.43 MHz color subcarrier. Some PAL video recorders can play back NTSC tapes in this mode for display on a 50/60 Hz agnostic PAL TV.

```
#define V4L2_STD_NTSC_M     ((v4l2_std_id)0x00001000)
#define V4L2_STD_NTSC_M_JP ((v4l2_std_id)0x00002000)
#define V4L2_STD_NTSC_443  ((v4l2_std_id)0x00004000)
```

V4L2_STD_NTSC_443 is a hybrid standard with 525 lines, 60 Hz refresh rate, and NTSC color modulation with a 4.43 MHz color subcarrier.

```
#define V4L2_STD_NTSC_M_KR  ((v4l2_std_id)0x00008000)

#define V4L2_STD_SECAM_B    ((v4l2_std_id)0x00010000)
#define V4L2_STD_SECAM_D    ((v4l2_std_id)0x00020000)
#define V4L2_STD_SECAM_G    ((v4l2_std_id)0x00040000)
#define V4L2_STD_SECAM_H    ((v4l2_std_id)0x00080000)
#define V4L2_STD_SECAM_K    ((v4l2_std_id)0x00100000)
#define V4L2_STD_SECAM_K1   ((v4l2_std_id)0x00200000)
#define V4L2_STD_SECAM_L    ((v4l2_std_id)0x00400000)
#define V4L2_STD_SECAM_LC   ((v4l2_std_id)0x00800000)

/* ATSC/HDTV */
#define V4L2_STD_ATSC_8_VSB  ((v4l2_std_id)0x01000000)
#define V4L2_STD_ATSC_16_VSB ((v4l2_std_id)0x02000000)
```

V4L2_STD_ATSC_8_VSB and V4L2_STD_ATSC_16_VSB are U.S. terrestrial digital TV standards. Presently the V4L2 API does not support digital TV. See also the Linux DVB API at <https://linuxtv.org>.

```
#define V4L2_STD_PAL_BG      (V4L2_STD_PAL_B      |
                             V4L2_STD_PAL_B1     |
                             V4L2_STD_PAL_G)
```

```

#define V4L2_STD_B          (V4L2_STD_PAL_B
    V4L2_STD_PAL_B1
    V4L2_STD_SECAM_B)
#define V4L2_STD_GH       (V4L2_STD_PAL_G
    V4L2_STD_PAL_H
    V4L2_STD_SECAM_G
    V4L2_STD_SECAM_H)
#define V4L2_STD_PAL_DK   (V4L2_STD_PAL_D
    V4L2_STD_PAL_D1
    V4L2_STD_PAL_K)
#define V4L2_STD_PAL      (V4L2_STD_PAL_BG
    V4L2_STD_PAL_DK
    V4L2_STD_PAL_H
    V4L2_STD_PAL_I)
#define V4L2_STD_NTSC     (V4L2_STD_NTSC_M
    V4L2_STD_NTSC_M_JP
    V4L2_STD_NTSC_M_KR)
#define V4L2_STD_MN       (V4L2_STD_PAL_M
    V4L2_STD_PAL_N
    V4L2_STD_PAL_Nc
    V4L2_STD_NTSC)
#define V4L2_STD_SECAM_DK (V4L2_STD_SECAM_D
    V4L2_STD_SECAM_K
    V4L2_STD_SECAM_K1)
#define V4L2_STD_DK       (V4L2_STD_PAL_DK
    V4L2_STD_SECAM_DK)

#define V4L2_STD_SECAM    (V4L2_STD_SECAM_B
    V4L2_STD_SECAM_G
    V4L2_STD_SECAM_H
    V4L2_STD_SECAM_DK
    V4L2_STD_SECAM_L
    V4L2_STD_SECAM_LC)

#define V4L2_STD_525_60   (V4L2_STD_PAL_M
    V4L2_STD_PAL_60
    V4L2_STD_NTSC
    V4L2_STD_NTSC_443)
#define V4L2_STD_625_50   (V4L2_STD_PAL
    V4L2_STD_PAL_N
    V4L2_STD_PAL_Nc
    V4L2_STD_SECAM)

#define V4L2_STD_UNKNOWN  0
#define V4L2_STD_ALL      (V4L2_STD_525_60
    V4L2_STD_625_50)

```

Table 179: Video Standards (based on ITU BT.470)

Characteristics	M/NTSC ²	M/PAL	N/PAL ³	B, B1, G/PAL	D, K/PAL	D1	H/PAL	I/PAL	B, G/SECAM	D, K/SECAM	K1/SECAM	L/SECAM
Frame lines	525		625									
Frame period (s)	1001/30000		1/25									
Chrominance sub-carrier frequency (Hz)	3579545 ± 10	3579611.49 ± 10	4433618.75 ± 5 (3582056.25 ± 5)	4433618.75 ± 5					4433618.75 ± 1	f _{OR} = 4406250 ± 2000, f _{OB} = 4250000 ± 2000		
Nominal radio-frequency channel bandwidth (MHz)	6	6	6	B: 7; B1, G: 8	8	8	8	8	8	8	8	8
Sound carrier relative to vision carrier (MHz)	4.5	4.5	4.5	5.5 ± 0.001 ^{4,5,6,7}	6.5 ± 0.001	5.5	5.9996 ± 0.0005	5.5 ± 0.001	6.5 ± 0.001	6.5	6.5	6.5 ⁸

Return Value

On success 0 is returned, on error -1 and the `errno` variable is set appropriately. The generic error codes are described at the [Generic Error Codes](#) chapter.

EINVAL The struct `v4l2_standard` index is out of bounds.

ENODATA Standard video timings are not supported for this input or output.

² Japan uses a standard similar to M/NTSC (V4L2_STD_NTSC_M_JP).

³ The values in brackets apply to the combination N/PAL a.k.a. N_C used in Argentina (V4L2_STD_PAL_Nc).

⁴ In the Federal Republic of Germany, Austria, Italy, the Netherlands, Slovakia and Switzerland a system of two sound carriers is used, the frequency of the second carrier being 242.1875 kHz above the frequency of the first sound carrier. For stereophonic sound transmissions a similar system is used in Australia.

⁵ New Zealand uses a sound carrier displaced 5.4996 ± 0.0005 MHz from the vision carrier.

⁶ In Denmark, Finland, New Zealand, Sweden and Spain a system of two sound carriers is used. In Iceland, Norway and Poland the same system is being introduced. The second carrier is 5.85 MHz above the vision carrier and is DQPSK modulated with 728 kbit/s sound and data multiplex. (NICAM system)

⁷ In the United Kingdom, a system of two sound carriers is used. The second sound carrier is 6.552 MHz above the vision carrier and is DQPSK modulated with a 728 kbit/s sound and data multiplex able to carry two sound channels. (NICAM system)

⁸ In France, a digital carrier 5.85 MHz away from the vision carrier may be used in addition to the main sound carrier. It is modulated in differentially encoded QPSK with a 728 kbit/s sound and data multiplexer capable of carrying two sound channels. (NICAM system)

3.2.7.21 ioctl VIDIOC_EXPBUF

Name

VIDIOC_EXPBUF - Export a buffer as a DMABUF file descriptor.

Synopsis

VIDIOC_EXPBUF

```
int ioctl(int fd, VIDIOC_EXPBUF, struct v4l2_exportbuffer *argp)
```

Arguments

fd File descriptor returned by `open()`.

argp Pointer to struct `v4l2_exportbuffer`.

Description

This `ioctl` is an extension to the *memory mapping* I/O method, therefore it is available only for `V4L2_MEMORY_MMAP` buffers. It can be used to export a buffer as a DMABUF file at any time after buffers have been allocated with the *ioctl VIDIOC_REQBUFS* `ioctl`.

To export a buffer, applications fill struct `v4l2_exportbuffer`. The `type` field is set to the same buffer type as was previously used with struct `v4l2_requestbuffers` type. Applications must also set the `index` field. Valid index numbers range from zero to the number of buffers allocated with *ioctl VIDIOC_REQBUFS* (struct `v4l2_requestbuffers` count) minus one. For the multi-planar API, applications set the `plane` field to the index of the plane to be exported. Valid planes range from zero to the maximal number of valid planes for the currently active format. For the single-planar API, applications must set `plane` to zero. Additional flags may be posted in the `flags` field. Refer to a manual for `open()` for details. Currently only `O_CLOEXEC`, `O_RDONLY`, `O_WRONLY`, and `O_RDWR` are supported. All other fields must be set to zero. In the case of multi-planar API, every plane is exported separately using multiple *ioctl VIDIOC_EXPBUF* calls.

After calling *ioctl VIDIOC_EXPBUF* the `fd` field will be set by a driver. This is a DMABUF file descriptor. The application may pass it to other DMABUF-aware devices. Refer to *DMABUF*

importing for details about importing DMABUF files into V4L2 nodes. It is recommended to close a DMABUF file when it is no longer used to allow the associated memory to be reclaimed.

Examples

```
int buffer_export(int v4lfd, enum v4l2_buf_type bt, int index, int *dmafd)
{
    struct v4l2_exportbuffer expbuf;

    memset(&expbuf, 0, sizeof(expbuf));
    expbuf.type = bt;
    expbuf.index = index;
    if (ioctl(v4lfd, VIDIOC_EXPBUF, &expbuf) == -1) {
        perror("VIDIOC_EXPBUF");
        return -1;
    }

    *dmafd = expbuf.fd;

    return 0;
}
```

```
int buffer_export_mp(int v4lfd, enum v4l2_buf_type bt, int index,
                    int dmafd[], int n_planes)
{
    int i;

    for (i = 0; i < n_planes; ++i) {
        struct v4l2_exportbuffer expbuf;

        memset(&expbuf, 0, sizeof(expbuf));
        expbuf.type = bt;
        expbuf.index = index;
        expbuf.plane = i;
        if (ioctl(v4lfd, VIDIOC_EXPBUF, &expbuf) == -1) {
            perror("VIDIOC_EXPBUF");
            while (i)
                close(dmafd[--i]);
            return -1;
        }
        dmafd[i] = expbuf.fd;
    }

    return 0;
}
```

v4l2_exportbuffer

Table 180: struct v4l2_exportbuffer

__u32	type	Type of the buffer, same as struct v4l2_format type or struct v4l2_requestbuffers type, set by the application. See v4l2_buf_type
__u32	index	Number of the buffer, set by the application. This field is only used for <i>memory mapping</i> I/O and can range from zero to the number of buffers allocated with the <i>ioctl VIDIOC_REQBUFS</i> and/or <i>ioctl VIDIOC_CREATE_BUFS</i> ioctls.
__u32	plane	Index of the plane to be exported when using the multi-planar API. Otherwise this value must be set to zero.
__u32	flags	Flags for the newly created file, currently only O_CLOEXEC, O_RDONLY, O_WRONLY, and O_RDWR are supported, refer to the manual of open() for more details.
__s32	fd	The DMABUF file descriptor associated with a buffer. Set by the driver.
__u32	reserved[11]	Reserved field for future use. Drivers and applications must set the array to zero.

Return Value

On success 0 is returned, on error -1 and the errno variable is set appropriately. The generic error codes are described at the *Generic Error Codes* chapter.

EINVAL A queue is not in MMAP mode or DMABUF exporting is not supported or flags or type or index or plane fields are invalid.

3.2.7.22 ioctl VIDIOC_G_AUDIO, VIDIOC_S_AUDIO

Name

VIDIOC_G_AUDIO - VIDIOC_S_AUDIO - Query or select the current audio input and its attributes

Synopsis

VIDIOC_G_AUDIO

```
int ioctl(int fd, VIDIOC_G_AUDIO, struct v4l2_audio *argp)
```

VIDIOC_S_AUDIO

```
int ioctl(int fd, VIDIOC_S_AUDIO, const struct v4l2_audio *argp)
```

Arguments

fd File descriptor returned by `open()`.

argp Pointer to struct `v4l2_audio`.

Description

To query the current audio input applications zero out the reserved array of a struct `v4l2_audio` and call the `VIDIOC_G_AUDIO` ioctl with a pointer to this structure. Drivers fill the rest of the structure or return an `EINVAL` error code when the device has no audio inputs, or none which combine with the current video input.

Audio inputs have one writable property, the audio mode. To select the current audio input *and* change the audio mode, applications initialize the `index` and `mode` fields, and the reserved array of a struct `v4l2_audio` structure and call the `VIDIOC_S_AUDIO` ioctl. Drivers may switch to a different audio mode if the request cannot be satisfied. However, this is a write-only ioctl, it does not return the actual new audio mode.

v4l2_audio

Table 181: struct `v4l2_audio`

<code>__u32</code>	<code>index</code>	Identifies the audio input, set by the driver or application.
<code>__u8</code>	<code>name[32]</code>	Name of the audio input, a NUL-terminated ASCII string, for example: "Line In". This information is intended for the user, preferably the connector label on the device itself.
<code>__u32</code>	<code>capability</code>	Audio capability flags, see Audio Capability Flags .
<code>__u32</code>	<code>mode</code>	Audio mode flags set by drivers and applications (on <code>VIDIOC_S_AUDIO</code> ioctl), see Audio Mode Flags .
<code>__u32</code>	<code>reserved[2]</code>	Reserved for future extensions. Drivers and applications must set the array to zero.

Table 182: Audio Capability Flags

<code>V4L2_AUDCAP_STEREO</code>	<code>0x00001</code>	This is a stereo input. The flag is intended to automatically disable stereo recording etc. when the signal is always monaural. The API provides no means to detect if stereo is <i>received</i> , unless the audio input belongs to a tuner.
<code>V4L2_AUDCAP_AVL</code>	<code>0x00002</code>	Automatic Volume Level mode is supported.

Table 183: Audio Mode Flags

<code>V4L2_AUDMODE_AVL</code>	<code>0x00001</code>	AVL mode is on.
-------------------------------	----------------------	-----------------

Return Value

On success 0 is returned, on error -1 and the `errno` variable is set appropriately. The generic error codes are described at the [Generic Error Codes](#) chapter.

EINVAL No audio inputs combine with the current video input, or the number of the selected audio input is out of bounds or it does not combine.

3.2.7.23 ioctl VIDIOC_G_AUDOUT, VIDIOC_S_AUDOUT

Name

VIDIOC_G_AUDOUT - VIDIOC_S_AUDOUT - Query or select the current audio output

Synopsis

VIDIOC_G_AUDOUT

```
int ioctl(int fd, VIDIOC_G_AUDOUT, struct v4l2_audioout *argp)
```

VIDIOC_S_AUDOUT

```
int ioctl(int fd, VIDIOC_S_AUDOUT, const struct v4l2_audioout *argp)
```

Arguments

fd File descriptor returned by `open()`.

argp Pointer to struct `v4l2_audioout`.

Description

To query the current audio output applications zero out the reserved array of a struct `v4l2_audioout` and call the `VIDIOC_G_AUDOUT` ioctl with a pointer to this structure. Drivers fill the rest of the structure or return an `EINVAL` error code when the device has no audio inputs, or none which combine with the current video output.

Audio outputs have no writable properties. Nevertheless, to select the current audio output applications can initialize the `index` field and reserved array (which in the future may contain writable properties) of a struct `v4l2_audioout` structure and call the `VIDIOC_S_AUDOUT` ioctl. Drivers switch to the requested output or return the `EINVAL` error code when the index is out of bounds. This is a write-only ioctl, it does not return the current audio output attributes as `VIDIOC_G_AUDOUT` does.

Note: Connectors on a TV card to loop back the received audio signal to a sound card are not audio outputs in this sense.

`v4l2_audioout`

Table 184: struct v4l2_audioout

__u32	index	Identifies the audio output, set by the driver or application.
__u8	name[32]	Name of the audio output, a NUL-terminated ASCII string, for example: "Line Out". This information is intended for the user, preferably the connector label on the device itself.
__u32	capability	Audio capability flags, none defined yet. Drivers must set this field to zero.
__u32	mode	Audio mode, none defined yet. Drivers and applications (on VIDIOC_S_AUDOUT) must set this field to zero.
__u32	reserved[2]	Reserved for future extensions. Drivers and applications must set the array to zero.

Return Value

On success 0 is returned, on error -1 and the `errno` variable is set appropriately. The generic error codes are described at the [Generic Error Codes](#) chapter.

EINVAL No audio outputs combine with the current video output, or the number of the selected audio output is out of bounds or it does not combine.

3.2.7.24 ioctl VIDIOC_G_CROP, VIDIOC_S_CROP

Name

VIDIOC_G_CROP - VIDIOC_S_CROP - Get or set the current cropping rectangle

Synopsis

VIDIOC_G_CROP

```
int ioctl(int fd, VIDIOC_G_CROP, struct v4l2_crop *argp)
```

VIDIOC_S_CROP

```
int ioctl(int fd, VIDIOC_S_CROP, const struct v4l2_crop *argp)
```

Arguments

fd File descriptor returned by `open()`.

argp Pointer to struct `v4l2_crop`.

Description

To query the cropping rectangle size and position applications set the `type` field of a struct `v4l2_crop` structure to the respective buffer (stream) type and call the `VIDIOC_G_CROP` ioctl with a pointer to this structure. The driver fills the rest of the structure or returns the `EINVAL` error code if cropping is not supported.

To change the cropping rectangle applications initialize the `type` and struct `v4l2_rect` sub-structure named `c` of a `v4l2_crop` structure and call the `VIDIOC_S_CROP` ioctl with a pointer to this structure.

The driver first adjusts the requested dimensions against hardware limits, i. e. the bounds given by the capture/output window, and it rounds to the closest possible values of horizontal and vertical offset, width and height. In particular the driver must round the vertical offset of the cropping rectangle to frame lines modulo two, such that the field order cannot be confused.

Second the driver adjusts the image size (the opposite rectangle of the scaling process, source or target depending on the data direction) to the closest size possible while maintaining the current horizontal and vertical scaling factor.

Finally the driver programs the hardware with the actual cropping and image parameters. `VIDIOC_S_CROP` is a write-only ioctl, it does not return the actual parameters. To query them applications must call `VIDIOC_G_CROP` and `ioctl VIDIOC_G_FMT, VIDIOC_S_FMT, VIDIOC_TRY_FMT`. When the parameters are unsuitable the application may modify the cropping or image parameters and repeat the cycle until satisfactory parameters have been negotiated.

When cropping is not supported then no parameters are changed and `VIDIOC_S_CROP` returns the `EINVAL` error code.

v4l2_crop

Table 185: struct `v4l2_crop`

<code>__u32</code>	<code>type</code>	Type of the data stream, set by the application. Only these types are valid here: <code>V4L2_BUF_TYPE_VIDEO_CAPTURE</code> , <code>V4L2_BUF_TYPE_VIDEO_CAPTURE_MPLANE</code> , <code>V4L2_BUF_TYPE_VIDEO_OUTPUT</code> , <code>V4L2_BUF_TYPE_VIDEO_OUTPUT_MPLANE</code> and <code>V4L2_BUF_TYPE_VIDEO_OVERLAY</code> . See <code>v4l2_buf_type</code> and the note below.
<code>struct v4l2_rect</code>	<code>c</code>	Cropping rectangle. The same co-ordinate system as for struct <code>v4l2_cropcap</code> bounds is used.

Note: Unfortunately in the case of multiplanar buffer types (`V4L2_BUF_TYPE_VIDEO_CAPTURE_MPLANE` and `V4L2_BUF_TYPE_VIDEO_OUTPUT_MPLANE`) this API

was messed up with regards to how the `v4l2_crop` type field should be filled in. Some drivers only accepted the `_MPLANE` buffer type while other drivers only accepted a non-multiplanar buffer type (i.e. without the `_MPLANE` at the end).

Starting with kernel 4.13 both variations are allowed.

Return Value

On success 0 is returned, on error -1 and the `errno` variable is set appropriately. The generic error codes are described at the [Generic Error Codes](#) chapter.

ENODATA Cropping is not supported for this input or output.

3.2.7.25 ioctl VIDIOC_G_CTRL, VIDIOC_S_CTRL

Name

VIDIOC_G_CTRL - VIDIOC_S_CTRL - Get or set the value of a control

Synopsis

VIDIOC_G_CTRL

```
int ioctl(int fd, VIDIOC_G_CTRL, struct v4l2_control *argp)
```

VIDIOC_S_CTRL

```
int ioctl(int fd, VIDIOC_S_CTRL, struct v4l2_control *argp)
```

Arguments

fd File descriptor returned by `open()`.

argp Pointer to `struct v4l2_control`.

Description

To get the current value of a control applications initialize the `id` field of a `struct v4l2_control` and call the `VIDIOC_G_CTRL` ioctl with a pointer to this structure. To change the value of a control applications initialize the `id` and `value` fields of a `struct v4l2_control` and call the `VIDIOC_S_CTRL` ioctl.

When the `id` is invalid drivers return an `EINVAL` error code. When the `value` is out of bounds drivers can choose to take the closest valid value or return an `ERANGE` error code, whatever seems more appropriate. However, `VIDIOC_S_CTRL` is a write-only ioctl, it does not return the actual new value. If the `value` is inappropriate for the control (e.g. if it refers to an unsupported menu index of a menu control), then `EINVAL` error code is returned as well.

These ioctls work only with user controls. For other control classes the `VIDIOC_G_EXT_CTRL`s, `VIDIOC_S_EXT_CTRL`s or `VIDIOC_TRY_EXT_CTRL`s must be used.

v4l2_control

Table 186: struct v4l2_control

<code>__u32</code>	<code>id</code>	Identifies the control, set by the application.
<code>__s32</code>	<code>value</code>	New value or current value.

Return Value

On success 0 is returned, on error -1 and the `errno` variable is set appropriately. The generic error codes are described at the [Generic Error Codes](#) chapter.

EINVAL The struct `v4l2_control` `id` is invalid or the `value` is inappropriate for the given control (i.e. if a menu item is selected that is not supported by the driver according to [VIDIOC_QUERYMENU](#)).

ERANGE The struct `v4l2_control` `value` is out of bounds.

EBUSY The control is temporarily not changeable, possibly because another applications took over control of the device function this control belongs to.

EACCES Attempt to set a read-only control or to get a write-only control.

Or if there is an attempt to set an inactive control and the driver is not capable of caching the new value until the control is active again.

3.2.7.26 ioctl VIDIOC_G_DV_TIMINGS, VIDIOC_S_DV_TIMINGS

Name

VIDIOC_G_DV_TIMINGS - VIDIOC_S_DV_TIMINGS - VIDIOC_SUBDEV_G_DV_TIMINGS - VIDIOC_SUBDEV_S_DV_TIMINGS - Get or set DV timings for input or output

Synopsis

VIDIOC_G_DV_TIMINGS

```
int ioctl(int fd, VIDIOC_G_DV_TIMINGS, struct v4l2_dv_timings *argp)
```

VIDIOC_S_DV_TIMINGS

```
int ioctl(int fd, VIDIOC_S_DV_TIMINGS, struct v4l2_dv_timings *argp)
```

VIDIOC_SUBDEV_G_DV_TIMINGS

```
int ioctl(int fd, VIDIOC_SUBDEV_G_DV_TIMINGS, struct v4l2_dv_timings *argp)
```

VIDIOC_SUBDEV_S_DV_TIMINGS

```
int ioctl(int fd, VIDIOC_SUBDEV_S_DV_TIMINGS, struct v4l2_dv_timings *argp)
```


Arguments

fd File descriptor returned by `open()`.

argp Pointer to struct `v4l2_dv_timings`.

Description

To set DV timings for the input or output, applications use the `VIDIOC_S_DV_TIMINGS` ioctl and to get the current timings, applications use the `VIDIOC_G_DV_TIMINGS` ioctl. The detailed timing information is filled in using the structure struct `v4l2_dv_timings`. These ioctls take a pointer to the struct `v4l2_dv_timings` structure as argument. If the ioctl is not supported or the timing values are not correct, the driver returns `EINVAL` error code.

Calling `VIDIOC_SUBDEV_S_DV_TIMINGS` on a subdev device node that has been registered in read-only mode is not allowed. An error is returned and the `errno` variable is set to `-EPERM`.

The `linux/v4l2-dv-timings.h` header can be used to get the timings of the formats in the *CEA-861-E* and *VESA DMT* standards. If the current input or output does not support DV timings (e.g. if ioctl `VIDIOC_ENUMINPUT` does not set the `V4L2_IN_CAP_DV_TIMINGS` flag), then `ENODATA` error code is returned.

Return Value

On success 0 is returned, on error -1 and the `errno` variable is set appropriately. The generic error codes are described at the *Generic Error Codes* chapter.

EINVAL This ioctl is not supported, or the `VIDIOC_S_DV_TIMINGS` parameter was unsuitable.

ENODATA Digital video timings are not supported for this input or output.

EBUSY The device is busy and therefore can not change the timings.

EPERM `VIDIOC_SUBDEV_S_DV_TIMINGS` has been called on a read-only subdevice.

`v4l2_bt_timings`

Table 187: struct `v4l2_bt_timings`

<code>__u32</code>	<code>width</code>	Width of the active video in pixels.
<code>__u32</code>	<code>height</code>	Height of the active video frame in lines. So for interlaced formats the height of the active video in each field is <code>height/2</code> .
<code>__u32</code>	<code>interlaced</code>	Progressive (<code>V4L2_DV_PROGRESSIVE</code>) or interlaced (<code>V4L2_DV_INTERLACED</code>).
<code>__u32</code>	<code>polarities</code>	This is a bit mask that defines polarities of sync signals. bit 0 (<code>V4L2_DV_VSYNC_POS_POL</code>) is for vertical sync polarity and bit 1 (<code>V4L2_DV_HSYNC_POS_POL</code>) is for horizontal sync polarity. If the bit is set (1) it is positive polarity and if is cleared (0), it is negative polarity.

Continued on next page

Table 187 - continued from previous page

__u64	pixelclock	Pixel clock in Hz. Ex. 74.25MHz->74250000
__u32	hfrontporch	Horizontal front porch in pixels
__u32	hsync	Horizontal sync length in pixels
__u32	hbackporch	Horizontal back porch in pixels
__u32	vfrontporch	Vertical front porch in lines. For interlaced formats this refers to the odd field (aka field 1).
__u32	vsync	Vertical sync length in lines. For interlaced formats this refers to the odd field (aka field 1).
__u32	vbackporch	Vertical back porch in lines. For interlaced formats this refers to the odd field (aka field 1).
__u32	il_vfrontporch	Vertical front porch in lines for the even field (aka field 2) of interlaced field formats. Must be 0 for progressive formats.
__u32	il_vsync	Vertical sync length in lines for the even field (aka field 2) of interlaced field formats. Must be 0 for progressive formats.
__u32	il_vbackporch	Vertical back porch in lines for the even field (aka field 2) of interlaced field formats. Must be 0 for progressive formats.
__u32	standards	The video standard(s) this format belongs to. This will be filled in by the driver. Applications must set this to 0. See <i>DV BT Timing standards</i> for a list of standards.
__u32	flags	Several flags giving more information about the format. See <i>DV BT Timing flags</i> for a description of the flags.
struct v4l2_fract	picture_aspect	The picture aspect if the pixels are not square. Only valid if the V4L2_DV_FL_HAS_PICTURE_ASPECT flag is set.
__u8	cea861_vic	The Video Identification Code according to the CEA-861 standard. Only valid if the V4L2_DV_FL_HAS_CEA861_VIC flag is set.
__u8	hdmi_vic	The Video Identification Code according to the HDMI standard. Only valid if the V4L2_DV_FL_HAS_HDMI_VIC flag is set.
__u8	reserved[46]	Reserved for future extensions. Drivers and applications must set the array to zero.

v4l2_dv_timings

Table 188: struct v4l2_dv_timings

__u32	type	Type of DV timings as listed in <i>DV Timing types</i> .
union {	(anonymous)	
struct v4l2_bt_timings	bt	Timings defined by BT.656/1120 specifications
__u32	reserved[32]	
}		

Table 189: DV Timing types

Timing type	value	Description
V4L2_DV_BT_656_1120	0	BT.656/1120 timings

Table 190: DV BT Timing standards

Timing standard	Description
V4L2_DV_BT_STD_CEA861	The timings follow the CEA-861 Digital TV Profile standard
V4L2_DV_BT_STD_DMT	The timings follow the VESA Discrete Monitor Timings standard
V4L2_DV_BT_STD_CVT	The timings follow the VESA Coordinated Video Timings standard
V4L2_DV_BT_STD_GTF	The timings follow the VESA Generalized Timings Formula standard
V4L2_DV_BT_STD_SDI	The timings follow the SDI Timings standard. There are no horizontal syncs/porches at all in this format. Total blanking timings must be set in hsync or vsync fields only.

Table 191: DV BT Timing flags

Flag	Description
V4L2_DV_FL_REduced_BLANKING	CVT/GTF specific: the timings use reduced blanking (CVT) or the 'Secondary GTF' curve (GTF). In both cases the horizontal and/or vertical blanking intervals are reduced, allowing a higher resolution over the same bandwidth. This is a read-only flag, applications must not set this.
V4L2_DV_FL_CAN_REDUCE_FPS	CEA-861 specific: set for CEA-861 formats with a framerate that is a multiple of six. These formats can be optionally played at 1 / 1.001 speed to be compatible with 60 Hz based standards such as NTSC and PAL-M that use a framerate of 29.97 frames per second. If the transmitter can't generate such frequencies, then the flag will also be cleared. This is a read-only flag, applications must not set this.

Continued on next page

Table 191 - continued from previous page

V4L2_DV_FL_REDUCED_FPS	<p>CEA-861 specific: only valid for video transmitters or video receivers that have the V4L2_DV_FL_CAN_DETECT_REDUCED_FPS set. This flag is cleared otherwise. It is also only valid for formats with the V4L2_DV_FL_CAN_REDUCE_FPS flag set, for other formats the flag will be cleared by the driver.</p> <p>If the application sets this flag for a transmitter, then the pixelclock used to set up the transmitter is divided by 1.001 to make it compatible with NTSC framerates. If the transmitter can't generate such frequencies, then the flag will be cleared.</p> <p>If a video receiver detects that the format uses a reduced framerate, then it will set this flag to signal this to the application.</p>
V4L2_DV_FL_HALF_LINE	<p>Specific to interlaced formats: if set, then the vertical frontporch of field 1 (aka the odd field) is really one half-line longer and the vertical backporch of field 2 (aka the even field) is really one half-line shorter, so each field has exactly the same number of half-lines. Whether half-lines can be detected or used depends on the hardware.</p>
V4L2_DV_FL_IS_CE_VIDEO	<p>If set, then this is a Consumer Electronics (CE) video format. Such formats differ from other formats (commonly called IT formats) in that if R'G'B' encoding is used then by default the R'G'B' values use limited range (i.e. 16-235) as opposed to full range (i.e. 0-255). All formats defined in CEA-861 except for the 640x480p59.94 format are CE formats.</p>
V4L2_DV_FL_FIRST_FIELD_EXTRA_LINE	<p>Some formats like SMPTE-125M have an interlaced signal with a odd total height. For these formats, if this flag is set, the first field has the extra line. Else, it is the second field.</p>
V4L2_DV_FL_HAS_PICTURE_ASPECT	<p>If set, then the picture_aspect field is valid. Otherwise assume that the pixels are square, so the picture aspect ratio is the same as the width to height ratio.</p>
V4L2_DV_FL_HAS_CEA861_VIC	<p>If set, then the cea861_vic field is valid and contains the Video Identification Code as per the CEA-861 standard.</p>
V4L2_DV_FL_HAS_HDMI_VIC	<p>If set, then the hdmi_vic field is valid and contains the Video Identification Code as per the HDMI standard (HDMI Vendor Specific InfoFrame).</p>

Continued on next page

Table 191 - continued from previous page

V4L2_DV_FL_CAN_DETECT_REDUCED_FPS	CEA-861 specific: only valid for video receivers, the flag is cleared by transmitters. If set, then the hardware can detect the difference between regular framerates and framerates reduced by 1000/1001. E.g.: 60 vs 59.94 Hz, 30 vs 29.97 Hz or 24 vs 23.976 Hz.
-----------------------------------	---

3.2.7.27 `ioctl VIDIOC_G_EDID, VIDIOC_S_EDID, VIDIOC_SUBDEV_G_EDID, VIDIOC_SUBDEV_S_EDID`

Name

VIDIOC_G_EDID - VIDIOC_S_EDID - VIDIOC_SUBDEV_G_EDID - VIDIOC_SUBDEV_S_EDID - Get or set the EDID of a video receiver/transmitter

Synopsis

VIDIOC_G_EDID

```
int ioctl(int fd, VIDIOC_G_EDID, struct v4l2_edid *argp)
```

VIDIOC_S_EDID

```
int ioctl(int fd, VIDIOC_S_EDID, struct v4l2_edid *argp)
```

VIDIOC_SUBDEV_G_EDID

```
int ioctl(int fd, VIDIOC_SUBDEV_G_EDID, struct v4l2_edid *argp)
```

VIDIOC_SUBDEV_S_EDID

```
int ioctl(int fd, VIDIOC_SUBDEV_S_EDID, struct v4l2_edid *argp)
```

Arguments

fd File descriptor returned by `open()`.

argp Pointer to struct `v4l2_edid`.

Description

These `ioctl`s can be used to get or set an EDID associated with an input from a receiver or an output of a transmitter device. They can be used with subdevice nodes (`/dev/v4l-subdevX`) or with video nodes (`/dev/videoX`).

When used with video nodes the `pad` field represents the input (for video capture devices) or output (for video output devices) index as is returned by `ioctl VIDIOC_ENUMINPUT` and `ioctl VIDIOC_ENUMOUTPUT` respectively. When used with subdevice nodes the `pad` field represents the input or output pad of the subdevice. If there is no EDID support for the given `pad` value, then the `EINVAL` error code will be returned.

To get the EDID data the application has to fill in the `pad`, `start_block`, `blocks` and `edid` fields, zero the reserved array and call `VIDIOC_G_EDID`. The current EDID from block `start_block` and of size `blocks` will be placed in the memory `edid` points to. The `edid` pointer must point to memory at least `blocks * 128` bytes large (the size of one block is 128 bytes).

If there are fewer blocks than specified, then the driver will set `blocks` to the actual number of blocks. If there are no EDID blocks available at all, then the error code `ENODATA` is set.

If blocks have to be retrieved from the sink, then this call will block until they have been read.

If `start_block` and `blocks` are both set to 0 when `VIDIOC_G_EDID` is called, then the driver will set `blocks` to the total number of available EDID blocks and it will return 0 without copying any data. This is an easy way to discover how many EDID blocks there are.

Note: If there are no EDID blocks available at all, then the driver will set `blocks` to 0 and it returns 0.

To set the EDID blocks of a receiver the application has to fill in the `pad`, `blocks` and `edid` fields, set `start_block` to 0 and zero the reserved array. It is not possible to set part of an EDID, it is always all or nothing. Setting the EDID data is only valid for receivers as it makes no sense for a transmitter.

The driver assumes that the full EDID is passed in. If there are more EDID blocks than the hardware can handle then the EDID is not written, but instead the error code `E2BIG` is set and `blocks` is set to the maximum that the hardware supports. If `start_block` is any value other than 0 then the error code `EINVAL` is set.

To disable an EDID you set `blocks` to 0. Depending on the hardware this will drive the hotplug pin low and/or block the source from reading the EDID data in some way. In any case, the end result is the same: the EDID is no longer available.

v4l2_edid

Table 192: struct `v4l2_edid`

<code>__u32</code>	<code>pad</code>	Pad for which to get/set the EDID blocks. When used with a video device node the <code>pad</code> represents the input or output index as returned by <code>ioctl VIDIOC_ENUMINPUT</code> and <code>ioctl VIDIOC_ENUMOUTPUT</code> respectively.
<code>__u32</code>	<code>start_block</code>	Read the EDID from starting with this block. Must be 0 when setting the EDID.
<code>__u32</code>	<code>blocks</code>	The number of blocks to get or set. Must be less or equal to 256 (the maximum number of blocks as defined by the standard). When you set the EDID and <code>blocks</code> is 0, then the EDID is disabled or erased.
<code>__u32</code>	<code>reserved[5]</code>	Reserved for future extensions. Applications and drivers must set the array to zero.
<code>__u8 *</code>	<code>edid</code>	Pointer to memory that contains the EDID. The minimum size is <code>blocks * 128</code> .

Return Value

On success 0 is returned, on error -1 and the `errno` variable is set appropriately. The generic error codes are described at the [Generic Error Codes](#) chapter.

ENODATA The EDID data is not available.

E2BIG The EDID data you provided is more than the hardware can handle.

3.2.7.28 ioctl VIDIOC_G_ENC_INDEX

Name

VIDIOC_G_ENC_INDEX - Get meta data about a compressed video stream

Synopsis

VIDIOC_G_ENC_INDEX

```
int ioctl(int fd, VIDIOC_G_ENC_INDEX, struct v4l2_enc_idx *argp)
```

Arguments

fd File descriptor returned by `open()`.

argp Pointer to struct `v4l2_enc_idx`.

Description

The `VIDIOC_G_ENC_INDEX` ioctl provides meta data about a compressed video stream the same or another application currently reads from the driver, which is useful for random access into the stream without decoding it.

To read the data applications must call `VIDIOC_G_ENC_INDEX` with a pointer to a struct `v4l2_enc_idx`. On success the driver fills the entry array, stores the number of elements written in the `entries` field, and initializes the `entries_cap` field.

Each element of the entry array contains meta data about one picture. A `VIDIOC_G_ENC_INDEX` call reads up to `V4L2_ENC_IDX_ENTRIES` entries from a driver buffer, which can hold up to `entries_cap` entries. This number can be lower or higher than `V4L2_ENC_IDX_ENTRIES`, but not zero. When the application fails to read the meta data in time the oldest entries will be lost. When the buffer is empty or no capturing/encoding is in progress, entries will be zero.

Currently this ioctl is only defined for MPEG-2 program streams and video elementary streams.

`v4l2_enc_idx`

Table 193: struct v4l2_enc_idx

__u32	entries	The number of entries the driver stored in the entry array.
__u32	entries_cap	The number of entries the driver can buffer. Must be greater than zero.
__u32	reserved[4]	Reserved for future extensions. Drivers must set the array to zero.
struct v4l2_enc_idx_entry	entry[V4L2_ENC_IDX_ENTRIES]	Meta data about a compressed video stream. Each element of the array corresponds to one picture, sorted in ascending order by their offset.

v4l2_enc_idx_entry

Table 194: struct v4l2_enc_idx_entry

__u64	offset	The offset in bytes from the beginning of the compressed video stream to the beginning of this picture, that is a <i>PES packet header</i> as defined in <i>ISO 13818-1</i> or a <i>picture header</i> as defined in <i>ISO 13818-2</i> . When the encoder is stopped, the driver resets the offset to zero.
__u64	pts	The 33 bit <i>Presentation Time Stamp</i> of this picture as defined in <i>ISO 13818-1</i> .
__u32	length	The length of this picture in bytes.
__u32	flags	Flags containing the coding type of this picture, see <i>Index Entry Flags</i> .
__u32	reserved[2]	Reserved for future extensions. Drivers must set the array to zero.

Table 195: Index Entry Flags

V4L2_ENC_IDX_FRAME_I	0x00	This is an Intra-coded picture.
V4L2_ENC_IDX_FRAME_P	0x01	This is a Predictive-coded picture.
V4L2_ENC_IDX_FRAME_B	0x02	This is a Bidirectionally predictive-coded picture.
V4L2_ENC_IDX_FRAME_MASK	0x0F	AND the flags field with this mask to obtain the picture coding type.

Return Value

On success 0 is returned, on error -1 and the `errno` variable is set appropriately. The generic error codes are described at the [Generic Error Codes](#) chapter.

3.2.7.29 ioctl VIDIOC_G_EXT_CTRLs, VIDIOC_S_EXT_CTRLs, VIDIOC_TRY_EXT_CTRLs

Name

VIDIOC_G_EXT_CTRLs - VIDIOC_S_EXT_CTRLs - VIDIOC_TRY_EXT_CTRLs - Get or set the value of several controls, try control values

Synopsis

VIDIOC_G_EXT_CTRLs

```
int ioctl(int fd, VIDIOC_G_EXT_CTRLs, struct v4l2_ext_controls *argp)
```

VIDIOC_S_EXT_CTRLs

```
int ioctl(int fd, VIDIOC_S_EXT_CTRLs, struct v4l2_ext_controls *argp)
```

VIDIOC_TRY_EXT_CTRLs

```
int ioctl(int fd, VIDIOC_TRY_EXT_CTRLs, struct v4l2_ext_controls *argp)
```

Arguments

fd File descriptor returned by `open()`.

argp Pointer to struct `v4l2_ext_controls`.

Description

These `ioctl`s allow the caller to get or set multiple controls atomically. Control IDs are grouped into control classes (see [Control classes](#)) and all controls in the control array must belong to the same control class.

Applications must always fill in the `count`, `which`, `controls` and `reserved` fields of struct `v4l2_ext_controls`, and initialize the struct `v4l2_ext_control` array pointed to by the `controls` fields.

To get the current value of a set of controls applications initialize the `id`, `size` and `reserved2` fields of each struct `v4l2_ext_control` and call the `VIDIOC_G_EXT_CTRLs` `ioctl`. String controls must also set the `string` field. Controls of compound types (`V4L2_CTRL_FLAG_HAS_PAYLOAD` is set) must set the `ptr` field.

If the `size` is too small to receive the control result (only relevant for pointer-type controls like strings), then the driver will set `size` to a valid value and return an `ENOSPC` error code. You should re-allocate the memory to this new size and try again. For the string type it is possible that the same issue occurs again if the string has grown in the meantime. It is recommended to

call *ioctl*s `VIDIOC_QUERYCTRL`, `VIDIOC_QUERY_EXT_CTRL` and `VIDIOC_QUERYMENU` first and use `maximum+1` as the new size value. It is guaranteed that that is sufficient memory.

N-dimensional arrays are set and retrieved row-by-row. You cannot set a partial array, all elements have to be set or retrieved. The total size is calculated as `elems * elem_size`. These values can be obtained by calling `VIDIOC_QUERY_EXT_CTRL`.

To change the value of a set of controls applications initialize the `id`, `size`, `reserved2` and `value/value64/string/ptr` fields of each struct `v4l2_ext_control` and call the `VIDIOC_S_EXT_CTRL` *ioctl*. The controls will only be set if *all* control values are valid.

To check if a set of controls have correct values applications initialize the `id`, `size`, `reserved2` and `value/value64/string/ptr` fields of each struct `v4l2_ext_control` and call the `VIDIOC_TRY_EXT_CTRL` *ioctl*. It is up to the driver whether wrong values are automatically adjusted to a valid value or if an error is returned.

When the `id` or which is invalid drivers return an `EINVAL` error code. When the value is out of bounds drivers can choose to take the closest valid value or return an `ERANGE` error code, whatever seems more appropriate. In the first case the new value is set in struct `v4l2_ext_control`. If the new control value is inappropriate (e.g. the given menu index is not supported by the menu control), then this will also result in an `EINVAL` error code error.

If `request_fd` is set to a not-yet-queued *request* file descriptor and which is set to `V4L2_CTRL_WHICH_REQUEST_VAL`, then the controls are not applied immediately when calling `VIDIOC_S_EXT_CTRL`, but instead are applied by the driver for the buffer associated with the same request. If the device does not support requests, then `EACCES` will be returned. If requests are supported but an invalid request file descriptor is given, then `EINVAL` will be returned.

An attempt to call `VIDIOC_S_EXT_CTRL` for a request that has already been queued will result in an `EBUSY` error.

If `request_fd` is specified and which is set to `V4L2_CTRL_WHICH_REQUEST_VAL` during a call to `VIDIOC_G_EXT_CTRL`, then it will return the values of the controls at the time of request completion. If the request is not yet completed, then this will result in an `EACCES` error.

The driver will only set/get these controls if all control values are correct. This prevents the situation where only some of the controls were set/get. Only low-level errors (e. g. a failed `i2c` command) can still cause this situation.

v4l2_ext_control

Table 196: struct `v4l2_ext_control`

<code>__u32</code>	<code>id</code>	Identifies the control, set by the application.
<code>__u32</code>	<code>size</code>	The total size in bytes of the payload of this control.
The <code>size</code> field is normally 0, but for pointer controls this should be set to the size of the memory that contains the payload or that will receive the payload. If <code>VIDIOC_G_EXT_CTRL</code> finds that this value is less than is required to store the payload result, then it is set to a value large enough to store the payload result and <code>ENOSPC</code> is returned.		
Note: For string controls, this <code>size</code> field should not be confused with the length of the string. This field refers to the size of the memory that contains the string. The actual <i>length</i> of the string may well be much smaller.		
<code>__u32</code>	<code>reserved2[1]</code>	Reserved for future extensions. Drivers and applications must set the array to zero.
<code>union {</code>	(anonymous)	

Continued on next page

Table 196 - continued from previous page

<code>__s32</code>	value	New value or current value. Valid if this control is not of type <code>V4L2_CTRL_TYPE_INTEGER64</code> and <code>V4L2_CTRL_FLAG_HAS_PAYLOAD</code> is not set.
<code>__s64</code>	value64	New value or current value. Valid if this control is of type <code>V4L2_CTRL_TYPE_INTEGER64</code> and <code>V4L2_CTRL_FLAG_HAS_PAYLOAD</code> is not set.
<code>char *</code>	string	A pointer to a string. Valid if this control is of type <code>V4L2_CTRL_TYPE_STRING</code> .
<code>__u8 *</code>	<code>p_u8</code>	A pointer to a matrix control of unsigned 8-bit values. Valid if this control is of type <code>V4L2_CTRL_TYPE_U8</code> .
<code>__u16 *</code>	<code>p_u16</code>	A pointer to a matrix control of unsigned 16-bit values. Valid if this control is of type <code>V4L2_CTRL_TYPE_U16</code> .
<code>__u32 *</code>	<code>p_u32</code>	A pointer to a matrix control of unsigned 32-bit values. Valid if this control is of type <code>V4L2_CTRL_TYPE_U32</code> .
<code>struct v4l2_area *</code>	<code>p_area</code>	A pointer to a struct <code>v4l2_area</code> . Valid if this control is of type <code>V4L2_CTRL_TYPE_AREA</code> .
<code>struct v4l2_ctrl_h264_sps *</code>	<code>p_h264_sps</code>	A pointer to a struct <code>v4l2_ctrl_h264_sps</code> . Valid if this control is of type <code>V4L2_CTRL_TYPE_H264_SPS</code> .
<code>struct v4l2_ctrl_h264_pps *</code>	<code>p_h264_pps</code>	A pointer to a struct <code>v4l2_ctrl_h264_pps</code> . Valid if this control is of type <code>V4L2_CTRL_TYPE_H264_PPS</code> .
<code>struct v4l2_ctrl_h264_scaling_matrix *</code>	<code>p_h264_scaling_matrix</code>	A pointer to a struct <code>v4l2_ctrl_h264_scaling_matrix</code> . Valid if this control is of type <code>V4L2_CTRL_TYPE_H264_SCALING_MATRIX</code> .
<code>struct v4l2_ctrl_h264_pred_weights *</code>	<code>p_h264_pred_weights</code>	A pointer to a struct <code>v4l2_ctrl_h264_pred_weights</code> . Valid if this control is of type <code>V4L2_CTRL_TYPE_H264_PRED_WEIGHTS</code> .
<code>struct v4l2_ctrl_h264_slice_params *</code>	<code>p_h264_slice_params</code>	A pointer to a struct <code>v4l2_ctrl_h264_slice_params</code> . Valid if this control is of type <code>V4L2_CTRL_TYPE_H264_SLICE_PARAMS</code> .
<code>struct v4l2_ctrl_h264_decode_params *</code>	<code>p_h264_decode_params</code>	A pointer to a struct <code>v4l2_ctrl_h264_decode_params</code> . Valid if this control is of type <code>V4L2_CTRL_TYPE_H264_DECODE_PARAMS</code> .
<code>struct v4l2_ctrl_fwht_params *</code>	<code>p_fwht_params</code>	A pointer to a struct <code>v4l2_ctrl_fwht_params</code> . Valid if this control is of type <code>V4L2_CTRL_TYPE_FWHT_PARAMS</code> .
<code>struct v4l2_ctrl_vp8_frame *</code>	<code>p_vp8_frame</code>	A pointer to a struct <code>v4l2_ctrl_vp8_frame</code> . Valid if this control is of type <code>V4L2_CTRL_TYPE_VP8_FRAME</code> .
<code>struct v4l2_ctrl_mpeg2_sequence *</code>	<code>p_mpeg2_sequence</code>	A pointer to a struct <code>v4l2_ctrl_mpeg2_sequence</code> . Valid if this control is of type <code>V4L2_CTRL_TYPE_MPEG2_SEQUENCE</code> .

Continued on next page

Table 196 - continued from previous page

struct <i>v4l2_ctrl_mpeg2_picture</i> *	p_mpeg2_picture	A pointer to a struct <i>v4l2_ctrl_mpeg2_picture</i> . Valid if this control is of type V4L2_CTRL_TYPE_MPEG2_PICTURE.
struct <i>v4l2_ctrl_mpeg2_quantisation</i> *	p_mpeg2_quantisation	A pointer to a struct <i>v4l2_ctrl_mpeg2_quantisation</i> . Valid if this control is of type V4L2_CTRL_TYPE_MPEG2_QUANTISATION.
struct <i>v4l2_ctrl_vp9_compressed_hdr</i> *	p_vp9_compressed_hdr	A pointer to a struct <i>v4l2_ctrl_vp9_compressed_hdr</i> . Valid if this control is of type V4L2_CTRL_TYPE_VP9_COMPRESSED_HDR.
struct <i>v4l2_ctrl_vp9_frame</i> *	p_vp9_frame	A pointer to a struct <i>v4l2_ctrl_vp9_frame</i> . Valid if this control is of type V4L2_CTRL_TYPE_VP9_FRAME.
struct <i>v4l2_ctrl_hdr10_cll_info</i> *	p_hdr10_cll	A pointer to a struct <i>v4l2_ctrl_hdr10_cll_info</i> . Valid if this control is of type V4L2_CTRL_TYPE_HDR10_CLL_INFO.
struct <i>v4l2_ctrl_hdr10_mastering_display</i> *	p_hdr10_mastering	A pointer to a struct <i>v4l2_ctrl_hdr10_mastering_display</i> . Valid if this control is of type V4L2_CTRL_TYPE_HDR10_MASTERING_DISPLAY.
void *	ptr	A pointer to a compound type which can be an N-dimensional array and/or a compound type (the control's type is \geq V4L2_CTRL_COMPOUND_TYPES). Valid if V4L2_CTRL_FLAG_HAS_PAYLOAD is set for this control.
}		

v4l2_ext_controls

Table 197: struct v4l2_ext_controls

union {	(anonymous)	
__u32	which	Which value of the control to get/set/try.
<p>V4L2_CTRL_WHICH_CUR_VAL will return the current value of the control, V4L2_CTRL_WHICH_DEF_VAL will return the default value of the control and V4L2_CTRL_WHICH_REQUEST_VAL indicates that these controls have to be retrieved from a request or tried/set for a request. In the latter case the request_fd field contains the file descriptor of the request that should be used. If the device does not support requests, then EACCES will be returned.</p> <p>When using V4L2_CTRL_WHICH_DEF_VAL be aware that you can only get the default value of the control, you cannot set or try it.</p> <p>For backwards compatibility you can also use a control class here (see <i>Control classes</i>). In that case all controls have to belong to that control class. This usage is deprecated, instead just use V4L2_CTRL_WHICH_CUR_VAL. There are some very old drivers that do not yet support V4L2_CTRL_WHICH_CUR_VAL and that require a control class here. You can test for such drivers by setting which to V4L2_CTRL_WHICH_CUR_VAL and calling <i>VID-IOC_TRY_EXT_CTRL</i>s with a count of 0. If that fails, then the driver does not support V4L2_CTRL_WHICH_CUR_VAL.</p>		
__u32	ctrl_class	Deprecated name kept for backwards compatibility. Use which instead.

Continued on next page

Table 197 - continued from previous page

}		
__u32	count	The number of controls in the controls array. May also be zero.
__u32	error_idx	Index of the failing control. Set by the driver in case of an error.
<p>If the error is associated with a particular control, then <code>error_idx</code> is set to the index of that control. If the error is not related to a specific control, or the validation step failed (see below), then <code>error_idx</code> is set to <code>count</code>. The value is undefined if the <code>ioctl</code> returned 0 (success).</p> <p>Before controls are read from/written to hardware a validation step takes place: this checks if all controls in the list are valid controls, if no attempt is made to write to a read-only control or read from a write-only control, and any other up-front checks that can be done without accessing the hardware. The exact validations done during this step are driver dependent since some checks might require hardware access for some devices, thus making it impossible to do those checks up-front. However, drivers should make a best-effort to do as many up-front checks as possible.</p> <p>This check is done to avoid leaving the hardware in an inconsistent state due to easy-to-avoid problems. But it leads to another problem: the application needs to know whether an error came from the validation step (meaning that the hardware was not touched) or from an error during the actual reading from/writing to hardware.</p> <p>The, in hindsight quite poor, solution for that is to set <code>error_idx</code> to <code>count</code> if the validation failed. This has the unfortunate side-effect that it is not possible to see which control failed the validation. If the validation was successful and the error happened while accessing the hardware, then <code>error_idx</code> is less than <code>count</code> and only the controls up to <code>error_idx - 1</code> were read or written correctly, and the state of the remaining controls is undefined.</p> <p>Since <code>VIDIOC_TRY_EXT_CTRL</code>s does not access hardware there is also no need to handle the validation step in this special way, so <code>error_idx</code> will just be set to the control that failed the validation step instead of to <code>count</code>. This means that if <code>VIDIOC_S_EXT_CTRL</code>s fails with <code>error_idx</code> set to <code>count</code>, then you can call <code>VIDIOC_TRY_EXT_CTRL</code>s to try to discover the actual control that failed the validation step. Unfortunately, there is no TRY equivalent for <code>VIDIOC_G_EXT_CTRL</code>s.</p>		
__s32	request_fd	File descriptor of the request to be used by this operation. Only valid if which is set to <code>V4L2_CTRL_WHICH_REQUEST_VAL</code> . If the device does not support requests, then <code>EACCES</code> will be returned. If requests are supported but an invalid request file descriptor is given, then <code>EINVAL</code> will be returned.
__u32	reserved[1]	Reserved for future extensions. Drivers and applications must set the array to zero.
struct v4l2_ext_control *	controls	Pointer to an array of <code>count</code> <code>v4l2_ext_control</code> structures. Ignored if <code>count</code> equals zero.

Table 198: Control classes

V4L2_CTRL_CLASS_USER	0x980000	The class containing user controls. These controls are described in <i>User Controls</i> . All controls that can be set using the <i>VIDIOC_S_CTRL</i> and <i>VIDIOC_G_CTRL</i> ioctl belong to this class.
V4L2_CTRL_CLASS_CODEC	0x990000	The class containing stateful codec controls. These controls are described in <i>Codec Control Reference</i> .
V4L2_CTRL_CLASS_CAMERA	0x9a0000	The class containing camera controls. These controls are described in <i>Camera Control Reference</i> .
V4L2_CTRL_CLASS_FM_TX	0x9b0000	The class containing FM Transmitter (FM TX) controls. These controls are described in <i>FM Transmitter Control Reference</i> .
V4L2_CTRL_CLASS_FLASH	0x9c0000	The class containing flash device controls. These controls are described in <i>Flash Control Reference</i> .
V4L2_CTRL_CLASS_JPEG	0x9d0000	The class containing JPEG compression controls. These controls are described in <i>JPEG Control Reference</i> .
V4L2_CTRL_CLASS_IMAGE_SOURCE	0x9e0000	The class containing image source controls. These controls are described in <i>Image Source Control Reference</i> .
V4L2_CTRL_CLASS_IMAGE_PROC	0x9f0000	The class containing image processing controls. These controls are described in <i>Image Process Control Reference</i> .
V4L2_CTRL_CLASS_FM_RX	0xa10000	The class containing FM Receiver (FM RX) controls. These controls are described in <i>FM Receiver Control Reference</i> .
V4L2_CTRL_CLASS_RF_TUNER	0xa20000	The class containing RF tuner controls. These controls are described in <i>RF Tuner Control Reference</i> .
V4L2_CTRL_CLASS_DETECT	0xa30000	The class containing motion or object detection controls. These controls are described in <i>Detect Control Reference</i> .
V4L2_CTRL_CLASS_CODEC_STATELESS	0xa40000	The class containing stateless codec controls. These controls are described in <i>Stateless Codec Control Reference</i> .
V4L2_CTRL_CLASS_COLORIMETRY	0xa50000	The class containing colorimetry controls. These controls are described in <i>Colorimetry Control Reference</i> .

Return Value

On success 0 is returned, on error -1 and the `errno` variable is set appropriately. The generic error codes are described at the [Generic Error Codes](#) chapter.

EINVAL The struct `v4l2_ext_control` `id` is invalid, or the struct `v4l2_ext_controls` `which` is invalid, or the struct `v4l2_ext_control` `value` was inappropriate (e.g. the given menu index is not supported by the driver), or the `which` field was set to `V4L2_CTRL_WHICH_REQUEST_VAL` but the given `request_fd` was invalid or `V4L2_CTRL_WHICH_REQUEST_VAL` is not supported by the kernel. This error code is also returned by the [VIDIOC_S_EXT_CTRL](#)s and [VIDIOC_TRY_EXT_CTRL](#)s ioctls if two or more control values are in conflict.

ERANGE The struct `v4l2_ext_control` `value` is out of bounds.

EBUSY The control is temporarily not changeable, possibly because another applications took over control of the device function this control belongs to, or (if the `which` field was set to `V4L2_CTRL_WHICH_REQUEST_VAL`) the request was queued but not yet completed.

ENOSPC The space reserved for the control's payload is insufficient. The field `size` is set to a value that is enough to store the payload and this error code is returned.

EACCES Attempt to try or set a read-only control, or to get a write-only control, or to get a control from a request that has not yet been completed.

Or the `which` field was set to `V4L2_CTRL_WHICH_REQUEST_VAL` but the device does not support requests.

Or if there is an attempt to set an inactive control and the driver is not capable of caching the new value until the control is active again.

3.2.7.30 ioctl VIDIOC_G_FBUF, VIDIOC_S_FBUF

Name

VIDIOC_G_FBUF - VIDIOC_S_FBUF - Get or set frame buffer overlay parameters

Synopsis

VIDIOC_G_FBUF

```
int ioctl(int fd, VIDIOC_G_FBUF, struct v4l2_framebuffer *argp)
```

VIDIOC_S_FBUF

```
int ioctl(int fd, VIDIOC_S_FBUF, const struct v4l2_framebuffer *argp)
```


Arguments

fd File descriptor returned by `open()`.

argp Pointer to struct `v4l2_framebuffer`.

Description

Applications can use the `VIDIOC_G_FBUF` and `VIDIOC_S_FBUF` ioctls to get and set the framebuffer parameters for a *Video Overlay* or *Video Output Overlay* (OSD). The type of overlay is implied by the device type (capture or output device) and can be determined with the `ioctl VIDIOC_QUERYCAP` ioctl. One `/dev/videoN` device must not support both kinds of overlay.

The V4L2 API distinguishes destructive and non-destructive overlays. A destructive overlay copies captured video images into the video memory of a graphics card. A non-destructive overlay blends video images into a VGA signal or graphics into a video signal. *Video Output Overlays* are always non-destructive.

To get the current parameters applications call the `VIDIOC_G_FBUF` ioctl with a pointer to a struct `v4l2_framebuffer` structure. The driver fills all fields of the structure or returns an `EINVAL` error code when overlays are not supported.

To set the parameters for a *Video Output Overlay*, applications must initialize the `flags` field of a struct `v4l2_framebuffer`. Since the framebuffer is implemented on the TV card all other parameters are determined by the driver. When an application calls `VIDIOC_S_FBUF` with a pointer to this structure, the driver prepares for the overlay and returns the framebuffer parameters as `VIDIOC_G_FBUF` does, or it returns an error code.

To set the parameters for a *non-destructive Video Overlay*, applications must initialize the `flags` field, the `fmt` substructure, and call `VIDIOC_S_FBUF`. Again the driver prepares for the overlay and returns the framebuffer parameters as `VIDIOC_G_FBUF` does, or it returns an error code.

For a *destructive Video Overlay* applications must additionally provide a base address. Setting up a DMA to a random memory location can jeopardize the system security, its stability or even damage the hardware, therefore only the superuser can set the parameters for a destructive video overlay.

v4l2_framebuffer

Table 199: struct `v4l2_framebuffer`

<code>__u32</code>	<code>capability</code>		Overlay capability flags set by the driver, see <i>Frame Buffer Capability Flags</i> .
<code>__u32</code>	<code>flags</code>		Overlay control flags set by application and driver, see <i>Frame Buffer Flags</i>
<code>void *</code>	<code>base</code>		Physical base address of the framebuffer, that is the address of the pixel in the top left corner of the framebuffer. ¹

Continued on next page

Table 199 - continued from previous page

			This field is irrelevant to <i>non-destructive Video Overlays</i> . For <i>destructive Video Overlays</i> applications must provide a base address. The driver may accept only base addresses which are a multiple of two, four or eight bytes. For <i>Video Output Overlays</i> the driver must return a valid base address, so applications can find the corresponding Linux framebuffer device (see <i>Video Output Overlay Interface</i>).
struct	fmt		Layout of the frame buffer.
	__u32	width	Width of the frame buffer in pixels.
	__u32	height	Height of the frame buffer in pixels.
	__u32	pixelformat	The pixel format of the framebuffer.
			For <i>non-destructive Video Overlays</i> this field only defines a format for the struct <code>v4l2_window</code> chromakey field.
			For <i>destructive Video Overlays</i> applications must initialize this field. For <i>Video Output Overlays</i> the driver must return a valid format.
			Usually this is an RGB format (for example <code>V4L2_PIX_FMT_RGB565</code>) but YUV formats (only packed YUV formats when chroma keying is used, not including <code>V4L2_PIX_FMT_YUYV</code> and <code>V4L2_PIX_FMT_UYVY</code>) and the <code>V4L2_PIX_FMT_PAL8</code> format are also permitted. The behavior of the driver when an application requests a compressed format is undefined. See <i>Image Formats</i> for information on pixel formats.
	enum <code>v4l2_field</code>	field	Drivers and applications shall ignore this field. If applicable, the field order is selected with the <code>VIDIOC_S_FMT</code> ioctl, using the field field of struct <code>v4l2_window</code> .

Continued on next page

Table 199 - continued from previous page

	<code>__u32</code>	<code>bytesperline</code>	Distance in bytes between the left-most pixels in two adjacent lines.
<p>This field is irrelevant to <i>non-destructive Video Overlays</i>. For <i>destructive Video Overlays</i> both applications and drivers can set this field to request padding bytes at the end of each line. Drivers however may ignore the requested value, returning width times bytes-per-pixel or a larger value required by the hardware. That implies applications can just set this field to zero to get a reasonable default. For <i>Video Output Overlays</i> the driver must return a valid value. Video hardware may access padding bytes, therefore they must reside in accessible memory. Consider for example the case where padding bytes after the last line of an image cross a system page boundary. Capture devices may write padding bytes, the value is undefined. Output devices ignore the contents of padding bytes. When the image format is planar the <code>bytesperline</code> value applies to the first plane and is divided by the same factor as the <code>width</code> field for the other planes. For example the Cb and Cr planes of a YUV 4:2:0 image have half as many padding bytes following each line as the Y plane. To avoid ambiguities drivers must return a <code>bytesperline</code> value rounded up to a multiple of the scale factor.</p>			
	<code>__u32</code>	<code>sizeimage</code>	This field is irrelevant to <i>non-destructive Video Overlays</i> . For <i>destructive Video Overlays</i> applications must initialize this field. For <i>Video Output Overlays</i> the driver must return a valid format. Together with <code>base</code> it defines the framebuffer memory accessible by the driver.
	enum <code>v4l2_colorspace</code>	<code>colorspace</code>	This information supplements the <code>pixelformat</code> and must be set by the driver, see <i>Colorspaces</i> .
	<code>__u32</code>	<code>priv</code>	Reserved. Drivers and applications must set this field to zero.

¹ A physical base address may not suit all platforms. GK notes in theory we should pass something like PCI device + memory region + offset instead. If you encounter problems please discuss on the linux-media mailing list: <https://linuxtv.org/lists.php>.

Table 200: Frame Buffer Capability Flags

V4L2_FBUF_CAP_EXTERNOVERLAY	0x0001	The device is capable of non-destructive overlays. When the driver clears this flag, only destructive overlays are supported. There are no drivers yet which support both destructive and non-destructive overlays. Video Output Overlays are in practice always non-destructive.
V4L2_FBUF_CAP_CHROMAKEY	0x0002	The device supports clipping by chroma-keying the images. That is, image pixels replace pixels in the VGA or video signal only where the latter assume a certain color. Chroma-keying makes no sense for destructive overlays.
V4L2_FBUF_CAP_LIST_CLIPPING	0x0004	The device supports clipping using a list of clip rectangles.
V4L2_FBUF_CAP_BITMAP_CLIPPING	0x0008	The device supports clipping using a bit mask.
V4L2_FBUF_CAP_LOCAL_ALPHA	0x0010	The device supports clipping/blending using the alpha channel of the framebuffer or VGA signal. Alpha blending makes no sense for destructive overlays.
V4L2_FBUF_CAP_GLOBAL_ALPHA	0x0020	The device supports alpha blending using a global alpha value. Alpha blending makes no sense for destructive overlays.
V4L2_FBUF_CAP_LOCAL_INV_ALPHA	0x0040	The device supports clipping/blending using the inverted alpha channel of the framebuffer or VGA signal. Alpha blending makes no sense for destructive overlays.
V4L2_FBUF_CAP_SRC_CHROMAKEY	0x0080	The device supports Source Chroma-keying. Video pixels with the chroma-key colors are replaced by framebuffer pixels, which is exactly opposite of V4L2_FBUF_CAP_CHROMAKEY

Table 201: Frame Buffer Flags

V4L2_FBUF_FLAG_PRIMARY	0x0001	The framebuffer is the primary graphics surface. In other words, the overlay is destructive. This flag is typically set by any driver that doesn't have the V4L2_FBUF_CAP_EXTERNOVERLAY capability and it is cleared otherwise.
------------------------	--------	---

Continued on next page

Table 201 - continued from previous page

V4L2_FBUF_FLAG_OVERLAY	0x0002	If this flag is set for a video capture device, then the driver will set the initial overlay size to cover the full framebuffer size, otherwise the existing overlay size (as set by <i>VIDIOC_S_FMT</i>) will be used. Only one video capture driver (bttv) supports this flag. The use of this flag for capture devices is deprecated. There is no way to detect which drivers support this flag, so the only reliable method of setting the overlay size is through <i>VIDIOC_S_FMT</i> . If this flag is set for a video output device, then the video output overlay window is relative to the top-left corner of the framebuffer and restricted to the size of the framebuffer. If it is cleared, then the video output overlay window is relative to the video output display.
V4L2_FBUF_FLAG_CHROMAKEY	0x0004	Use chroma-keying. The chroma-key color is determined by the chromakey field of struct <i>v4l2_window</i> and negotiated with the <i>VIDIOC_S_FMT</i> ioctl, see <i>Video Overlay Interface</i> and <i>Video Output Overlay Interface</i> .
There are no flags to enable clipping using a list of clip rectangles or a bitmap. These methods are negotiated with the <i>VIDIOC_S_FMT</i> ioctl, see <i>Video Overlay Interface</i> and <i>Video Output Overlay Interface</i> .		
V4L2_FBUF_FLAG_LOCAL_ALPHA	0x0008	Use the alpha channel of the framebuffer to clip or blend framebuffer pixels with video images. The blend function is: $\text{output} = \text{framebuffer pixel} * \text{alpha} + \text{video pixel} * (1 - \text{alpha})$. The actual alpha depth depends on the framebuffer pixel format.
V4L2_FBUF_FLAG_GLOBAL_ALPHA	0x0010	Use a global alpha value to blend the framebuffer with video images. The blend function is: $\text{output} = (\text{framebuffer pixel} * \text{alpha} + \text{video pixel} * (255 - \text{alpha})) / 255$. The alpha value is determined by the <i>global_alpha</i> field of struct <i>v4l2_window</i> and negotiated with the <i>VIDIOC_S_FMT</i> ioctl, see <i>Video Overlay Interface</i> and <i>Video Output Overlay Interface</i> .

Continued on next page

Table 201 - continued from previous page

V4L2_FBUF_FLAG_LOCAL_INV_ALPHA	0x0020	Like V4L2_FBUF_FLAG_LOCAL_ALPHA, use the alpha channel of the framebuffer to clip or blend framebuffer pixels with video images, but with an inverted alpha value. The blend function is: $\text{output} = \text{framebuffer pixel} * (1 - \text{alpha}) + \text{video pixel} * \text{alpha}$. The actual alpha depth depends on the framebuffer pixel format.
V4L2_FBUF_FLAG_SRC_CHROMAKEY	0x0040	Use source chroma-keying. The source chroma-key color is determined by the chromakey field of struct <code>v4l2_window</code> and negotiated with the <code>VIDIOC_S_FMT</code> ioctl, see <i>Video Overlay Interface</i> and <i>Video Output Overlay Interface</i> . Both chroma-keying are mutual exclusive to each other, so same chromakey field of struct <code>v4l2_window</code> is being used.

Return Value

On success 0 is returned, on error -1 and the `errno` variable is set appropriately. The generic error codes are described at the *Generic Error Codes* chapter.

EPERM `VIDIOC_S_FBUF` can only be called by a privileged user to negotiate the parameters for a destructive overlay.

EINVAL The `VIDIOC_S_FBUF` parameters are unsuitable.

3.2.7.31 ioctl VIDIOC_G_FMT, VIDIOC_S_FMT, VIDIOC_TRY_FMT

Name

VIDIOC_G_FMT - VIDIOC_S_FMT - VIDIOC_TRY_FMT - Get or set the data format, try a format

Synopsis

VIDIOC_G_FMT

```
int ioctl(int fd, VIDIOC_G_FMT, struct v4l2_format *argp)
```

VIDIOC_S_FMT

```
int ioctl(int fd, VIDIOC_S_FMT, struct v4l2_format *argp)
```

VIDIOC_TRY_FMT

```
int ioctl(int fd, VIDIOC_TRY_FMT, struct v4l2_format *argp)
```

Arguments

fd File descriptor returned by `open()`.

argp Pointer to struct `v4l2_format`.

Description

These ioctls are used to negotiate the format of data (typically image format) exchanged between driver and application.

To query the current parameters applications set the `type` field of a struct `v4l2_format` to the respective buffer (stream) type. For example video capture devices use `V4L2_BUF_TYPE_VIDEO_CAPTURE` or `V4L2_BUF_TYPE_VIDEO_CAPTURE_MPLANE`. When the application calls the `VIDIOC_G_FMT` ioctl with a pointer to this structure the driver fills the respective member of the `fmt` union. In case of video capture devices that is either the struct `v4l2_pix_format` `pix` or the struct `v4l2_pix_format_mplane` `pix_mp` member. When the requested buffer type is not supported drivers return an `EINVAL` error code.

To change the current format parameters applications initialize the `type` field and all fields of the respective `fmt` union member. For details see the documentation of the various devices types in *Interfaces*. Good practice is to query the current parameters first, and to modify only those parameters not suitable for the application. When the application calls the `VIDIOC_S_FMT` ioctl with a pointer to a struct `v4l2_format` structure the driver checks and adjusts the parameters against hardware abilities. Drivers should not return an error code unless the `type` field is invalid, this is a mechanism to fathom device capabilities and to approach parameters acceptable for both the application and driver. On success the driver may program the hardware, allocate resources and generally prepare for data exchange. Finally the `VIDIOC_S_FMT` ioctl returns the current format parameters as `VIDIOC_G_FMT` does. Very simple, inflexible devices may even ignore all input and always return the default parameters. However all V4L2 devices exchanging data with the application must implement the `VIDIOC_G_FMT` and `VIDIOC_S_FMT` ioctl. When the requested buffer type is not supported drivers return an `EINVAL` error code on a `VIDIOC_S_FMT` attempt. When I/O is already in progress or the resource is not available for other reasons drivers return the `EBUSY` error code.

The `VIDIOC_TRY_FMT` ioctl is equivalent to `VIDIOC_S_FMT` with one exception: it does not change driver state. It can also be called at any time, never returning `EBUSY`. This function is provided to negotiate parameters, to learn about hardware limitations, without disabling I/O or possibly time consuming hardware preparations. Although strongly recommended drivers are not required to implement this ioctl.

The format as returned by `VIDIOC_TRY_FMT` must be identical to what `VIDIOC_S_FMT` returns for the same input or output.

`v4l2_format`

Table 202: struct v4l2_format

__u32	type	Type of the data stream, see v4l2_buf_type.
union {	fmt	
struct v4l2_pix_format	pix	Definition of an image format, see <i>Image Formats</i> , used by video capture and output devices.
struct v4l2_pix_format_mplane	pix_mp	Definition of an image format, see <i>Image Formats</i> , used by video capture and output devices that support the <i>multi-planar version of the API</i> .
struct v4l2_window	win	Definition of an overlaid image, see <i>Video Overlay Interface</i> , used by video overlay devices.
struct v4l2_vbi_format	vbi	Raw VBI capture or output parameters. This is discussed in more detail in <i>Raw VBI Data Interface</i> . Used by raw VBI capture and output devices.
struct v4l2_sliced_vbi_format	sliced	Sliced VBI capture or output parameters. See <i>Sliced VBI Data Interface</i> for details. Used by sliced VBI capture and output devices.
struct v4l2_sdr_format	sdr	Definition of a data format, see <i>Image Formats</i> , used by SDR capture and output devices.
struct v4l2_meta_format	meta	Definition of a metadata format, see <i>Metadata Formats</i> , used by metadata capture devices.
__u8	raw_data[200]	Place holder for future extensions.
}		

Return Value

On success 0 is returned, on error -1 and the `errno` variable is set appropriately. The generic error codes are described at the [Generic Error Codes](#) chapter.

EINVAL The struct `v4l2_format` type field is invalid or the requested buffer type not supported.

EBUSY The device is busy and cannot change the format. This could be because or the device is streaming or buffers are allocated or queued to the driver. Relevant for `VIDIOC_S_FMT` only.

3.2.7.32 ioctl VIDIOC_G_FREQUENCY, VIDIOC_S_FREQUENCY

Name

`VIDIOC_G_FREQUENCY` - `VIDIOC_S_FREQUENCY` - Get or set tuner or modulator radio frequency

Synopsis

`VIDIOC_G_FREQUENCY`

```
int ioctl(int fd, VIDIOC_G_FREQUENCY, struct v4l2_frequency *argp)
```

`VIDIOC_S_FREQUENCY`

```
int ioctl(int fd, VIDIOC_S_FREQUENCY, const struct v4l2_frequency *argp)
```

Arguments

fd File descriptor returned by `open()`.

argp Pointer to struct `v4l2_frequency`.

Description

To get the current tuner or modulator radio frequency applications set the `tuner` field of a struct `v4l2_frequency` to the respective tuner or modulator number (only input devices have tuners, only output devices have modulators), zero out the reserved array and call the `VIDIOC_G_FREQUENCY` `ioctl` with a pointer to this structure. The driver stores the current frequency in the `frequency` field.

To change the current tuner or modulator radio frequency applications initialize the `tuner`, `type` and `frequency` fields, and the reserved array of a struct `v4l2_frequency` and call the `VIDIOC_S_FREQUENCY` `ioctl` with a pointer to this structure. When the requested frequency is not possible the driver assumes the closest possible value. However `VIDIOC_S_FREQUENCY` is a write-only `ioctl`, it does not return the actual new frequency.

`v4l2_frequency`

Table 203: struct v4l2_frequency

__u32	tuner	The tuner or modulator index number. This is the same value as in the struct v4l2_input tuner field and the struct v4l2_tuner index field, or the struct v4l2_output modulator field and the struct v4l2_modulator index field.
__u32	type	The tuner type. This is the same value as in the struct v4l2_tuner type field. The type must be set to V4L2_TUNER_RADIO for /dev/radioX device nodes, and to V4L2_TUNER_ANALOG_TV for all others. Set this field to V4L2_TUNER_RADIO for modulators (currently only radio modulators are supported). See v4l2_tuner_type
__u32	frequency	Tuning frequency in units of 62.5 kHz, or if the struct v4l2_tuner or struct v4l2_modulator capability flag V4L2_TUNER_CAP_LOW is set, in units of 62.5 Hz. A 1 Hz unit is used when the capability flag V4L2_TUNER_CAP_1HZ is set.
__u32	reserved[8]	Reserved for future extensions. Drivers and applications must set the array to zero.

Return Value

On success 0 is returned, on error -1 and the `errno` variable is set appropriately. The generic error codes are described at the [Generic Error Codes](#) chapter.

EINVAL The tuner index is out of bounds or the value in the `type` field is wrong.

EBUSY A hardware seek is in progress.

3.2.7.33 ioctl VIDIOC_G_INPUT, VIDIOC_S_INPUT

Name

VIDIOC_G_INPUT - VIDIOC_S_INPUT - Query or select the current video input

Synopsis

VIDIOC_G_INPUT

```
int ioctl(int fd, VIDIOC_G_INPUT, int *argp)
```

VIDIOC_S_INPUT

```
int ioctl(int fd, VIDIOC_S_INPUT, int *argp)
```

Arguments

fd File descriptor returned by `open()`.

argp Pointer an integer with input index.

Description

To query the current video input applications call the `VIDIOC_G_INPUT` `ioctl` with a pointer to an integer where the driver stores the number of the input, as in the struct `v4l2_input` `index` field. This `ioctl` will fail only when there are no video inputs, returning `EINVAL`.

To select a video input applications store the number of the desired input in an integer and call the `VIDIOC_S_INPUT` `ioctl` with a pointer to this integer. Side effects are possible. For example inputs may support different video standards, so the driver may implicitly switch the current standard. Because of these possible side effects applications must select an input before querying or negotiating any other parameters.

Information about video inputs is available using the `ioctl VIDIOC_ENUMINPUT` `ioctl`.

Return Value

On success 0 is returned, on error -1 and the `errno` variable is set appropriately. The generic error codes are described at the [Generic Error Codes](#) chapter.

EINVAL The number of the video input is out of bounds.

3.2.7.34 `ioctl VIDIOC_G_JPEGCOMP, VIDIOC_S_JPEGCOMP`

Name

VIDIOC_G_JPEGCOMP - VIDIOC_S_JPEGCOMP

Synopsis

VIDIOC_G_JPEGCOMP

```
int ioctl(int fd, VIDIOC_G_JPEGCOMP, v4l2_jpegcompression *argp)
```

VIDIOC_S_JPEGCOMP

```
int ioctl(int fd, VIDIOC_S_JPEGCOMP, const v4l2_jpegcompression *argp)
```

Arguments

fd File descriptor returned by `open()`.

argp Pointer to struct `v4l2_jpegcompression`.

Description

These ioctls are **deprecated**. New drivers and applications should use *JPEG class controls* for image quality and JPEG markers control.

[to do]

Ronald Bultje elaborates:

APP is some application-specific information. The application can set it itself, and it'll be stored in the JPEG-encoded fields (eg; interlacing information for in an AVI or so). COM is the same, but it's comments, like 'encoded by me' or so.

jpeg_markers describes whether the huffman tables, quantization tables and the restart interval information (all JPEG-specific stuff) should be stored in the JPEG-encoded fields. These define how the JPEG field is encoded. If you omit them, applications assume you've used standard encoding. You usually do want to add them.

v4l2_jpegcompression

Table 204: struct `v4l2_jpegcompression`

int	quality	Deprecated. If <code>V4L2_CID_JPEG_COMPRESSION_QUALITY</code> control is exposed by a driver applications should use it instead and ignore this field.
int	APPn	
int	APP_len	
char	APP_data[60]	
int	COM_len	
char	COM_data[60]	
__u32	jpeg_markers	See <i>JPEG Markers Flags</i> . Deprecated. If <code>V4L2_CID_JPEG_ACTIVE_MARKER</code> control is exposed by a driver applications should use it instead and ignore this field.

Table 205: JPEG Markers Flags

V4L2_JPEG_MARKER_DHT	(1<<3)	Define Huffman Tables
V4L2_JPEG_MARKER_DQT	(1<<4)	Define Quantization Tables
V4L2_JPEG_MARKER_DRI	(1<<5)	Define Restart Interval
V4L2_JPEG_MARKER_COM	(1<<6)	Comment segment
V4L2_JPEG_MARKER_APP	(1<<7)	App segment, driver will always use APP0

Return Value

On success 0 is returned, on error -1 and the `errno` variable is set appropriately. The generic error codes are described at the [Generic Error Codes](#) chapter.

3.2.7.35 ioctl VIDIOC_G_MODULATOR, VIDIOC_S_MODULATOR

Name

VIDIOC_G_MODULATOR - VIDIOC_S_MODULATOR - Get or set modulator attributes

Synopsis

VIDIOC_G_MODULATOR

```
int ioctl(int fd, VIDIOC_G_MODULATOR, struct v4l2_modulator *argp)
```

VIDIOC_S_MODULATOR

```
int ioctl(int fd, VIDIOC_S_MODULATOR, const struct v4l2_modulator *argp)
```

Arguments

fd File descriptor returned by `open()`.

argp Pointer to struct `v4l2_modulator`.

Description

To query the attributes of a modulator applications initialize the `index` field and zero out the reserved array of a struct `v4l2_modulator` and call the [VIDIOC_G_MODULATOR](#) `ioctl` with a pointer to this structure. Drivers fill the rest of the structure or return an `EINVAL` error code when the `index` is out of bounds. To enumerate all modulators applications shall begin at `index` zero, incrementing by one until the driver returns `EINVAL`.

Modulators have two writable properties, an audio modulation set and the radio frequency. To change the modulated audio subprograms, applications initialize the `index` and `txsubchans` fields and the reserved array and call the [VIDIOC_S_MODULATOR](#) `ioctl`. Drivers may choose a different audio modulation if the request cannot be satisfied. However this is a write-only `ioctl`, it does not return the actual audio modulation selected.

SDR specific modulator types are `V4L2_TUNER_SDR` and `V4L2_TUNER_RF`. For SDR devices `txsubchans` field must be initialized to zero. The term ‘modulator’ means SDR transmitter in this context.

To change the radio frequency the `VIDIOC_S_FREQUENCY` ioctl is available.

v4l2_modulator

Table 206: struct v4l2_modulator

__u32	index	Identifies the modulator, set by the application.
__u8	name[32]	Name of the modulator, a NUL-terminated ASCII string. This information is intended for the user.
__u32	capability	Modulator capability flags. No flags are defined for this field, the tuner flags in struct v4l2_tuner are used accordingly. The audio flags indicate the ability to encode audio subprograms. They will <i>not</i> change for example with the current video standard.
__u32	range_low	The lowest tunable frequency in units of 62.5 KHz, or if the capability flag <code>V4L2_TUNER_CAP_LOW</code> is set, in units of 62.5 Hz, or if the capability flag <code>V4L2_TUNER_CAP_1HZ</code> is set, in units of 1 Hz.
__u32	range_high	The highest tunable frequency in units of 62.5 KHz, or if the capability flag <code>V4L2_TUNER_CAP_LOW</code> is set, in units of 62.5 Hz, or if the capability flag <code>V4L2_TUNER_CAP_1HZ</code> is set, in units of 1 Hz.
__u32	txsubchans	With this field applications can determine how audio sub-carriers shall be modulated. It contains a set of flags as defined in <i>Modulator Audio Transmission Flags</i> . Note: The tuner <code>rxsubchans</code> flags are reused, but the semantics are different. Video output devices are assumed to have an analog or PCM audio input with 1-3 channels. The <code>txsubchans</code> flags select one or more channels for modulation, together with some audio subprogram indicator, for example, a stereo pilot tone.
__u32	type	Type of the modulator, see <code>v4l2_tuner_type</code> .
__u32	reserved[3]	Reserved for future extensions. Drivers and applications must set the array to zero.

Table 207: Modulator Audio Transmission Flags

<code>V4L2_TUNER_SUB_MONO</code>	0x0001	Modulate channel 1 as mono audio, when the input has more channels, a down-mix of channel 1 and 2. This flag does not combine with <code>V4L2_TUNER_SUB_STEREO</code> or <code>V4L2_TUNER_SUB_LANG1</code> .
----------------------------------	--------	--

Continued on next page

Table 207 - continued from previous page

V4L2_TUNER_SUB_STEREO	0x0002	Modulate channel 1 and 2 as left and right channel of a stereo audio signal. When the input has only one channel or two channels and V4L2_TUNER_SUB_SAP is also set, channel 1 is encoded as left and right channel. This flag does not combine with V4L2_TUNER_SUB_MONO or V4L2_TUNER_SUB_LANG1. When the driver does not support stereo audio it shall fall back to mono.
V4L2_TUNER_SUB_LANG1	0x0008	Modulate channel 1 and 2 as primary and secondary language of a bilingual audio signal. When the input has only one channel it is used for both languages. It is not possible to encode the primary or secondary language only. This flag does not combine with V4L2_TUNER_SUB_MONO, V4L2_TUNER_SUB_STEREO or V4L2_TUNER_SUB_SAP. If the hardware does not support the respective audio matrix, or the current video standard does not permit bilingual audio the <i>VIDIOC_S_MODULATOR</i> ioctl shall return an EINVAL error code and the driver shall fall back to mono or stereo mode.
V4L2_TUNER_SUB_LANG2	0x0004	Same effect as V4L2_TUNER_SUB_SAP.
V4L2_TUNER_SUB_SAP	0x0004	When combined with V4L2_TUNER_SUB_MONO the first channel is encoded as mono audio, the last channel as Second Audio Program. When the input has only one channel it is used for both audio tracks. When the input has three channels the mono track is a down-mix of channel 1 and 2. When combined with V4L2_TUNER_SUB_STEREO channel 1 and 2 are encoded as left and right stereo audio, channel 3 as Second Audio Program. When the input has only two channels, the first is encoded as left and right channel and the second as SAP. When the input has only one channel it is used for all audio tracks. It is not possible to encode a Second Audio Program only. This flag must combine with V4L2_TUNER_SUB_MONO or V4L2_TUNER_SUB_STEREO. If the hardware does not support the respective audio matrix, or the current video standard does not permit SAP the <i>VIDIOC_S_MODULATOR</i> ioctl shall return an EINVAL error code and driver shall fall back to mono or stereo mode.
V4L2_TUNER_SUB_RDS	0x0010	Enable the RDS encoder for a radio FM transmitter.

Return Value

On success 0 is returned, on error -1 and the `errno` variable is set appropriately. The generic error codes are described at the [Generic Error Codes](#) chapter.

EINVAL The struct `v4l2_modulator` index is out of bounds.

3.2.7.36 `ioctl VIDIOC_G_OUTPUT, VIDIOC_S_OUTPUT`

Name

`VIDIOC_G_OUTPUT - VIDIOC_S_OUTPUT` - Query or select the current video output

Synopsis

`VIDIOC_G_OUTPUT`

```
int ioctl(int fd, VIDIOC_G_OUTPUT, int *argp)
```

`VIDIOC_S_OUTPUT`

```
int ioctl(int fd, VIDIOC_S_OUTPUT, int *argp)
```

Arguments

fd File descriptor returned by `open()`.

argp Pointer to an integer with output index.

Description

To query the current video output applications call the `VIDIOC_G_OUTPUT` `ioctl` with a pointer to an integer where the driver stores the number of the output, as in the struct `v4l2_output` `index` field. This `ioctl` will fail only when there are no video outputs, returning the `EINVAL` error code.

To select a video output applications store the number of the desired output in an integer and call the `VIDIOC_S_OUTPUT` `ioctl` with a pointer to this integer. Side effects are possible. For example outputs may support different video standards, so the driver may implicitly switch the current standard. Because of these possible side effects applications must select an output before querying or negotiating any other parameters.

Information about video outputs is available using the `ioctl VIDIOC_ENUMOUTPUT` `ioctl`.

Return Value

On success 0 is returned, on error -1 and the `errno` variable is set appropriately. The generic error codes are described at the [Generic Error Codes](#) chapter.

EINVAL The number of the video output is out of bounds, or there are no video outputs at all.

3.2.7.37 ioctl VIDIOC_G_PARM, VIDIOC_S_PARM

Name

VIDIOC_G_PARM - VIDIOC_S_PARM - Get or set streaming parameters

Synopsis

VIDIOC_G_PARM

```
int ioctl(int fd, VIDIOC_G_PARM, v4l2_streamparm *argp)
```

VIDIOC_S_PARM

```
int ioctl(int fd, VIDIOC_S_PARM, v4l2_streamparm *argp)
```

Arguments

fd File descriptor returned by `open()`.

argp Pointer to struct `v4l2_streamparm`.

Description

Applications can request a different frame interval. The capture or output device will be reconfigured to support the requested frame interval if possible. Optionally drivers may choose to skip or repeat frames to achieve the requested frame interval.

For stateful encoders (see [Memory-to-Memory Stateful Video Encoder Interface](#)) this represents the frame interval that is typically embedded in the encoded video stream.

Changing the frame interval shall never change the format. Changing the format, on the other hand, may change the frame interval.

Further these ioctls can be used to determine the number of buffers used internally by a driver in read/write mode. For implications see the section discussing the `read()` function.

To get and set the streaming parameters applications call the [VIDIOC_G_PARM](#) and [VIDIOC_S_PARM](#) ioctl, respectively. They take a pointer to a struct `v4l2_streamparm` which contains a union holding separate parameters for input and output devices.

v4l2_streamparm

Table 208: struct v4l2_streamparm

__u32	type	The buffer (stream) type, same as struct v4l2_format type, set by the application. See v4l2_buf_type.
union {	parm	
struct v4l2_captureparm	capture	Parameters for capture devices, used when type is V4L2_BUF_TYPE_VIDEO_CAPTURE or V4L2_BUF_TYPE_VIDEO_CAPTURE_MPLANE.
struct v4l2_outputparm	output	Parameters for output devices, used when type is V4L2_BUF_TYPE_VIDEO_OUTPUT or V4L2_BUF_TYPE_VIDEO_OUTPUT_MPLANE.
__u8	raw_data[200]	A place holder for future extensions.
}		

v4l2_captureparm

Table 209: struct v4l2_captureparm

__u32	capability	See <i>Streaming Parameters Capabilities</i> .
__u32	capturemode	Set by drivers and applications, see <i>Capture Parameters Flags</i> .
struct v4l2_fract	timeperframe	This is the desired period between successive frames captured by the driver, in seconds. This will configure the speed at which the video source (e.g. a sensor) generates video frames. If the speed is fixed, then the driver may choose to skip or repeat frames in order to achieve the requested frame rate. For stateful encoders (see <i>Memory-to-Memory Stateful Video Encoder Interface</i>) this represents the frame interval that is typically embedded in the encoded video stream. Applications store here the desired frame period, drivers return the actual frame period. Changing the video standard (also implicitly by switching the video input) may reset this parameter to the nominal frame period. To reset manually applications can just set this field to zero. Drivers support this function only when they set the V4L2_CAP_TIMEPERFRAME flag in the capability field.
__u32	extendedmode	Custom (driver specific) streaming parameters. When unused, applications and drivers must set this field to zero. Applications using this field should check the driver name and version, see <i>Querying Capabilities</i> .
__u32	readbuffers	Applications set this field to the desired number of buffers used internally by the driver in read() mode. Drivers return the actual number of buffers. When an application requests zero buffers, drivers should just return the current setting rather than the minimum or an error code. For details see <i>Read/Write</i> .
__u32	reserved[4]	Reserved for future extensions. Drivers and applications must set the array to zero.

v4l2_outputparm

Table 210: struct v4l2_outputparm

__u32	capability	See <i>Streaming Parameters Capabilities</i> .
__u32	outputmode	Set by drivers and applications, see <i>Capture Parameters Flags</i> .
struct v4l2_fract	timeperframe	This is the desired period between successive frames output by the driver, in seconds. The field is intended to repeat frames on the driver side in write() mode (in streaming mode timestamps can be used to throttle the output), saving I/O bandwidth. For stateful encoders (see <i>Memory-to-Memory Stateful Video Encoder Interface</i>) this represents the frame interval that is typically embedded in the encoded video stream and it provides a hint to the encoder of the speed at which raw frames are queued up to the encoder. Applications store here the desired frame period, drivers return the actual frame period. Changing the video standard (also implicitly by switching the video output) may reset this parameter to the nominal frame period. To reset manually applications can just set this field to zero. Drivers support this function only when they set the V4L2_CAP_TIMEPERFRAME flag in the capability field.
__u32	extendedmode	Custom (driver specific) streaming parameters. When unused, applications and drivers must set this field to zero. Applications using this field should check the driver name and version, see <i>Querying Capabilities</i> .
__u32	writebuffers	Applications set this field to the desired number of buffers used internally by the driver in write() mode. Drivers return the actual number of buffers. When an application requests zero buffers, drivers should just return the current setting rather than the minimum or an error code. For details see <i>Read/Write</i> .
__u32	reserved[4]	Reserved for future extensions. Drivers and applications must set the array to zero.

Table 211: Streaming Parameters Capabilities

V4L2_CAP_TIMEPERFRAME	0x1000	The frame period can be modified by setting the timeperframe field.
-----------------------	--------	---

Table 212: Capture Parameters Flags

V4L2_MODE_HIGHQUALITY	0x0001	<p>High quality imaging mode. High quality mode is intended for still imaging applications. The idea is to get the best possible image quality that the hardware can deliver. It is not defined how the driver writer may achieve that; it will depend on the hardware and the ingenuity of the driver writer. High quality mode is a different mode from the regular motion video capture modes. In high quality mode:</p> <ul style="list-style-type: none"> • The driver may be able to capture higher resolutions than for motion capture. • The driver may support fewer pixel formats than motion capture (eg; true color). • The driver may capture and arithmetically combine multiple successive fields or frames to remove color edge artifacts and reduce the noise in the video data. • The driver may capture images in slices like a scanner in order to handle larger format images than would otherwise be possible. • An image capture operation may be significantly slower than motion capture. • Moving objects in the image might have excessive motion blur. • Capture might only work through the <code>read()</code> call.
-----------------------	--------	---

Return Value

On success 0 is returned, on error -1 and the `errno` variable is set appropriately. The generic error codes are described at the [Generic Error Codes](#) chapter.

3.2.7.38 ioctl VIDIOC_G_PRIORITY, VIDIOC_S_PRIORITY

Name

VIDIOC_G_PRIORITY - VIDIOC_S_PRIORITY - Query or request the access priority associated with a file descriptor

Synopsis

VIDIOC_G_PRIORITY

```
int ioctl(int fd, VIDIOC_G_PRIORITY, enum v4l2_priority *argp)
```

VIDIOC_S_PRIORITY

```
int ioctl(int fd, VIDIOC_S_PRIORITY, const enum v4l2_priority *argp)
```

Arguments

fd File descriptor returned by `open()`.

argp Pointer to an enum `v4l2_priority` type.

Description

To query the current access priority applications call the *VIDIOC_G_PRIORITY* ioctl with a pointer to an enum `v4l2_priority` variable where the driver stores the current priority.

To request an access priority applications store the desired priority in an enum `v4l2_priority` variable and call *VIDIOC_S_PRIORITY* ioctl with a pointer to this variable.

`v4l2_priority`

Table 213: enum v4l2_priority

V4L2_PRIORITY_UNSET	0	
V4L2_PRIORITY_BACKGROUND	1	Lowest priority, usually applications running in background, for example monitoring VBI transmissions. A proxy application running in user space will be necessary if multiple applications want to read from a device at this priority.
V4L2_PRIORITY_INTERACTIVE	2	
V4L2_PRIORITY_DEFAULT	2	Medium priority, usually applications started and interactively controlled by the user. For example TV viewers, Teletext browsers, or just “panel” applications to change the channel or video controls. This is the default priority unless an application requests another.
V4L2_PRIORITY_RECORD	3	Highest priority. Only one file descriptor can have this priority, it blocks any other fd from changing device properties. Usually applications which must not be interrupted, like video recording.

Return Value

On success 0 is returned, on error -1 and the `errno` variable is set appropriately. The generic error codes are described at the [Generic Error Codes](#) chapter.

EINVAL The requested priority value is invalid.

EBUSY Another application already requested higher priority.

3.2.7.39 ioctl VIDIOC_G_SELECTION, VIDIOC_S_SELECTION

Name

VIDIOC_G_SELECTION - VIDIOC_S_SELECTION - Get or set one of the selection rectangles

Synopsis

VIDIOC_G_SELECTION

```
int ioctl(int fd, VIDIOC_G_SELECTION, struct v4l2_selection *argp)
```

VIDIOC_S_SELECTION

```
int ioctl(int fd, VIDIOC_S_SELECTION, struct v4l2_selection *argp)
```

Arguments

fd File descriptor returned by `open()`.

argp Pointer to struct `v4l2_selection`.

Description

The `ioctl`s are used to query and configure selection rectangles.

To query the cropping (composing) rectangle set struct `v4l2_selection` type field to the respective buffer type. The next step is setting the value of struct `v4l2_selection` target field to `V4L2_SEL_TGT_CROP` (`V4L2_SEL_TGT_COMPOSE`). Please refer to table *Common selection definitions* or *Cropping, composing and scaling - the SELECTION API* for additional targets. The flags and reserved fields of struct `v4l2_selection` are ignored and they must be filled with zeros. The driver fills the rest of the structure or returns `EINVAL` error code if incorrect buffer type or target was used. If cropping (composing) is not supported then the active rectangle is not mutable and it is always equal to the bounds rectangle. Finally, the struct `v4l2_rect` `r` rectangle is filled with the current cropping (composing) coordinates. The coordinates are expressed in driver-dependent units. The only exception are rectangles for images in raw formats, whose coordinates are always expressed in pixels.

To change the cropping (composing) rectangle set the struct `v4l2_selection` type field to the respective buffer type. The next step is setting the value of struct `v4l2_selection` target to `V4L2_SEL_TGT_CROP` (`V4L2_SEL_TGT_COMPOSE`). Please refer to table *Common selection definitions* or *Cropping, composing and scaling - the SELECTION API* for additional targets. The struct `v4l2_rect` `r` rectangle need to be set to the desired active area. Field struct `v4l2_selection` reserved is ignored and must be filled with zeros. The driver may adjust coordinates of the requested rectangle. An application may introduce constraints to control rounding behaviour. The struct `v4l2_selection` flags field must be set to one of the following:

- `0` - The driver can adjust the rectangle size freely and shall choose a crop/compose rectangle as close as possible to the requested one.
- `V4L2_SEL_FLAG_GE` - The driver is not allowed to shrink the rectangle. The original rectangle must lay inside the adjusted one.
- `V4L2_SEL_FLAG_LE` - The driver is not allowed to enlarge the rectangle. The adjusted rectangle must lay inside the original one.
- `V4L2_SEL_FLAG_GE | V4L2_SEL_FLAG_LE` - The driver must choose the size exactly the same as in the requested rectangle.

Please refer to *Size adjustments with constraint flags*..

The driver may have to adjust the requested dimensions against hardware limits and other parts as the pipeline, i.e. the bounds given by the capture/output window or TV display. The closest possible values of horizontal and vertical offset and sizes are chosen according to following priority:

1. Satisfy constraints from struct `v4l2_selection` flags.
2. Adjust width, height, left, and top to hardware limits and alignments.
3. Keep center of adjusted rectangle as close as possible to the original one.

4. Keep width and height as close as possible to original ones.
5. Keep horizontal and vertical offset as close as possible to original ones.

On success the struct `v4l2_rect` `r` field contains the adjusted rectangle. When the parameters are unsuitable the application may modify the cropping (composing) or image parameters and repeat the cycle until satisfactory parameters have been negotiated. If constraints flags have to be violated at then `ERANGE` is returned. The error indicates that *there exist no rectangle* that satisfies the constraints.

Selection targets and flags are documented in *Common selection definitions*.

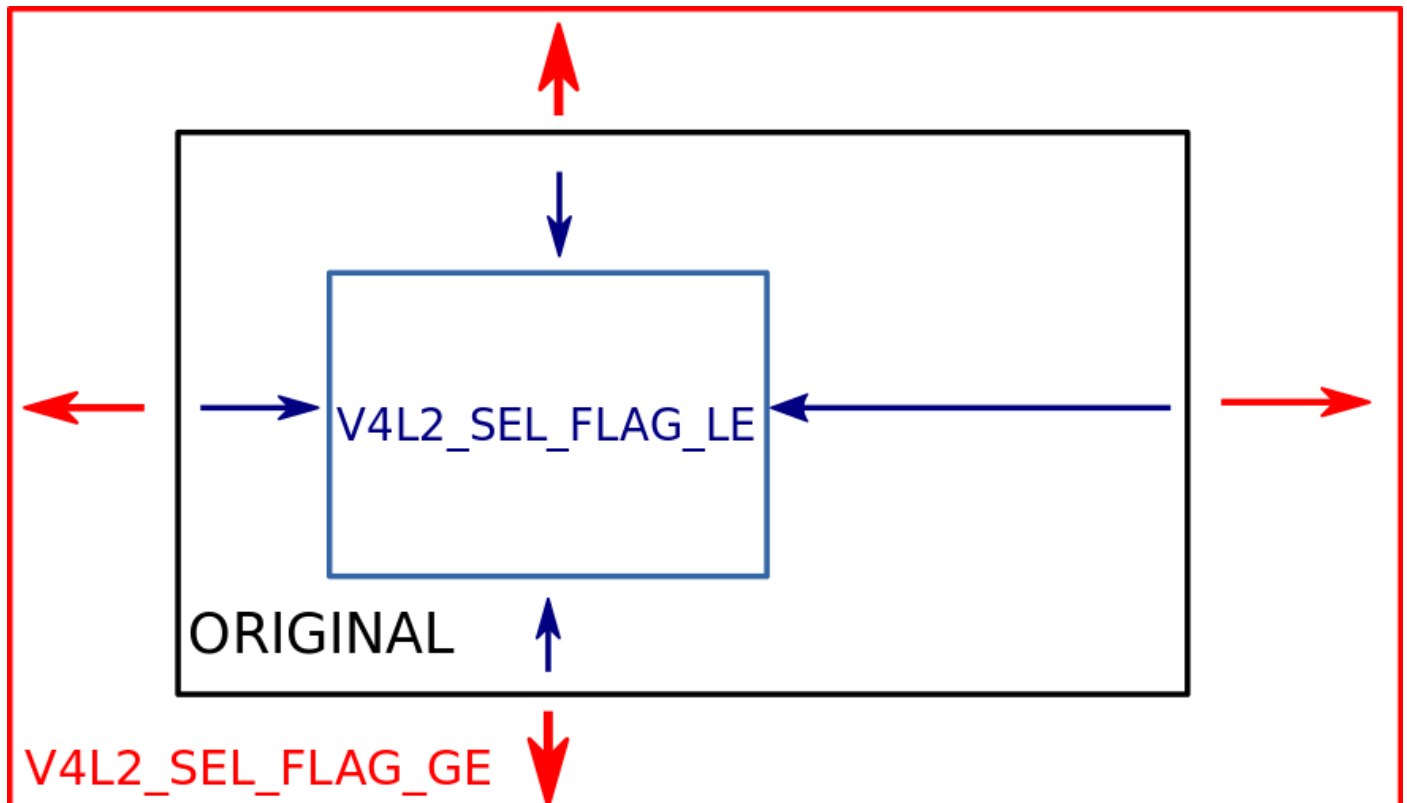


Fig. 18: Size adjustments with constraint flags.
Behaviour of rectangle adjustment for different constraint flags.

v4l2_selection

Table 214: struct `v4l2_selection`

<code>__u32</code>	<code>type</code>	Type of the buffer (from enum <code>v4l2_buf_type</code>).
<code>__u32</code>	<code>target</code>	Used to select between <i>cropping and composing rectangles</i> .
<code>__u32</code>	<code>flags</code>	Flags controlling the selection rectangle adjustments, refer to <i>selection flags</i> .
struct <code>v4l2_rect</code>	<code>r</code>	The selection rectangle.
<code>__u32</code>	<code>reserved[9]</code>	Reserved fields for future use. Drivers and applications must zero this array.

Note: Unfortunately in the case of multiplanar buffer types

(`V4L2_BUF_TYPE_VIDEO_CAPTURE_MPLANE` and `V4L2_BUF_TYPE_VIDEO_OUTPUT_MPLANE`) this API was messed up with regards to how the `v4l2_selection_type` field should be filled in. Some drivers only accepted the `_MPLANE` buffer type while other drivers only accepted a non-multiplanar buffer type (i.e. without the `_MPLANE` at the end).

Starting with kernel 4.13 both variations are allowed.

Return Value

On success 0 is returned, on error -1 and the `errno` variable is set appropriately. The generic error codes are described at the [Generic Error Codes](#) chapter.

EINVAL Given buffer type `type` or the selection target `target` is not supported, or the `flags` argument is not valid.

ERANGE It is not possible to adjust struct `v4l2_rect r` rectangle to satisfy all constraints given in the `flags` argument.

ENODATA Selection is not supported for this input or output.

EBUSY It is not possible to apply change of the selection rectangle at the moment. Usually because streaming is in progress.

3.2.7.40 ioctl VIDIOC_G_SLICED_VBI_CAP

Name

`VIDIOC_G_SLICED_VBI_CAP` - Query sliced VBI capabilities

Synopsis

`VIDIOC_G_SLICED_VBI_CAP`

```
int ioctl(int fd, VIDIOC_G_SLICED_VBI_CAP, struct v4l2_sliced_vbi_cap *argp)
```

Arguments

fd File descriptor returned by `open()`.

argp Pointer to struct `v4l2_sliced_vbi_cap`.

Description

To find out which data services are supported by a sliced VBI capture or output device, applications initialize the type field of a struct `v4l2_sliced_vbi_cap`, clear the reserved array and call the `VIDIOC_G_SLICED_VBI_CAP` ioctl. The driver fills in the remaining fields or returns an `EINVAL` error code if the sliced VBI API is unsupported or type is invalid.

Note: The type field was added, and the ioctl changed from read-only to write-read, in Linux 2.6.19.

v4l2_sliced_vbi_cap

Table 215: struct `v4l2_sliced_vbi_cap`

__u16	service_set	A set of all data services supported by the driver. Equal to the union of all elements of the <code>service_lines</code> array.		
__u16	service_lines[2][24]	Each element of this array contains a set of data services the hardware can look for or insert into a particular scan line. Data services are defined in <i>Sliced VBI services</i> . Array indices map to ITU-R line numbers ¹ as follows:		
		Element	525 line systems	625 line systems
		<code>service_lines[0][1]</code>	1	1
		<code>service_lines[0][23]</code>	23	23
		<code>service_lines[1][1]</code>	264	314
		<code>service_lines[1][23]</code>	286	336
		The number of VBI lines the hardware can capture or output per frame, or the number of services it can identify on a given line may be limited. For example on PAL line 16 the hardware may be able to look for a VPS or Teletext signal, but not both at the same time. Applications can learn about these limits using the <code>VIDIOC_S_FMT</code> ioctl as described in <i>Sliced VBI Data Interface</i> .		
		Drivers must set <code>service_lines[0][0]</code> and <code>service_lines[1][0]</code> to zero.		
__u32	type	Type of the data stream, see <code>v4l2_buf_type</code> . Should be <code>V4L2_BUF_TYPE_SLICED_VBI_CAPTURE</code> or <code>V4L2_BUF_TYPE_SLICED_VBI_OUTPUT</code> .		
__u32	reserved[3]	This array is reserved for future extensions. Applications and drivers must set it to zero.		

¹ See also *Figure 4.2. ITU-R 525 line numbering (M/NTSC and M/PAL)* and *Figure 4.3. ITU-R 625 line numbering*.

Table 216: Sliced VBI services

Symbol	Value	Reference	Lines, usually	Payload
V4L2_SLICED_TELETEXT_B (Teletext System B)	0x0001	<i>ETS 300 706</i> , <i>ITU BT.653</i>	PAL/SECAM line 7-22, 320-335 (second field 7-22)	Last 42 of the 45 byte Teletext packet, that is without clock run-in and framing code, lsb first transmitted.
V4L2_SLICED_VPS	0x0400	<i>ETS 300 231</i>	PAL line 16	Byte number 3 to 15 according to Figure 9 of ETS 300 231, lsb first transmitted.
V4L2_SLICED_CAPTION_525	0x1000	<i>CEA 608-E</i>	NTSC line 21, 284 (second field 21)	Two bytes in transmission order, including parity bit, lsb first transmitted.
V4L2_SLICED_WSS_625	0x4000	<i>EN 300 294</i> , <i>ITU BT.1119</i>	PAL/SECAM line 23	See <i>V4L2_SLICED_VBI_CAP WSS_625 payload</i> below.
V4L2_SLICED_VBI_525	0x1000	Set of services applicable to 525 line systems.		
V4L2_SLICED_VBI_625	0x4401	Set of services applicable to 625 line systems.		

V4L2_SLICED_VBI_CAP WSS_625 payload

The payload for V4L2_SLICED_WSS_625 is:

Byte	0								1							
Bit	msb				lsb				msb				lsb			
	7	6	5	4	3	2	1	0	x	x	13	12	11	10	9	8

Return Value

On success 0 is returned, on error -1 and the `errno` variable is set appropriately. The generic error codes are described at the [Generic Error Codes](#) chapter.

EINVAL The value in the type field is wrong.

3.2.7.41 `ioctl VIDIIOC_G_STD, VIDIIOC_S_STD, VIDIIOC_SUBDEV_G_STD, VIDIIOC_SUBDEV_S_STD`

Name

`VIDIIOC_G_STD` - `VIDIIOC_S_STD` - `VIDIIOC_SUBDEV_G_STD` - `VIDIIOC_SUBDEV_S_STD` - Query or select the video standard of the current input

Synopsis

`VIDIIOC_G_STD`

```
int ioctl(int fd, VIDIIOC_G_STD, v4l2_std_id *argp)
```

`VIDIIOC_S_STD`

```
int ioctl(int fd, VIDIIOC_S_STD, const v4l2_std_id *argp)
```

`VIDIIOC_SUBDEV_G_STD`

```
int ioctl(int fd, VIDIIOC_SUBDEV_G_STD, v4l2_std_id *argp)
```

`VIDIIOC_SUBDEV_S_STD`

```
int ioctl(int fd, VIDIOC_SUBDEV_S_STD, const v4l2_std_id *argp)
```

Arguments

fd File descriptor returned by `open()`.

argp Pointer to `v4l2_std_id`.

Description

To query and select the current video standard applications use the `VIDIOC_G_STD` and `VIDIOC_S_STD` ioctls which take a pointer to a `v4l2_std_id` type as argument. `VIDIOC_G_STD` can return a single flag or a set of flags as in struct `v4l2_standard` field `id`. The flags must be unambiguous such that they appear in only one enumerated struct `v4l2_standard` structure.

`VIDIOC_S_STD` accepts one or more flags, being a write-only ioctl it does not return the actual new standard as `VIDIOC_G_STD` does. When no flags are given or the current input does not support the requested standard the driver returns an `EINVAL` error code. When the standard set is ambiguous drivers may return `EINVAL` or choose any of the requested standards. If the current input or output does not support standard video timings (e.g. if `ioctl VIDIOC_ENUMINPUT` does not set the `V4L2_IN_CAP_STD` flag), then `ENODATA` error code is returned.

Calling `VIDIOC_SUBDEV_S_STD` on a subdev device node that has been registered in read-only mode is not allowed. An error is returned and the `errno` variable is set to `-EPERM`.

Return Value

On success 0 is returned, on error -1 and the `errno` variable is set appropriately. The generic error codes are described at the [Generic Error Codes](#) chapter.

EINVAL The `VIDIOC_S_STD` parameter was unsuitable.

ENODATA Standard video timings are not supported for this input or output.

EPERM `VIDIOC_SUBDEV_S_STD` has been called on a read-only subdevice.

3.2.7.42 ioctl VIDIOC_G_TUNER, VIDIOC_S_TUNER

Name

`VIDIOC_G_TUNER` - `VIDIOC_S_TUNER` - Get or set tuner attributes

Synopsis

VIDIOC_G_TUNER

```
int ioctl(int fd, VIDIOC_G_TUNER, struct v4l2_tuner *argp)
```

VIDIOC_S_TUNER

```
int ioctl(int fd, VIDIOC_S_TUNER, const struct v4l2_tuner *argp)
```

Arguments

fd File descriptor returned by `open()`.

argp Pointer to struct `v4l2_tuner`.

Description

To query the attributes of a tuner applications initialize the `index` field and zero out the reserved array of a struct `v4l2_tuner` and call the `VIDIOC_G_TUNER` `ioctl` with a pointer to this structure. Drivers fill the rest of the structure or return an `EINVAL` error code when the index is out of bounds. To enumerate all tuners applications shall begin at index zero, incrementing by one until the driver returns `EINVAL`.

Tuners have two writable properties, the audio mode and the radio frequency. To change the audio mode, applications initialize the `index`, `audmode` and `reserved` fields and call the `VIDIOC_S_TUNER` `ioctl`. This will *not* change the current tuner, which is determined by the current video input. Drivers may choose a different audio mode if the requested mode is invalid or unsupported. Since this is a write-only `ioctl`, it does not return the actually selected audio mode.

SDR specific tuner types are `V4L2_TUNER_SDR` and `V4L2_TUNER_RF`. For SDR devices `audmode` field must be initialized to zero. The term 'tuner' means SDR receiver in this context.

To change the radio frequency the `VIDIOC_S_FREQUENCY` `ioctl` is available.

v4l2_tuner

Table 217: struct `v4l2_tuner`

<code>__u32</code>	<code>index</code>	Identifies the tuner, set by the application.
<code>__u8</code>	<code>name[32]</code>	Name of the tuner, a NUL-terminated ASCII string. This information is intended for the user.
<code>__u32</code>	<code>type</code>	Type of the tuner, see <code>v4l2_tuner_type</code> .

Continued on next page

Table 217 - continued from previous page

__u32	capability	Tuner capability flags, see <i>Tuner and Modulator Capability Flags</i> . Audio flags indicate the ability to decode audio subprograms. They will <i>not</i> change, for example with the current video standard. When the structure refers to a radio tuner the V4L2_TUNER_CAP_LANG1, V4L2_TUNER_CAP_LANG2 and V4L2_TUNER_CAP_NORM flags can't be used. If multiple frequency bands are supported, then capability is the union of all capability fields of each struct v4l2_frequency_band.	
__u32	rangelow	The lowest tunable frequency in units of 62.5 kHz, or if the capability flag V4L2_TUNER_CAP_LOW is set, in units of 62.5 Hz, or if the capability flag V4L2_TUNER_CAP_1HZ is set, in units of 1 Hz. If multiple frequency bands are supported, then rangelow is the lowest frequency of all the frequency bands.	
__u32	rangehigh	The highest tunable frequency in units of 62.5 kHz, or if the capability flag V4L2_TUNER_CAP_LOW is set, in units of 62.5 Hz, or if the capability flag V4L2_TUNER_CAP_1HZ is set, in units of 1 Hz. If multiple frequency bands are supported, then rangehigh is the highest frequency of all the frequency bands.	
__u32	rxsubchans	Some tuners or audio decoders can determine the received audio subprograms by analyzing audio carriers, pilot tones or other indicators. To pass this information drivers set flags defined in <i>Tuner Audio Reception Flags</i> in this field. For example:	
		V4L2_TUNER_SUB_MONO	receiving mono audio
		STEREO SAP	receiving stereo audio and a secondary audio program
		MONO STEREO	receiving mono or stereo audio, the hardware cannot distinguish
		LANG1 LANG2	receiving bilingual audio
		MONO STEREO LANG1 LANG2	receiving mono, stereo or bilingual audio
		When the V4L2_TUNER_CAP_STEREO, _LANG1, _LANG2 or _SAP flag is cleared in the capability field, the corresponding V4L2_TUNER_SUB_ flag must not be set here. This field is valid only if this is the tuner of the current video input, or when the structure refers to a radio tuner.	
__u32	audmode	The selected audio mode, see <i>Tuner Audio Modes</i> for valid values. The audio mode does not affect audio subprogram detection, and like a <i>User Controls</i> it does not automatically change unless the requested mode is invalid or unsupported. See <i>Tuner Audio Matrix</i> for possible results when the selected and received audio programs do not match. Currently this is the only field of struct struct v4l2_tuner applications can change.	
__u32	signal	The signal strength if known. Ranging from 0 to 65535. Higher values indicate a better signal.	
__s32	afc	Automatic frequency control. When the afc value is negative, the frequency is too low, when positive too high.	

Continued on next page

Table 217 - continued from previous page

<code>__u32</code>	<code>reserved[4]</code>	Reserved for future extensions. Drivers and applications must set the array to zero.
--------------------	--------------------------	---

v4l2_tuner_type

Table 218: enum v4l2_tuner_type

<code>V4L2_TUNER_RADIO</code>	1	Tuner supports radio
<code>V4L2_TUNER_ANALOG_TV</code>	2	Tuner supports analog TV
<code>V4L2_TUNER_SDR</code>	4	Tuner controls the A/D and/or D/A block of a Software Digital Radio (SDR)
<code>V4L2_TUNER_RF</code>	5	Tuner controls the RF part of a Software Digital Radio (SDR)

Table 219: Tuner and Modulator Capability Flags

<code>V4L2_TUNER_CAP_LOW</code>	0x0001	When set, tuning frequencies are expressed in units of 62.5 Hz instead of 62.5 kHz.
<code>V4L2_TUNER_CAP_NORM</code>	0x0002	This is a multi-standard tuner; the video standard can or must be switched. (B/G PAL tuners for example are typically not considered multi-standard because the video standard is automatically determined from the frequency band.) The set of supported video standards is available from the struct <code>v4l2_input</code> pointing to this tuner, see the description of <code>ioctl VIDIOC_ENUMINPUT</code> for details. Only <code>V4L2_TUNER_ANALOG_TV</code> tuners can have this capability.
<code>V4L2_TUNER_CAP_HWSEEK_BOUNDED</code>	0x0004	If set, then this tuner supports the hardware seek functionality where the seek stops when it reaches the end of the frequency range.
<code>V4L2_TUNER_CAP_HWSEEK_WRAP</code>	0x0008	If set, then this tuner supports the hardware seek functionality where the seek wraps around when it reaches the end of the frequency range.
<code>V4L2_TUNER_CAP_STEREO</code>	0x0010	Stereo audio reception is supported.
<code>V4L2_TUNER_CAP_LANG1</code>	0x0040	Reception of the primary language of a bilingual audio program is supported. Bilingual audio is a feature of two-channel systems, transmitting the primary language monaural on the main audio carrier and a secondary language monaural on a second carrier. Only <code>V4L2_TUNER_ANALOG_TV</code> tuners can have this capability.

Continued on next page

Table 219 - continued from previous page

V4L2_TUNER_CAP_LANG2	0x0020	Reception of the secondary language of a bilingual audio program is supported. Only V4L2_TUNER_ANALOG_TV tuners can have this capability.
V4L2_TUNER_CAP_SAP	0x0020	<p>Reception of a secondary audio program is supported. This is a feature of the BTSC system which accompanies the NTSC video standard. Two audio carriers are available for mono or stereo transmissions of a primary language, and an independent third carrier for a monaural secondary language. Only V4L2_TUNER_ANALOG_TV tuners can have this capability.</p> <hr/> <p>Note: The V4L2_TUNER_CAP_LANG2 and V4L2_TUNER_CAP_SAP flags are synonyms. V4L2_TUNER_CAP_SAP applies when the tuner supports the V4L2_STD_NTSC_M video standard.</p> <hr/>
V4L2_TUNER_CAP_RDS	0x0080	RDS capture is supported. This capability is only valid for radio tuners.
V4L2_TUNER_CAP_RDS_BLOCK_IO	0x0100	The RDS data is passed as unparsed RDS blocks.
V4L2_TUNER_CAP_RDS_CONTROLS	0x0200	The RDS data is parsed by the hardware and set via controls.
V4L2_TUNER_CAP_FREQ_BANDS	0x0400	The <i>ioctl VIDIOC_ENUM_FREQ_BANDS</i> ioctl can be used to enumerate the available frequency bands.
V4L2_TUNER_CAP_HWSEEK_PROG_LIM	0x0800	The range to search when using the hardware seek functionality is programmable, see <i>ioctl VIDIOC_S_HW_FREQ_SEEK</i> for details.
V4L2_TUNER_CAP_1HZ	0x1000	When set, tuning frequencies are expressed in units of 1 Hz instead of 62.5 kHz.

Table 220: Tuner Audio Reception Flags

V4L2_TUNER_SUB_MONO	0x0001	The tuner receives a mono audio signal.
V4L2_TUNER_SUB_STEREO	0x0002	The tuner receives a stereo audio signal.
V4L2_TUNER_SUB_LANG1	0x0008	The tuner receives the primary language of a bilingual audio signal. Drivers must clear this flag when the current video standard is V4L2_STD_NTSC_M.
V4L2_TUNER_SUB_LANG2	0x0004	The tuner receives the secondary language of a bilingual audio signal (or a second audio program).
V4L2_TUNER_SUB_SAP	0x0004	The tuner receives a Second Audio Program. Note: The V4L2_TUNER_SUB_LANG2 and V4L2_TUNER_SUB_SAP flags are synonyms. The V4L2_TUNER_SUB_SAP flag applies when the current video standard is V4L2_STD_NTSC_M.
V4L2_TUNER_SUB_RDS	0x0010	The tuner receives an RDS channel.

Table 221: Tuner Audio Modes

V4L2_TUNER_MODE_MONO	0	Play mono audio. When the tuner receives a stereo signal this a down-mix of the left and right channel. When the tuner receives a bilingual or SAP signal this mode selects the primary language.
V4L2_TUNER_MODE_STEREO	1	Play stereo audio. When the tuner receives bilingual audio it may play different languages on the left and right channel or the primary language is played on both channels. Playing different languages in this mode is deprecated. New drivers should do this only in MODE_LANG1_LANG2. When the tuner receives no stereo signal or does not support stereo reception the driver shall fall back to MODE_MONO.
V4L2_TUNER_MODE_LANG1	3	Play the primary language, mono or stereo. Only V4L2_TUNER_ANALOG_TV tuners support this mode.
V4L2_TUNER_MODE_LANG2	2	Play the secondary language, mono. When the tuner receives no bilingual audio or SAP, or their reception is not supported the driver shall fall back to mono or stereo mode. Only V4L2_TUNER_ANALOG_TV tuners support this mode.
V4L2_TUNER_MODE_SAP	2	Play the Second Audio Program. When the tuner receives no bilingual audio or SAP, or their reception is not supported the driver shall fall back to mono or stereo mode. Only V4L2_TUNER_ANALOG_TV tuners support this mode. Note: The V4L2_TUNER_MODE_LANG2 and V4L2_TUNER_MODE_SAP are synonyms.
V4L2_TUNER_MODE_LANG1_LANG2	4	Play the primary language on the left channel, the secondary language on the right channel. When the tuner receives no bilingual audio or SAP, it shall fall back to MODE_LANG1 or MODE_MONO. Only V4L2_TUNER_ANALOG_TV tuners support this mode.

Table 222: Tuner Audio Matrix

	Selected V4L2_TUNER_MODE_				
Received V4L2_TUNER_SUB_	MONO	STEREO	LANG1	LANG2 = SAP	LANG1_LANG2 ¹
MONO	Mono	Mono/Mono	Mono	Mono	Mono/Mono
MONO SAP	Mono	Mono/Mono	Mono	SAP	Mono/SAP (preferred) or Mono/Mono
STEREO	L+R	L/R	Stereo L/R (preferred) or Mono L+R	Stereo L/R (preferred) or Mono L+R	L/R (preferred) or L+R/L+R
STEREO SAP	L+R	L/R	Stereo L/R (preferred) or Mono L+R	SAP	L+R/SAP (preferred) or L/R or L+R/L+R
LANG1 LANG2	Language 1	Lang1/Lang2 (deprecated ²) or Lang1/Lang1	Language 1	Language 2	Lang1/Lang2 (preferred) or Lang1/Lang1

Return Value

On success 0 is returned, on error -1 and the `errno` variable is set appropriately. The generic error codes are described at the [Generic Error Codes](#) chapter.

EINVAL The struct `v4l2_tuner` index is out of bounds.

3.2.7.43 ioctl VIDIOC_LOG_STATUS

Name

VIDIOC_LOG_STATUS - Log driver status information

Synopsis

VIDIOC_LOG_STATUS

```
int ioctl(int fd, VIDIOC_LOG_STATUS)
```

Arguments

fd File descriptor returned by `open()`.

¹ This mode has been added in Linux 2.6.17 and may not be supported by older drivers.

² Playback of both languages in `MODE_STEREO` is deprecated. In the future drivers should produce only the primary language in this mode. Applications should request `MODE_LANG1_LANG2` to record both languages or a stereo signal.

Description

As the video/audio devices become more complicated it becomes harder to debug problems. When this ioctl is called the driver will output the current device status to the kernel log. This is particular useful when dealing with problems like no sound, no video and incorrectly tuned channels. Also many modern devices autodetect video and audio standards and this ioctl will report what the device thinks what the standard is. Mismatches may give an indication where the problem is.

This ioctl is optional and not all drivers support it. It was introduced in Linux 2.6.15.

Return Value

On success 0 is returned, on error -1 and the `errno` variable is set appropriately. The generic error codes are described at the [Generic Error Codes](#) chapter.

3.2.7.44 ioctl VIDIOC_OVERLAY

Name

VIDIOC_OVERLAY - Start or stop video overlay

Synopsis

VIDIOC_OVERLAY

```
int ioctl(int fd, VIDIOC_OVERLAY, const int *argp)
```

Arguments

fd File descriptor returned by `open()`.

argp Pointer to an integer.

Description

This ioctl is part of the *video overlay* I/O method. Applications call *ioctl VIDIOC_OVERLAY* to start or stop the overlay. It takes a pointer to an integer which must be set to zero by the application to stop overlay, to one to start.

Drivers do not support *ioctl VIDIOC_STREAMON*, *VIDIOC_STREAMOFF* or *VIDIOC_STREAMOFF* with `V4L2_BUF_TYPE_VIDEO_OVERLAY`.

Return Value

On success 0 is returned, on error -1 and the `errno` variable is set appropriately. The generic error codes are described at the [Generic Error Codes](#) chapter.

EINVAL The overlay parameters have not been set up. See [Video Overlay Interface](#) for the necessary steps.

3.2.7.45 ioctl VIDIOC_PREPARE_BUF

Name

VIDIOC_PREPARE_BUF - Prepare a buffer for I/O

Synopsis

VIDIOC_PREPARE_BUF

```
int ioctl(int fd, VIDIOC_PREPARE_BUF, struct v4l2_buffer *argp)
```

Arguments

fd File descriptor returned by `open()`.

argp Pointer to struct `v4l2_buffer`.

Description

Applications can optionally call the *ioctl VIDIOC_PREPARE_BUF* ioctl to pass ownership of the buffer to the driver before actually enqueueing it, using the *VIDIOC_QBUF* ioctl, and to prepare it for future I/O. Such preparations may include cache invalidation or cleaning. Performing them in advance saves time during the actual I/O.

The struct `v4l2_buffer` structure is specified in [Buffers](#).

Return Value

On success 0 is returned, on error -1 and the `errno` variable is set appropriately. The generic error codes are described at the [Generic Error Codes](#) chapter.

EBUSY File I/O is in progress.

EINVAL The buffer type is not supported, or the index is out of bounds, or no buffers have been allocated yet, or the `userptr` or `length` are invalid.

3.2.7.46 ioctl VIDIOC_QBUF, VIDIOC_DQBUF

Name

VIDIOC_QBUF - VIDIOC_DQBUF - Exchange a buffer with the driver

Synopsis

VIDIOC_QBUF

```
int ioctl(int fd, VIDIOC_QBUF, struct v4l2_buffer *argp)
```

VIDIOC_DQBUF

```
int ioctl(int fd, VIDIOC_DQBUF, struct v4l2_buffer *argp)
```

Arguments

fd File descriptor returned by `open()`.

argp Pointer to struct `v4l2_buffer`.

Description

Applications call the `VIDIOC_QBUF` `ioctl` to enqueue an empty (capturing) or filled (output) buffer in the driver's incoming queue. The semantics depend on the selected I/O method.

To enqueue a buffer applications set the `type` field of a struct `v4l2_buffer` to the same buffer type as was previously used with struct `v4l2_format` type and struct `v4l2_requestbuffers` type. Applications must also set the `index` field. Valid index numbers range from zero to the number of buffers allocated with *ioctl VIDIOC_REQBUFS* (struct `v4l2_requestbuffers` count) minus one. The contents of the struct `v4l2_buffer` returned by a *ioctl VIDIOC_QUERYBUF* `ioctl` will do as well. When the buffer is intended for output (`type` is `V4L2_BUF_TYPE_VIDEO_OUTPUT`, `V4L2_BUF_TYPE_VIDEO_OUTPUT_MPLANE`, or `V4L2_BUF_TYPE_VBI_OUTPUT`) applications must also initialize the `bytesused`, `field` and `timestamp` fields, see *Buffers* for details. Applications must also set `flags` to 0. The `reserved2` and `reserved` fields must be set to 0. When using the *multi-planar API*, the `m.planes` field must contain a userspace pointer to a filled-in array of struct `v4l2_plane` and the `length` field must be set to the number of elements in that array.

To enqueue a *memory mapped* buffer applications set the `memory` field to `V4L2_MEMORY_MMAP`. When `VIDIOC_QBUF` is called with a pointer to this structure the driver sets the `V4L2_BUF_FLAG_MAPPED` and `V4L2_BUF_FLAG_QUEUED` flags and clears the `V4L2_BUF_FLAG_DONE` flag in the `flags` field, or it returns an `EINVAL` error code.

To enqueue a *user pointer* buffer applications set the `memory` field to `V4L2_MEMORY_USERPTR`, the `m.userptr` field to the address of the buffer and `length` to its size. When the multi-planar API is used, `m.userptr` and `length` members of the passed array of struct `v4l2_plane` have to be used instead. When `VIDIOC_QBUF` is called with a pointer to this structure the driver sets the `V4L2_BUF_FLAG_QUEUED` flag and clears the `V4L2_BUF_FLAG_MAPPED` and `V4L2_BUF_FLAG_DONE` flags in the `flags` field, or it returns an error code. This `ioctl` locks the memory pages of the buffer in physical memory, they cannot be swapped out to disk. Buffers remain locked until

dequeued, until the `VIDIOC_STREAMOFF` or `ioctl VIDIOC_REQBUFS` ioctl is called, or until the device is closed.

To enqueue a *DMABUF* buffer applications set the `memory` field to `V4L2_MEMORY_DMABUF` and the `m.fd` field to a file descriptor associated with a *DMABUF* buffer. When the multi-planar API is used the `m.fd` fields of the passed array of `struct v4l2_plane` have to be used instead. When `VIDIOC_QBUF` is called with a pointer to this structure the driver sets the `V4L2_BUF_FLAG_QUEUED` flag and clears the `V4L2_BUF_FLAG_MAPPED` and `V4L2_BUF_FLAG_DONE` flags in the `flags` field, or it returns an error code. This ioctl locks the buffer. Locking a buffer means passing it to a driver for a hardware access (usually DMA). If an application accesses (reads/writes) a locked buffer then the result is undefined. Buffers remain locked until dequeued, until the `VIDIOC_STREAMOFF` or `ioctl VIDIOC_REQBUFS` ioctl is called, or until the device is closed.

The `request_fd` field can be used with the `VIDIOC_QBUF` ioctl to specify the file descriptor of a *request*, if requests are in use. Setting it means that the buffer will not be passed to the driver until the request itself is queued. Also, the driver will apply any settings associated with the request for this buffer. This field will be ignored unless the `V4L2_BUF_FLAG_REQUEST_FD` flag is set. If the device does not support requests, then `EBADR` will be returned. If requests are supported but an invalid request file descriptor is given, then `EINVAL` will be returned.

Caution: It is not allowed to mix queuing requests with queuing buffers directly. `EBUSY` will be returned if the first buffer was queued directly and then the application tries to queue a request, or vice versa. After closing the file descriptor, calling `VIDIOC_STREAMOFF` or calling `ioctl VIDIOC_REQBUFS` the check for this will be reset.

For *memory-to-memory devices* you can specify the `request_fd` only for output buffers, not for capture buffers. Attempting to specify this for a capture buffer will result in an `EBADR` error.

Applications call the `VIDIOC_DQBUF` ioctl to dequeue a filled (capturing) or displayed (output) buffer from the driver's outgoing queue. They just set the `type`, `memory` and `reserved` fields of a `struct v4l2_buffer` as above, when `VIDIOC_DQBUF` is called with a pointer to this structure the driver fills the remaining fields or returns an error code. The driver may also set `V4L2_BUF_FLAG_ERROR` in the `flags` field. It indicates a non-critical (recoverable) streaming error. In such case the application may continue as normal, but should be aware that data in the dequeued buffer might be corrupted. When using the multi-planar API, the `planes` array must be passed in as well.

If the application sets the `memory` field to `V4L2_MEMORY_DMABUF` to dequeue a *DMABUF* buffer, the driver fills the `m.fd` field with a file descriptor numerically the same as the one given to `VIDIOC_QBUF` when the buffer was enqueued. No new file descriptor is created at dequeue time and the value is only for the application convenience. When the multi-planar API is used the `m.fd` fields of the passed array of `struct v4l2_plane` are filled instead.

By default `VIDIOC_DQBUF` blocks when no buffer is in the outgoing queue. When the `O_NONBLOCK` flag was given to the `open()` function, `VIDIOC_DQBUF` returns immediately with an `EAGAIN` error code when no buffer is available.

The `struct v4l2_buffer` structure is specified in *Buffers*.

Return Value

On success 0 is returned, on error -1 and the `errno` variable is set appropriately. The generic error codes are described at the [Generic Error Codes](#) chapter.

EAGAIN Non-blocking I/O has been selected using `O_NONBLOCK` and no buffer was in the outgoing queue.

EINVAL The buffer type is not supported, or the index is out of bounds, or no buffers have been allocated yet, or the `userptr` or `length` are invalid, or the `V4L2_BUF_FLAG_REQUEST_FD` flag was set but the given `request_fd` was invalid, or `m.fd` was an invalid DMABUF file descriptor.

EIO `VIDIOC_DQBUF` failed due to an internal error. Can also indicate temporary problems like signal loss.

Note: The driver might dequeue an (empty) buffer despite returning an error, or even stop capturing. Reusing such buffer may be unsafe though and its details (e.g. `index`) may not be returned either. It is recommended that drivers indicate recoverable errors by setting the `V4L2_BUF_FLAG_ERROR` and returning 0 instead. In that case the application should be able to safely reuse the buffer and continue streaming.

EPIPE `VIDIOC_DQBUF` returns this on an empty capture queue for mem2mem codecs if a buffer with the `V4L2_BUF_FLAG_LAST` was already dequeued and no new buffers are expected to become available.

EBADR The `V4L2_BUF_FLAG_REQUEST_FD` flag was set but the device does not support requests for the given buffer type, or the `V4L2_BUF_FLAG_REQUEST_FD` flag was not set but the device requires that the buffer is part of a request.

EBUSY The first buffer was queued via a request, but the application now tries to queue it directly, or vice versa (it is not permitted to mix the two APIs).

3.2.7.47 ioctl VIDIOC_QUERYBUF

Name

`VIDIOC_QUERYBUF` - Query the status of a buffer

Synopsis

`VIDIOC_QUERYBUF`

```
int ioctl(int fd, VIDIOC_QUERYBUF, struct v4l2_buffer *argp)
```

Arguments

fd File descriptor returned by `open()`.

argp Pointer to struct `v4l2_buffer`.

Description

This `ioctl` is part of the *streaming* I/O method. It can be used to query the status of a buffer at any time after buffers have been allocated with the `ioctl VIDIOC_REQBUFS` `ioctl`.

Applications set the `type` field of a struct `v4l2_buffer` to the same buffer type as was previously used with struct `v4l2_format` type and struct `v4l2_requestbuffers` type, and the `index` field. Valid index numbers range from zero to the number of buffers allocated with `ioctl VIDIOC_REQBUFS` (struct `v4l2_requestbuffers` count) minus one. The `reserved` and `reserved2` fields must be set to 0. When using the *multi-planar API*, the `m.planes` field must contain a userspace pointer to an array of struct `v4l2_plane` and the `length` field has to be set to the number of elements in that array. After calling `ioctl VIDIOC_QUERYBUF` with a pointer to this structure drivers return an error code or fill the rest of the structure.

In the `flags` field the `V4L2_BUF_FLAG_MAPPED`, `V4L2_BUF_FLAG_PREPARED`, `V4L2_BUF_FLAG_QUEUED` and `V4L2_BUF_FLAG_DONE` flags will be valid. The `memory` field will be set to the current I/O method. For the single-planar API, the `m.offset` contains the offset of the buffer from the start of the device memory, the `length` field its size. For the multi-planar API, fields `m.mem_offset` and `length` in the `m.planes` array elements will be used instead and the `length` field of struct `v4l2_buffer` is set to the number of filled-in array elements. The driver may or may not set the remaining fields and flags, they are meaningless in this context.

The struct `v4l2_buffer` structure is specified in *Buffers*.

Return Value

On success 0 is returned, on error -1 and the `errno` variable is set appropriately. The generic error codes are described at the *Generic Error Codes* chapter.

EINVAL The buffer type is not supported, or the `index` is out of bounds.

3.2.7.48 `ioctl VIDIOC_QUERYCAP`

Name

`VIDIOC_QUERYCAP` - Query device capabilities

Synopsis

VIDIOC_QUERYCAP

```
int ioctl(int fd, VIDIOC_QUERYCAP, struct v4l2_capability *argp)
```

Arguments

fd File descriptor returned by `open()`.

argp Pointer to struct `v4l2_capability`.

Description

All V4L2 devices support the `VIDIOC_QUERYCAP` `ioctl`. It is used to identify kernel devices compatible with this specification and to obtain information about driver and hardware capabilities. The `ioctl` takes a pointer to a struct `v4l2_capability` which is filled by the driver. When the driver is not compatible with this specification the `ioctl` returns an `EINVAL` error code.

`v4l2_capability`

Table 223: struct `v4l2_capability`

<code>__u8</code>	<code>driver[16]</code>	Name of the driver, a unique NUL-terminated ASCII string. For example: "bttv". Driver specific applications can use this information to verify the driver identity. It is also useful to work around known bugs, or to identify drivers in error reports. Storing strings in fixed sized arrays is bad practice but unavoidable here. Drivers and applications should take precautions to never read or write beyond the end of the array and to make sure the strings are properly NUL-terminated.
<code>__u8</code>	<code>card[32]</code>	Name of the device, a NUL-terminated UTF-8 string. For example: "Yoyodyne TV/FM". One driver may support different brands or models of video hardware. This information is intended for users, for example in a menu of available devices. Since multiple TV cards of the same brand may be installed which are supported by the same driver, this name should be combined with the character device file name (e. g. <code>/dev/video2</code>) or the <code>bus_info</code> string to avoid ambiguities.
<code>__u8</code>	<code>bus_info[32]</code>	Location of the device in the system, a NUL-terminated ASCII string. For example: "PCI:0000:05:06.0". This information is intended for users, to distinguish multiple identical devices. If no such information is available the field must simply count the devices controlled by the driver ("platform:vivid-000"). The <code>bus_info</code> must start with "PCI:" for PCI boards, "PCIe:" for PCI Express boards, "usb-" for USB devices, "I2C:" for i2c devices, "ISA:" for ISA devices, "parport" for parallel port devices and "platform:" for platform devices.

Continued on next page

Table 223 - continued from previous page

__u32	version	Version number of the driver. Starting with kernel 3.1, the version reported is provided by the V4L2 subsystem following the kernel numbering scheme. However, it may not always return the same version as the kernel if, for example, a stable or distribution-modified kernel uses the V4L2 stack from a newer kernel. The version number is formatted using the <code>KERNEL_VERSION()</code> macro. For example if the media stack corresponds to the V4L2 version shipped with Kernel 4.14, it would be equivalent to: <pre>#define KERNEL_VERSION(a,b,c) (((a) << 16) + ((b) << 8) + (c)) __u32 version = KERNEL_VERSION(4, 14, 0); printf ("Version: %u.%u.%u\\n", (version >> 16) & 0xFF, (version >> 8) & 0xFF, version & 0xFF);</pre>
__u32	capabilities	Available capabilities of the physical device as a whole, see <i>Device Capabilities Flags</i> . The same physical device can export multiple devices in <code>/dev</code> (e.g. <code>/dev/videoX</code> , <code>/dev/vbiY</code> and <code>/dev/radioZ</code>). The capabilities field should contain a union of all capabilities available around the several V4L2 devices exported to userspace. For all those devices the capabilities field returns the same set of capabilities. This allows applications to open just one of the devices (typically the video device) and discover whether video, vbi and/or radio are also supported.
__u32	device_caps	Device capabilities of the opened device, see <i>Device Capabilities Flags</i> . Should contain the available capabilities of that specific device node. So, for example, <code>device_caps</code> of a radio device will only contain radio related capabilities and no video or vbi capabilities. This field is only set if the capabilities field contains the <code>V4L2_CAP_DEVICE_CAPS</code> capability. Only the capabilities field can have the <code>V4L2_CAP_DEVICE_CAPS</code> capability, <code>device_caps</code> will never set <code>V4L2_CAP_DEVICE_CAPS</code> .
__u32	reserved[3]	Reserved for future extensions. Drivers must set this array to zero.

Table 224: Device Capabilities Flags

<code>V4L2_CAP_VIDEO_CAPTURE</code>	0x00000001	The device supports the single-planar API through the <i>Video Capture</i> interface.
<code>V4L2_CAP_VIDEO_CAPTURE_MPLANE</code>	0x00001000	The device supports the <i>multi-planar API</i> through the <i>Video Capture</i> interface.
<code>V4L2_CAP_VIDEO_OUTPUT</code>	0x00000002	The device supports the single-planar API through the <i>Video Output</i> interface.
<code>V4L2_CAP_VIDEO_OUTPUT_MPLANE</code>	0x00002000	The device supports the <i>multi-planar API</i> through the <i>Video Output</i> interface.
<code>V4L2_CAP_VIDEO_M2M</code>	0x00008000	The device supports the single-planar API through the Video Memory-To-Memory interface.

Continued on next page

Table 224 - continued from previous page

V4L2_CAP_VIDEO_M2M_MPLANE	0x00004000	The device supports the <i>multi-planar API</i> through the Video Memory-To-Memory interface.
V4L2_CAP_VIDEO_OVERLAY	0x00000004	The device supports the <i>Video Overlay</i> interface. A video overlay device typically stores captured images directly in the video memory of a graphics card, with hardware clipping and scaling.
V4L2_CAP_VBI_CAPTURE	0x00000010	The device supports the <i>Raw VBI Capture</i> interface, providing Teletext and Closed Caption data.
V4L2_CAP_VBI_OUTPUT	0x00000020	The device supports the <i>Raw VBI Output</i> interface.
V4L2_CAP_SLICED_VBI_CAPTURE	0x00000040	The device supports the <i>Sliced VBI Capture</i> interface.
V4L2_CAP_SLICED_VBI_OUTPUT	0x00000080	The device supports the <i>Sliced VBI Output</i> interface.
V4L2_CAP_RDS_CAPTURE	0x00000100	The device supports the <i>RDS</i> capture interface.
V4L2_CAP_VIDEO_OUTPUT_OVERLAY	0x00000200	The device supports the <i>Video Output Overlay</i> (OSD) interface. Unlike the <i>Video Overlay</i> interface, this is a secondary function of video output devices and overlays an image onto an outgoing video signal. When the driver sets this flag, it must clear the V4L2_CAP_VIDEO_OVERLAY flag and vice versa. ¹
V4L2_CAP_HW_FREQ_SEEK	0x00000400	The device supports the <i>ioctl VIDIOC_S_HW_FREQ_SEEK</i> ioctl for hardware frequency seeking.
V4L2_CAP_RDS_OUTPUT	0x00000800	The device supports the <i>RDS</i> output interface.
V4L2_CAP_TUNER	0x00010000	The device has some sort of tuner to receive RF-modulated video signals. For more information about tuner programming see <i>Tuners and Modulators</i> .
V4L2_CAP_AUDIO	0x00020000	The device has audio inputs or outputs. It may or may not support audio recording or playback, in PCM or compressed formats. PCM audio support must be implemented as ALSA or OSS interface. For more information on audio inputs and outputs see <i>Audio Inputs and Outputs</i> .
V4L2_CAP_RADIO	0x00040000	This is a radio receiver.

Continued on next page

Table 224 - continued from previous page

V4L2_CAP_MODULATOR	0x00080000	The device has some sort of modulator to emit RF-modulated video/audio signals. For more information about modulator programming see <i>Tuners and Modulators</i> .
V4L2_CAP_SDR_CAPTURE	0x00100000	The device supports the <i>SDR Capture</i> interface.
V4L2_CAP_EXT_PIX_FORMAT	0x00200000	The device supports the struct <i>v4l2_pix_format</i> extended fields.
V4L2_CAP_SDR_OUTPUT	0x00400000	The device supports the <i>SDR Output</i> interface.
V4L2_CAP_META_CAPTURE	0x00800000	The device supports the <i>Metadata Interface</i> capture interface.
V4L2_CAP_READWRITE	0x01000000	The device supports the <code>read()</code> and/or <code>write()</code> I/O methods.
V4L2_CAP_ASYNCIO	0x02000000	The device supports the <i>asynchronous</i> I/O methods.
V4L2_CAP_STREAMING	0x04000000	The device supports the <i>streaming</i> I/O method.
V4L2_CAP_META_OUTPUT	0x08000000	The device supports the <i>Metadata Interface</i> output interface.
V4L2_CAP_TOUCH	0x10000000	This is a touch device.
V4L2_CAP_IO_MC	0x20000000	There is only one input and/or output seen from userspace. The whole video topology configuration, including which I/O entity is routed to the input/output, is configured by userspace via the Media Controller. See <i>Part IV - Media Controller API</i> .
V4L2_CAP_DEVICE_CAPS	0x80000000	The driver fills the <code>device_caps</code> field. This capability can only appear in the <code>capabilities</code> field and never in the <code>device_caps</code> field.

¹ The struct `v4l2_framebuffer` lacks an `enum v4l2_buf_type` field, therefore the type of overlay is implied by the driver capabilities.

Return Value

On success 0 is returned, on error -1 and the `errno` variable is set appropriately. The generic error codes are described at the [Generic Error Codes](#) chapter.

3.2.7.49 `ioctl`s `VIDIOC_QUERYCTRL`, `VIDIOC_QUERY_EXT_CTRL` and `VIDIOC_QUERYMENU`

Name

`VIDIOC_QUERYCTRL` - `VIDIOC_QUERY_EXT_CTRL` - `VIDIOC_QUERYMENU` - Enumerate controls and menu control items

Synopsis

```
int ioctl(int fd, int VIDIOC_QUERYCTRL, struct v4l2_queryctrl *argp)
```

`VIDIOC_QUERY_EXT_CTRL`

```
int ioctl(int fd, VIDIOC_QUERY_EXT_CTRL, struct v4l2_query_ext_ctrl *argp)
```

`VIDIOC_QUERYMENU`

```
int ioctl(int fd, VIDIOC_QUERYMENU, struct v4l2_querymenu *argp)
```

Arguments

fd File descriptor returned by `open()`.

argp Pointer to struct `v4l2_queryctrl`, `v4l2_query_ext_ctrl` or `v4l2_querymenu` (depending on the `ioctl`).

Description

To query the attributes of a control applications set the `id` field of a struct `v4l2_queryctrl` and call the `VIDIOC_QUERYCTRL` `ioctl` with a pointer to this structure. The driver fills the rest of the structure or returns an `EINVAL` error code when the `id` is invalid.

It is possible to enumerate controls by calling `VIDIOC_QUERYCTRL` with successive `id` values starting from `V4L2_CID_BASE` up to and exclusive `V4L2_CID_LASTP1`. Drivers may return `EINVAL` if a control in this range is not supported. Further applications can enumerate private controls, which are not defined in this specification, by starting at `V4L2_CID_PRIVATE_BASE` and incrementing `id` until the driver returns `EINVAL`.

In both cases, when the driver sets the `V4L2_CTRL_FLAG_DISABLED` flag in the `flags` field this control is permanently disabled and should be ignored by the application.¹

¹ `V4L2_CTRL_FLAG_DISABLED` was intended for two purposes: Drivers can skip predefined controls not supported by the hardware (although returning `EINVAL` would do as well), or disable predefined and private controls after

When the application ORs `id` with `V4L2_CTRL_FLAG_NEXT_CTRL` the driver returns the next supported non-compound control, or `EINVAL` if there is none. In addition, the `V4L2_CTRL_FLAG_NEXT_COMPOUND` flag can be specified to enumerate all compound controls (i.e. controls with `type ≥ V4L2_CTRL_COMPOUND_TYPES` and/or array control, in other words controls that contain more than one value). Specify both `V4L2_CTRL_FLAG_NEXT_CTRL` and `V4L2_CTRL_FLAG_NEXT_COMPOUND` in order to enumerate all controls, compound or not. Drivers which do not support these flags yet always return `EINVAL`.

The `VIDIOC_QUERY_EXT_CTRL` ioctl was introduced in order to better support controls that can use compound types, and to expose additional control information that cannot be returned in struct `v4l2_queryctrl` since that structure is full.

`VIDIOC_QUERY_EXT_CTRL` is used in the same way as `VIDIOC_QUERYCTRL`, except that the reserved array must be zeroed as well.

Additional information is required for menu controls: the names of the menu items. To query them applications set the `id` and `index` fields of struct `v4l2_querymenu` and call the `VIDIOC_QUERYMENU` ioctl with a pointer to this structure. The driver fills the rest of the structure or returns an `EINVAL` error code when the `id` or `index` is invalid. Menu items are enumerated by calling `VIDIOC_QUERYMENU` with successive `index` values from struct `v4l2_queryctrl` minimum to maximum, inclusive.

Note: It is possible for `VIDIOC_QUERYMENU` to return an `EINVAL` error code for some indices between minimum and maximum. In that case that particular menu item is not supported by this driver. Also note that the minimum value is not necessarily 0.

See also the examples in [User Controls](#).

Table 225: struct `v4l2_queryctrl`

<code>__u32</code>	<code>id</code>	Identifies the control, set by the application. See Control IDs for predefined IDs. When the ID is ORed with <code>V4L2_CTRL_FLAG_NEXT_CTRL</code> the driver clears the flag and returns the first control with a higher ID. Drivers which do not support this flag yet always return an <code>EINVAL</code> error code.
<code>__u32</code>	<code>type</code>	Type of control, see <code>v4l2_ctrl_type</code> .
<code>__u8</code>	<code>name[32]</code>	Name of the control, a NUL-terminated ASCII string. This information is intended for the user.
<code>__s32</code>	<code>minimum</code>	Minimum value, inclusive. This field gives a lower bound for the control. See enum <code>v4l2_ctrl_type</code> how the minimum value is to be used for each possible control type. Note that this a signed 32-bit value.
<code>__s32</code>	<code>maximum</code>	Maximum value, inclusive. This field gives an upper bound for the control. See enum <code>v4l2_ctrl_type</code> how the maximum value is to be used for each possible control type. Note that this a signed 32-bit value.

Continued on next page

hardware detection without the trouble of reordering control arrays and indices (`EINVAL` cannot be used to skip private controls because it would prematurely end the enumeration).

Table 225 - continued from previous page

__s32	step	<p>This field gives a step size for the control. See enum <code>v4l2_ctrl_type</code> how the step value is to be used for each possible control type. Note that this an unsigned 32-bit value.</p> <p>Generally drivers should not scale hardware control values. It may be necessary for example when the name or id imply a particular unit and the hardware actually accepts only multiples of said unit. If so, drivers must take care values are properly rounded when scaling, such that errors will not accumulate on repeated read-write cycles.</p> <p>This field gives the smallest change of an integer control actually affecting hardware. Often the information is needed when the user can change controls by keyboard or GUI buttons, rather than a slider. When for example a hardware register accepts values 0-511 and the driver reports 0-65535, step should be 128. Note that although signed, the step value is supposed to be always positive.</p>
__s32	default_value	<p>The default value of a <code>V4L2_CTRL_TYPE_INTEGER</code>, <code>_BOOLEAN</code>, <code>_BITMASK</code>, <code>_MENU</code> or <code>_INTEGER_MENU</code> control. Not valid for other types of controls.</p> <hr/> <p>Note: Drivers reset controls to their default value only when the driver is first loaded, never afterwards.</p> <hr/>
__u32	flags	Control flags, see <i>Control Flags</i> .
__u32	reserved[2]	Reserved for future extensions. Drivers must set the array to zero.

Table 226: struct `v4l2_query_ext_ctrl`

__u32	id	Identifies the control, set by the application. See <i>Control IDs</i> for predefined IDs. When the ID is Ored with <code>V4L2_CTRL_FLAG_NEXT_CTRL</code> the driver clears the flag and returns the first non-compound control with a higher ID. When the ID is Ored with <code>V4L2_CTRL_FLAG_NEXT_COMPOUND</code> the driver clears the flag and returns the first compound control with a higher ID. Set both to get the first control (compound or not) with a higher ID.
__u32	type	Type of control, see <code>v4l2_ctrl_type</code> .
char	name[32]	Name of the control, a NUL-terminated ASCII string. This information is intended for the user.
__s64	minimum	Minimum value, inclusive. This field gives a lower bound for the control. See enum <code>v4l2_ctrl_type</code> how the minimum value is to be used for each possible control type. Note that this a signed 64-bit value.

Continued on next page

Table 226 - continued from previous page

__s64	maximum	Maximum value, inclusive. This field gives an upper bound for the control. See enum <code>v4l2_ctrl_type</code> how the maximum value is to be used for each possible control type. Note that this a signed 64-bit value.
__u64	step	<p>This field gives a step size for the control. See enum <code>v4l2_ctrl_type</code> how the step value is to be used for each possible control type. Note that this an unsigned 64-bit value.</p> <p>Generally drivers should not scale hardware control values. It may be necessary for example when the name or <code>id</code> imply a particular unit and the hardware actually accepts only multiples of said unit. If so, drivers must take care values are properly rounded when scaling, such that errors will not accumulate on repeated read-write cycles.</p> <p>This field gives the smallest change of an integer control actually affecting hardware. Often the information is needed when the user can change controls by keyboard or GUI buttons, rather than a slider. When for example a hardware register accepts values 0-511 and the driver reports 0-65535, step should be 128.</p>
__s64	default_value	<p>The default value of a <code>V4L2_CTRL_TYPE_INTEGER</code>, <code>_INTEGER64</code>, <code>_BOOLEAN</code>, <code>_BITMASK</code>, <code>_MENU</code>, <code>_INTEGER_MENU</code>, <code>_U8</code> or <code>_U16</code> control. Not valid for other types of controls.</p> <hr/> <p>Note: Drivers reset controls to their default value only when the driver is first loaded, never afterwards.</p> <hr/>
__u32	flags	Control flags, see <i>Control Flags</i> .
__u32	elem_size	The size in bytes of a single element of the array. Given a char pointer <code>p</code> to a 3-dimensional array you can find the position of cell <code>(z, y, x)</code> as follows: <code>p + ((z * dims[1] + y) * dims[0] + x) * elem_size</code> . <code>elem_size</code> is always valid, also when the control isn't an array. For string controls <code>elem_size</code> is equal to <code>maximum + 1</code> .
__u32	elems	The number of elements in the N-dimensional array. If this control is not an array, then <code>elems</code> is 1. The <code>elems</code> field can never be 0.
__u32	nr_of_dims	The number of dimension in the N-dimensional array. If this control is not an array, then this field is 0.
__u32	<code>dims[V4L2_CTRL_MAX_DIMS]</code>	The size of each dimension. The first <code>nr_of_dims</code> elements of this array must be non-zero, all remaining elements must be zero.
__u32	<code>reserved[32]</code>	Reserved for future extensions. Applications and drivers must set the array to zero.

Table 227: struct v4l2_querymenu

<code>__u32</code>	<code>id</code>	Identifies the control, set by the application from the respective struct <code>v4l2_queryctrl</code> id.
<code>__u32</code>	<code>index</code>	Index of the menu item, starting at zero, set by the application.
union	(anonymous)	
{		
<code>__u8</code>	<code>name[32]</code>	Name of the menu item, a NUL-terminated ASCII string. This information is intended for the user. This field is valid for <code>V4L2_CTRL_TYPE_MENU</code> type controls.
<code>__s64</code>	<code>value</code>	Value of the integer menu item. This field is valid for <code>V4L2_CTRL_TYPE_INTEGER_MENU</code> type controls.
}		
<code>__u32</code>	<code>reserved</code>	Reserved for future extensions. Drivers must set the array to zero.

v4l2_ctrl_type

Table 228: enum v4l2_ctrl_type

Type	minimum	step	maximum	Description
<code>V4L2_CTRL_TYPE_INTEGER</code>	any	any	any	An integer-valued control ranging from minimum to maximum inclusive. The step value indicates the increment between values.
<code>V4L2_CTRL_TYPE_BOOLEAN</code>	0	1	1	A boolean-valued control. Zero corresponds to “disabled”, and one means “enabled”.
<code>V4L2_CTRL_TYPE_MENU</code>	≥ 0	1	N-1	The control has a menu of N choices. The names of the menu items can be enumerated with the <code>VIDIOC_QUERYMENU</code> ioctl.
<code>V4L2_CTRL_TYPE_INTEGER_MENU</code>	≥ 0	1	N-1	The control has a menu of N choices. The values of the menu items can be enumerated with the <code>VIDIOC_QUERYMENU</code> ioctl. This is similar to <code>V4L2_CTRL_TYPE_MENU</code> except that instead of strings, the menu items are signed 64-bit integers.
<code>V4L2_CTRL_TYPE_BITMASK</code>	0	n/a	any	A bitmask field. The maximum value is the set of bits that can be used, all other bits are to be 0. The maximum value is interpreted as a <code>__u32</code> , allowing the use of bit 31 in the bitmask.
<code>V4L2_CTRL_TYPE_BUTTON</code>	0	0	0	A control which performs an action when set. Drivers must ignore the value passed with <code>VIDIOC_S_CTRL</code> and return an <code>EACCESS</code> error code on a <code>VIDIOC_G_CTRL</code> attempt.
<code>V4L2_CTRL_TYPE_INTEGER64</code>	any	any	any	A 64-bit integer valued control. Minimum, maximum and step size cannot be queried using <code>VIDIOC_QUERYCTRL</code> . Only <code>VIDIOC_QUERY_EXT_CTRL</code> can retrieve the 64-bit min/max/step values, they should be interpreted as n/a when using <code>VIDIOC_QUERYCTRL</code> .

Continued on next page

Table 228 - continued from previous page

Type	minimum	step	maximum	Description
V4L2_CTRL_TYPE_STRING	≥ 0	≥ 1	≥ 0	The minimum and maximum string lengths. The step size means that the string must be (minimum + N * step) characters long for $N \geq 0$. These lengths do not include the terminating zero, so in order to pass a string of length 8 to VIDIOC_S_EXT_CTRL s you need to set the size field of struct <code>v4l2_ext_control</code> to 9. For VIDIOC_G_EXT_CTRL s you can set the size field to maximum + 1. Which character encoding is used will depend on the string control itself and should be part of the control documentation.
V4L2_CTRL_TYPE_CTRL_CLASS	n/a	n/a	n/a	This is not a control. When <code>VIDIOC_QUERYCTRL</code> is called with a control ID equal to a control class code (see Control classes) + 1, the <code>ioctl</code> returns the name of the control class and this control type. Older drivers which do not support this feature return an <code>EINVAL</code> error code.
V4L2_CTRL_TYPE_U8	any	any	any	An unsigned 8-bit valued control ranging from minimum to maximum inclusive. The step value indicates the increment between values.
V4L2_CTRL_TYPE_U16	any	any	any	An unsigned 16-bit valued control ranging from minimum to maximum inclusive. The step value indicates the increment between values.
V4L2_CTRL_TYPE_U32	any	any	any	An unsigned 32-bit valued control ranging from minimum to maximum inclusive. The step value indicates the increment between values.
V4L2_CTRL_TYPE_MPEG2_QUANTISATION	n/a	n/a	n/a	A struct v4l2_ctrl_mpeg2_quantisation , containing MPEG-2 quantisation matrices for stateless video decoders.
V4L2_CTRL_TYPE_MPEG2_SEQUENCE	n/a	n/a	n/a	A struct v4l2_ctrl_mpeg2_sequence , containing MPEG-2 sequence parameters for stateless video decoders.
V4L2_CTRL_TYPE_MPEG2_PICTURE	n/a	n/a	n/a	A struct v4l2_ctrl_mpeg2_picture , containing MPEG-2 picture parameters for stateless video decoders.
V4L2_CTRL_TYPE_AREA	n/a	n/a	n/a	A struct v4l2_area , containing the width and the height of a rectangular area. Units depend on the use case.
V4L2_CTRL_TYPE_H264_SPS	n/a	n/a	n/a	A struct v4l2_ctrl_h264_sps , containing H264 sequence parameters for stateless video decoders.
V4L2_CTRL_TYPE_H264_PPS	n/a	n/a	n/a	A struct v4l2_ctrl_h264_pps , containing H264 picture parameters for stateless video decoders.
V4L2_CTRL_TYPE_H264_SCALING_MATRIX	n/a	n/a	n/a	A struct v4l2_ctrl_h264_scaling_matrix , containing H264 scaling matrices for stateless video decoders.
V4L2_CTRL_TYPE_H264_SLICE_PARAMS	n/a	n/a	n/a	A struct v4l2_ctrl_h264_slice_params , containing H264 slice parameters for stateless video decoders.
V4L2_CTRL_TYPE_H264_DECODE_PARAMS	n/a	n/a	n/a	A struct v4l2_ctrl_h264_decode_params , containing H264 decode parameters for stateless video decoders.

Continued on next page

Table 228 - continued from previous page

Type	minimum	step	maximum	Description
V4L2_CTRL_TYPE_FWHT_PARAMS	n/a	n/a	n/a	A struct <i>v4l2_ctrl_fwht_params</i> , containing FWHT parameters for stateless video decoders.
V4L2_CTRL_TYPE_HEVC_SPS	n/a	n/a	n/a	A struct <i>v4l2_ctrl_hevc_sps</i> , containing HEVC Sequence Parameter Set for stateless video decoders.
V4L2_CTRL_TYPE_HEVC_PPS	n/a	n/a	n/a	A struct <i>v4l2_ctrl_hevc_pps</i> , containing HEVC Picture Parameter Set for stateless video decoders.
V4L2_CTRL_TYPE_HEVC_SLICE_PARAMS	n/a	n/a	n/a	A struct <i>v4l2_ctrl_hevc_slice_params</i> , containing HEVC slice parameters for stateless video decoders.
V4L2_CTRL_TYPE_HEVC_SCALING_MATRIX	n/a	n/a	n/a	A struct <i>v4l2_ctrl_hevc_scaling_matrix</i> , containing HEVC scaling matrix for stateless video decoders.
V4L2_CTRL_TYPE_VP8_FRAME	n/a	n/a	n/a	A struct <i>v4l2_ctrl_vp8_frame</i> , containing VP8 frame parameters for stateless video decoders.
V4L2_CTRL_TYPE_HEVC_DECODE_PARAMS	n/a	n/a	n/a	A struct <i>v4l2_ctrl_hevc_decode_params</i> , containing HEVC decoding parameters for stateless video decoders.
V4L2_CTRL_TYPE_VP9_COMPRESSED_HDR	n/a	n/a	n/a	A struct <i>v4l2_ctrl_vp9_compressed_hdr</i> , containing VP9 probabilities updates for stateless video decoders.
V4L2_CTRL_TYPE_VP9_FRAME	n/a	n/a	n/a	A struct <i>v4l2_ctrl_vp9_frame</i> , containing VP9 frame decode parameters for stateless video decoders.

Table 229: Control Flags

V4L2_CTRL_FLAG_DISABLED	0x0001	This control is permanently disabled and should be ignored by the application. Any attempt to change the control will result in an EINVAL error code.
V4L2_CTRL_FLAG_GRABBED	0x0002	This control is temporarily unchangeable, for example because another application took over control of the respective resource. Such controls may be displayed specially in a user interface. Attempts to change the control may result in an EBUSY error code.
V4L2_CTRL_FLAG_READ_ONLY	0x0004	This control is permanently readable only. Any attempt to change the control will result in an EINVAL error code.
V4L2_CTRL_FLAG_UPDATE	0x0008	A hint that changing this control may affect the value of other controls within the same control class. Applications should update their user interface accordingly.

Continued on next page

Table 229 - continued from previous page

V4L2_CTRL_FLAG_INACTIVE	0x0010	This control is not applicable to the current configuration and should be displayed accordingly in a user interface. For example the flag may be set on a MPEG audio level 2 bitrate control when MPEG audio encoding level 1 was selected with another control.
V4L2_CTRL_FLAG_SLIDER	0x0020	A hint that this control is best represented as a slider-like element in a user interface.
V4L2_CTRL_FLAG_WRITE_ONLY	0x0040	This control is permanently writable only. Any attempt to read the control will result in an EACCES error code error code. This flag is typically present for relative controls or action controls where writing a value will cause the device to carry out a given action (e. g. motor control) but no meaningful value can be returned.
V4L2_CTRL_FLAG_VOLATILE	0x0080	<p>This control is volatile, which means that the value of the control changes continuously. A typical example would be the current gain value if the device is in auto-gain mode. In such a case the hardware calculates the gain value based on the lighting conditions which can change over time.</p> <hr/> <p>Note: Setting a new value for a volatile control will be ignored unless V4L2_CTRL_FLAG_EXECUTE_ON_WRITE is also set. Setting a new value for a volatile control will <i>never</i> trigger a V4L2_EVENT_CTRL_CH_VALUE event.</p> <hr/>
V4L2_CTRL_FLAG_HAS_PAYLOAD	0x0100	This control has a pointer type, so its value has to be accessed using one of the pointer fields of struct <code>v4l2_ext_control</code> . This flag is set for controls that are an array, string, or have a compound type. In all cases you have to set a pointer to memory containing the payload of the control.
V4L2_CTRL_FLAG_EXECUTE_ON_WRITE	0x0200	The value provided to the control will be propagated to the driver even if it remains constant. This is required when the control represents an action on the hardware. For example: clearing an error flag or triggering the flash. All the controls of the type <code>V4L2_CTRL_TYPE_BUTTON</code> have this flag set.

Continued on next page

Table 229 - continued from previous page

V4L2_CTRL_FLAG_MODIFY_LAYOUT	0x0400	Changing this control value may modify the layout of the buffer (for video devices) or the media bus format (for sub-devices). A typical example would be the V4L2_CID_ROTATE control. Note that typically controls with this flag will also set the V4L2_CTRL_FLAG_GRABBED flag when buffers are allocated or streaming is in progress since most drivers do not support changing the format in that case.
------------------------------	--------	--

Return Value

On success 0 is returned, on error -1 and the `errno` variable is set appropriately. The generic error codes are described at the [Generic Error Codes](#) chapter.

EINVAL The struct `v4l2_queryctrl` id is invalid. The struct `v4l2_querymenu` id is invalid or index is out of range (less than minimum or greater than maximum) or this particular menu item is not supported by the driver.

EACCES An attempt was made to read a write-only control.

3.2.7.50 ioctl VIDIOC_QUERY_DV_TIMINGS

Name

VIDIOC_QUERY_DV_TIMINGS - VIDIOC_SUBDEV_QUERY_DV_TIMINGS - Sense the DV preset received by the current input

Synopsis

VIDIOC_QUERY_DV_TIMINGS

```
int ioctl(int fd, VIDIOC_QUERY_DV_TIMINGS, struct v4l2_dv_timings *argp)
```

VIDIOC_SUBDEV_QUERY_DV_TIMINGS

```
int ioctl(int fd, VIDIOC_SUBDEV_QUERY_DV_TIMINGS, struct v4l2_dv_timings *argp)
```

Arguments

fd File descriptor returned by `open()`.

argp Pointer to struct `v4l2_dv_timings`.

Description

The hardware may be able to detect the current DV timings automatically, similar to sensing the video standard. To do so, applications call *ioctl VIDIOC_QUERY_DV_TIMINGS* with a pointer to a struct `v4l2_dv_timings`. Once the hardware detects the timings, it will fill in the timings structure.

Note: Drivers shall *not* switch timings automatically if new timings are detected. Instead, drivers should send the `V4L2_EVENT_SOURCE_CHANGE` event (if they support this) and expect that userspace will take action by calling *ioctl VIDIOC_QUERY_DV_TIMINGS*. The reason is that new timings usually mean different buffer sizes as well, and you cannot change buffer sizes on the fly. In general, applications that receive the Source Change event will have to call *ioctl VIDIOC_QUERY_DV_TIMINGS*, and if the detected timings are valid they will have to stop streaming, set the new timings, allocate new buffers and start streaming again.

If the timings could not be detected because there was no signal, then `ENOLINK` is returned. If a signal was detected, but it was unstable and the receiver could not lock to the signal, then `ENOLCK` is returned. If the receiver could lock to the signal, but the format is unsupported (e.g. because the pixelclock is out of range of the hardware capabilities), then the driver fills in whatever timings it could find and returns `ERANGE`. In that case the application can call *ioctl VIDIOC_DV_TIMINGS_CAP, VIDIOC_SUBDEV_DV_TIMINGS_CAP* to compare the found timings with the hardware's capabilities in order to give more precise feedback to the user.

Return Value

On success 0 is returned, on error -1 and the `errno` variable is set appropriately. The generic error codes are described at the *Generic Error Codes* chapter.

ENODATA Digital video timings are not supported for this input or output.

ENOLINK No timings could be detected because no signal was found.

ENOLCK The signal was unstable and the hardware could not lock on to it.

ERANGE Timings were found, but they are out of range of the hardware capabilities.

3.2.7.51 *ioctl* VIDIOC_QUERYSTD, VIDIOC_SUBDEV_QUERYSTD

Name

`VIDIOC_QUERYSTD - VIDIOC_SUBDEV_QUERYSTD` - Sense the video standard received by the current input

Synopsis

VIDIOC_QUERYSTD

```
int ioctl(int fd, VIDIOC_QUERYSTD, v4l2_std_id *argp)
```

VIDIOC_SUBDEV_QUERYSTD

```
int ioctl(int fd, VIDIOC_SUBDEV_QUERYSTD, v4l2_std_id *argp)
```

Arguments

fd File descriptor returned by `open()`.

argp Pointer to `v4l2_std_id`.

Description

The hardware may be able to detect the current video standard automatically. To do so, applications call *ioctl VIDIOC_QUERYSTD, VIDIOC_SUBDEV_QUERYSTD* with a pointer to a *v4l2_std_id* type. The driver stores here a set of candidates, this can be a single flag or a set of supported standards if for example the hardware can only distinguish between 50 and 60 Hz systems. If no signal was detected, then the driver will return `V4L2_STD_UNKNOWN`. When detection is not possible or fails, the set must contain all standards supported by the current video input or output.

Note: Drivers shall *not* switch the video standard automatically if a new video standard is detected. Instead, drivers should send the `V4L2_EVENT_SOURCE_CHANGE` event (if they support this) and expect that userspace will take action by calling *ioctl VIDIOC_QUERYSTD, VIDIOC_SUBDEV_QUERYSTD*. The reason is that a new video standard can mean different buffer sizes as well, and you cannot change buffer sizes on the fly. In general, applications that receive the Source Change event will have to call *ioctl VIDIOC_QUERYSTD, VIDIOC_SUBDEV_QUERYSTD*, and if the detected video standard is valid they will have to stop streaming, set the new standard, allocate new buffers and start streaming again.

Return Value

On success 0 is returned, on error -1 and the `errno` variable is set appropriately. The generic error codes are described at the *Generic Error Codes* chapter.

ENODATA Standard video timings are not supported for this input or output.

3.2.7.52 ioctl VIDIOC_REQBUFS

Name

VIDIOC_REQBUFS - Initiate Memory Mapping, User Pointer I/O or DMA buffer I/O

Synopsis

VIDIOC_REQBUFS

```
int ioctl(int fd, VIDIOC_REQBUFS, struct v4l2_requestbuffers *argp)
```

Arguments

fd File descriptor returned by `open()`.

argp Pointer to struct `v4l2_requestbuffers`.

Description

This `ioctl` is used to initiate *memory mapped*, *user pointer* or *DMABUF* based I/O. Memory mapped buffers are located in device memory and must be allocated with this `ioctl` before they can be mapped into the application's address space. User buffers are allocated by applications themselves, and this `ioctl` is merely used to switch the driver into user pointer I/O mode and to setup some internal structures. Similarly, *DMABUF* buffers are allocated by applications through a device driver, and this `ioctl` only configures the driver into *DMABUF* I/O mode without performing any direct allocation.

To allocate device buffers applications initialize all fields of the struct `v4l2_requestbuffers` structure. They set the `type` field to the respective stream or buffer type, the `count` field to the desired number of buffers, `memory` must be set to the requested I/O method and the `reserved` array must be zeroed. When the `ioctl` is called with a pointer to this structure the driver will attempt to allocate the requested number of buffers and it stores the actual number allocated in the `count` field. It can be smaller than the number requested, even zero, when the driver runs out of free memory. A larger number is also possible when the driver requires more buffers to function correctly. For example video output requires at least two buffers, one displayed and one filled by the application.

When the I/O method is not supported the `ioctl` returns an `EINVAL` error code.

Applications can call *ioctl VIDIOC_REQBUFS* again to change the number of buffers. Note that if any buffers are still mapped or exported via *DMABUF*, then *ioctl VIDIOC_REQBUFS* can only succeed if the `V4L2_BUF_CAP_SUPPORTS_ORPHANED_BUFS` capability is set. Otherwise *ioctl VIDIOC_REQBUFS* will return the `EBUSY` error code. If `V4L2_BUF_CAP_SUPPORTS_ORPHANED_BUFS` is set, then these buffers are orphaned and will be freed when they are unmapped or when the exported *DMABUF* fds are closed. A count value of zero frees or orphans all buffers, after aborting or finishing any DMA in progress, an implicit *VIDIOC_STREAMOFF*.

`v4l2_requestbuffers`

Table 230: struct v4l2_requestbuffers

__u32	count	The number of buffers requested or granted.
__u32	type	Type of the stream or buffers, this is the same as the struct v4l2_format type field. See v4l2_buf_type for valid values.
__u32	memory	Applications set this field to V4L2_MEMORY_MMAP, V4L2_MEMORY_DMABUF or V4L2_MEMORY_USERPTR. See v4l2_memory.
__u32	capabilities	Set by the driver. If 0, then the driver doesn't support capabilities. In that case all you know is that the driver is guaranteed to support V4L2_MEMORY_MMAP and <i>might</i> support other v4l2_memory types. It will not support any other capabilities. If you want to query the capabilities with a minimum of side-effects, then this can be called with count set to 0, memory set to V4L2_MEMORY_MMAP and type set to the buffer type. This will free any previously allocated buffers, so this is typically something that will be done at the start of the application.
__u8	flags	Specifies additional buffer management attributes. See <i>Memory Consistency Flags</i> .
__u8	reserved[3]	Reserved for future extensions.

Table 231: V4L2 Buffer Capabilities Flags

V4L2_BUF_CAP_SUPPORTS_MMAP	0x00000001	This buffer type supports the V4L2_MEMORY_MMAP streaming mode.
V4L2_BUF_CAP_SUPPORTS_USERPTR	0x00000002	This buffer type supports the V4L2_MEMORY_USERPTR streaming mode.
V4L2_BUF_CAP_SUPPORTS_DMABUF	0x00000004	This buffer type supports the V4L2_MEMORY_DMABUF streaming mode.
V4L2_BUF_CAP_SUPPORTS_REQUESTS	0x00000008	This buffer type supports <i>requests</i> .
V4L2_BUF_CAP_SUPPORTS_ORPHANED_BUFS	0x00000010	The kernel allows calling <i>ioctl VIDIOC_REQBUFS</i> while buffers are still mapped or exported via DMABUF. These orphaned buffers will be freed when they are unmapped or when the exported DMABUF fds are closed.
V4L2_BUF_CAP_SUPPORTS_M2M_HOLD_CAPTURE_BUF	0x00000020	Only valid for stateless decoders. If set, then userspace can set the V4L2_BUF_FLAG_M2M_HOLD_CAPTURE_BUF flag to hold off on returning the capture buffer until the OUTPUT timestamp changes.

Continued on next page

Table 231 - continued from previous page

V4L2_BUF_CAP_SUPPORTS_MMAP_CACHE_HINTS	0x00000040	This capability is set by the driver to indicate that the queue supports cache and memory management hints. However, it's only valid when the queue is used for <i>memory mapping</i> streaming I/O. See V4L2_BUF_FLAG_NO_CACHE_INVALIDATE , V4L2_BUF_FLAG_NO_CACHE_CLEAN and V4L2_MEMORY_FLAG_NON_COHERENT .
--	------------	---

Return Value

On success 0 is returned, on error -1 and the `errno` variable is set appropriately. The generic error codes are described at the [Generic Error Codes](#) chapter.

EINVAL The buffer type (type field) or the requested I/O method (memory) is not supported.

3.2.7.53 ioctl VIDIOC_S_HW_FREQ_SEEK

Name

VIDIOC_S_HW_FREQ_SEEK - Perform a hardware frequency seek

Synopsis

VIDIOC_S_HW_FREQ_SEEK

```
int ioctl(int fd, VIDIOC_S_HW_FREQ_SEEK, struct v4l2_hw_freq_seek *argp)
```

Arguments

fd File descriptor returned by `open()`.

argp Pointer to struct `v4l2_hw_freq_seek`.

Description

Start a hardware frequency seek from the current frequency. To do this applications initialize the tuner, type, seek_upward, wrap_around, spacing, rangelow and rangehigh fields, and zero out the reserved array of a struct `v4l2_hw_freq_seek` and call the `VIDIOC_S_HW_FREQ_SEEK` `ioctl` with a pointer to this structure.

The `rangelow` and `rangehigh` fields can be set to a non-zero value to tell the driver to search a specific band. If the struct `v4l2_tuner` capability field has the `V4L2_TUNER_CAP_HWSEEK_PROG_LIM` flag set, these values must fall within one of the bands returned by `ioctl VIDIOC_ENUM_FREQ_BANDS`. If the `V4L2_TUNER_CAP_HWSEEK_PROG_LIM` flag is not set, then these values must exactly match those of one of the bands returned by `ioctl VIDIOC_ENUM_FREQ_BANDS`. If the current frequency of the tuner does not fall within the selected band it will be clamped to fit in the band before the seek is started.

If an error is returned, then the original frequency will be restored.

This ioctl is supported if the V4L2_CAP_HW_FREQ_SEEK capability is set.

If this ioctl is called from a non-blocking filehandle, then EAGAIN error code is returned and no seek takes place.

v4l2_hw_freq_seek

Table 232: struct v4l2_hw_freq_seek

__u32	tuner	The tuner index number. This is the same value as in the struct v4l2_input tuner field and the struct v4l2_tuner index field.
__u32	type	The tuner type. This is the same value as in the struct v4l2_tuner type field. See v4l2_tuner_type
__u32	seek_upward	If non-zero, seek upward from the current frequency, else seek downward.
__u32	wrap_around	If non-zero, wrap around when at the end of the frequency range, else stop seeking. The struct v4l2_tuner capability field will tell you what the hardware supports.
__u32	spacing	If non-zero, defines the hardware seek resolution in Hz. The driver selects the nearest value that is supported by the device. If spacing is zero a reasonable default value is used.
__u32	rangelow	If non-zero, the lowest tunable frequency of the band to search in units of 62.5 kHz, or if the struct v4l2_tuner capability field has the V4L2_TUNER_CAP_LOW flag set, in units of 62.5 Hz or if the struct v4l2_tuner capability field has the V4L2_TUNER_CAP_1HZ flag set, in units of 1 Hz. If rangelow is zero a reasonable default value is used.
__u32	rangehigh	If non-zero, the highest tunable frequency of the band to search in units of 62.5 kHz, or if the struct v4l2_tuner capability field has the V4L2_TUNER_CAP_LOW flag set, in units of 62.5 Hz or if the struct v4l2_tuner capability field has the V4L2_TUNER_CAP_1HZ flag set, in units of 1 Hz. If rangehigh is zero a reasonable default value is used.
__u32	reserved[5]	Reserved for future extensions. Applications must set the array to zero.

Return Value

On success 0 is returned, on error -1 and the `errno` variable is set appropriately. The generic error codes are described at the [Generic Error Codes](#) chapter.

EINVAL The tuner index is out of bounds, the `wrap_around` value is not supported or one of the values in the `type`, `range_low` or `range_high` fields is wrong.

EAGAIN Attempted to call `VIDIOC_S_HW_FREQ_SEEK` with the filehandle in non-blocking mode.

ENODATA The hardware seek found no channels.

EBUSY Another hardware seek is already in progress.

3.2.7.54 ioctl VIDIOC_STREAMON, VIDIOC_STREAMOFF

Name

VIDIOC_STREAMON - VIDIOC_STREAMOFF - Start or stop streaming I/O

Synopsis

VIDIOC_STREAMON

```
int ioctl(int fd, VIDIOC_STREAMON, const int *argp)
```

VIDIOC_STREAMOFF

```
int ioctl(int fd, VIDIOC_STREAMOFF, const int *argp)
```

Arguments

fd File descriptor returned by `open()`.

argp Pointer to an integer.

Description

The `VIDIOC_STREAMON` and `VIDIOC_STREAMOFF` `ioctl` start and stop the capture or output process during streaming (*memory mapping*, *user pointer* or *DMABUF*) I/O.

Capture hardware is disabled and no input buffers are filled (if there are any empty buffers in the incoming queue) until `VIDIOC_STREAMON` has been called. Output hardware is disabled and no video signal is produced until `VIDIOC_STREAMON` has been called. The `ioctl` will succeed when at least one output buffer is in the incoming queue.

Memory-to-memory devices will not start until `VIDIOC_STREAMON` has been called for both the capture and output stream types.

If `VIDIOC_STREAMON` fails then any already queued buffers will remain queued.

The `VIDIOC_STREAMOFF` `ioctl`, apart of aborting or finishing any DMA in progress, unlocks any user pointer buffers locked in physical memory, and it removes all buffers from the incoming and

outgoing queues. That means all images captured but not dequeued yet will be lost, likewise all images enqueued for output but not transmitted yet. I/O returns to the same state as after calling *ioctl VIDIOC_REQBUFS* and can be restarted accordingly.

If buffers have been queued with *ioctl VIDIOC_QBUF, VIDIOC_DQBUF* and `VIDIOC_STREAMOFF` is called without ever having called `VIDIOC_STREAMON`, then those queued buffers will also be removed from the incoming queue and all are returned to the same state as after calling *ioctl VIDIOC_REQBUFS* and can be restarted accordingly.

Both *ioctls* take a pointer to an integer, the desired buffer or stream type. This is the same as struct `v4l2_requestbuffers` type.

If `VIDIOC_STREAMON` is called when streaming is already in progress, or if `VIDIOC_STREAMOFF` is called when streaming is already stopped, then 0 is returned. Nothing happens in the case of `VIDIOC_STREAMON`, but `VIDIOC_STREAMOFF` will return queued buffers to their starting state as mentioned above.

Note: Applications can be preempted for unknown periods right before or after the `VIDIOC_STREAMON` or `VIDIOC_STREAMOFF` calls, there is no notion of starting or stopping “now”. Buffer timestamps can be used to synchronize with other events.

Return Value

On success 0 is returned, on error -1 and the `errno` variable is set appropriately. The generic error codes are described at the *Generic Error Codes* chapter.

EINVAL The buffer type is not supported, or no buffers have been allocated (memory mapping) or enqueued (output) yet.

EPIPE The driver implements *pad-level format configuration* and the pipeline configuration is invalid.

ENOLINK The driver implements Media Controller interface and the pipeline link configuration is invalid.

3.2.7.55 ioctl VIDIOC_SUBDEV_ENUM_FRAME_INTERVAL

Name

VIDIOC_SUBDEV_ENUM_FRAME_INTERVAL - Enumerate frame intervals

Synopsis

VIDIOC_SUBDEV_ENUM_FRAME_INTERVAL

```
int ioctl(int fd, VIDIOC_SUBDEV_ENUM_FRAME_INTERVAL, struct
v4l2_subdev_frame_interval_enum * argp)
```

Arguments

fd File descriptor returned by `open()`.

argp Pointer to struct `v4l2_subdev_frame_interval_enum`.

Description

This `ioctl` lets applications enumerate available frame intervals on a given sub-device pad. Frame intervals only makes sense for sub-devices that can control the frame period on their own. This includes, for instance, image sensors and TV tuners.

For the common use case of image sensors, the frame intervals available on the sub-device output pad depend on the frame format and size on the same pad. Applications must thus specify the desired format and size when enumerating frame intervals.

To enumerate frame intervals applications initialize the `index`, `pad`, `which`, `code`, `width` and `height` fields of struct `v4l2_subdev_frame_interval_enum` and call the `ioctl VIDIOC_SUBDEV_ENUM_FRAME_INTERVAL` `ioctl` with a pointer to this structure. Drivers fill the rest of the structure or return an `EINVAL` error code if one of the input fields is invalid. All frame intervals are enumerable by beginning at index zero and incrementing by one until `EINVAL` is returned.

Available frame intervals may depend on the current 'try' formats at other pads of the sub-device, as well as on the current active links. See `ioctl VIDIOC_SUBDEV_G_FMT`, `VIDIOC_SUBDEV_S_FMT` for more information about the try formats.

Sub-devices that support the frame interval enumeration `ioctl` should implemented it on a single pad only. Its behaviour when supported on multiple pads of the same sub-device is not defined.

`v4l2_subdev_frame_interval_enum`

Table 233: struct v4l2_subdev_frame_interval_enum

__u32	index	Number of the format in the enumeration, set by the application.
__u32	pad	Pad number as reported by the media controller API.
__u32	code	The media bus format code, as defined in <i>Media Bus Formats</i> .
__u32	width	Frame width, in pixels.
__u32	height	Frame height, in pixels.
struct v4l2_fract	interval	Period, in seconds, between consecutive video frames.
__u32	which	Frame intervals to be enumerated, from enum <i>v4l2_subdev_format_whence</i> .
__u32	reserved[8]	Reserved for future extensions. Applications and drivers must set the array to zero.

Return Value

On success 0 is returned, on error -1 and the `errno` variable is set appropriately. The generic error codes are described at the *Generic Error Codes* chapter.

EINVAL The struct `v4l2_subdev_frame_interval_enum` pad references a non-existing pad, one of the code, width or height fields are invalid for the given pad or the index field is out of bounds.

3.2.7.56 ioctl VIDIOC_SUBDEV_ENUM_FRAME_SIZE

Name

VIDIOC_SUBDEV_ENUM_FRAME_SIZE - Enumerate media bus frame sizes

Synopsis

VIDIOC_SUBDEV_ENUM_FRAME_SIZE

```
int ioctl(int fd, VIDIOC_SUBDEV_ENUM_FRAME_SIZE, struct v4l2_subdev_frame_size_enum * argp)
```

Arguments

fd File descriptor returned by `open()`.

argp Pointer to struct `v4l2_subdev_frame_size_enum`.

Description

This ioctl allows applications to enumerate all frame sizes supported by a sub-device on the given pad for the given media bus format. Supported formats can be retrieved with the *ioctl VIDIOC_SUBDEV_ENUM_MBUS_CODE* ioctl.

To enumerate frame sizes applications initialize the `pad`, `code` and `index` fields of the struct `v4l2_subdev_mbus_code_enum` and call the *ioctl VIDIOC_SUBDEV_ENUM_FRAME_SIZE* ioctl with a pointer to the structure. Drivers fill the minimum and maximum frame sizes or return an `EINVAL` error code if one of the input parameters is invalid.

Sub-devices that only support discrete frame sizes (such as most sensors) will return one or more frame sizes with identical minimum and maximum values.

Not all possible sizes in given [minimum, maximum] ranges need to be supported. For instance, a scaler that uses a fixed-point scaling ratio might not be able to produce every frame size between the minimum and maximum values. Applications must use the *VIDIOC_SUBDEV_S_FMT* ioctl to try the sub-device for an exact supported frame size.

Available frame sizes may depend on the current 'try' formats at other pads of the sub-device, as well as on the current active links and the current values of V4L2 controls. See *ioctl VIDIOC_SUBDEV_G_FMT*, *VIDIOC_SUBDEV_S_FMT* for more information about try formats.

v4l2_subdev_frame_size_enum

Table 234: struct `v4l2_subdev_frame_size_enum`

<code>__u32</code>	<code>index</code>	Number of the format in the enumeration, set by the application.
<code>__u32</code>	<code>pad</code>	Pad number as reported by the media controller API.
<code>__u32</code>	<code>code</code>	The media bus format code, as defined in <i>Media Bus Formats</i> .
<code>__u32</code>	<code>min_width</code>	Minimum frame width, in pixels.
<code>__u32</code>	<code>max_width</code>	Maximum frame width, in pixels.
<code>__u32</code>	<code>min_height</code>	Minimum frame height, in pixels.
<code>__u32</code>	<code>max_height</code>	Maximum frame height, in pixels.
<code>__u32</code>	<code>which</code>	Frame sizes to be enumerated, from enum <i>v4l2_subdev_format_whence</i> .
<code>__u32</code>	<code>reserved[8]</code>	Reserved for future extensions. Applications and drivers must set the array to zero.

Return Value

On success 0 is returned, on error -1 and the `errno` variable is set appropriately. The generic error codes are described at the *Generic Error Codes* chapter.

EINVAL The struct `v4l2_subdev_frame_size_enum` `pad` references a non-existing pad, the `code` is invalid for the given `pad` or the `index` field is out of bounds.

3.2.7.57 ioctl VIDIOC_SUBDEV_ENUM_MBUS_CODE

Name

VIDIOC_SUBDEV_ENUM_MBUS_CODE - Enumerate media bus formats

Synopsis

VIDIOC_SUBDEV_ENUM_MBUS_CODE

```
int ioctl(int fd, VIDIOC_SUBDEV_ENUM_MBUS_CODE, struct v4l2_subdev_mbus_code_enum
* argp)
```

Arguments

fd File descriptor returned by `open()`.

argp Pointer to struct `v4l2_subdev_mbus_code_enum`.

Description

To enumerate media bus formats available at a given sub-device pad applications initialize the pad, which and index fields of struct `v4l2_subdev_mbus_code_enum` and call the `ioctl VIDIOC_SUBDEV_ENUM_MBUS_CODE` ioctl with a pointer to this structure. Drivers fill the rest of the structure or return an EINVAL error code if either the pad or index are invalid. All media bus formats are enumerable by beginning at index zero and incrementing by one until EINVAL is returned.

Available media bus formats may depend on the current 'try' formats at other pads of the sub-device, as well as on the current active links. See `ioctl VIDIOC_SUBDEV_G_FMT`, `VIDIOC_SUBDEV_S_FMT` for more information about the try formats.

v4l2_subdev_mbus_code_enum

Table 235: struct `v4l2_subdev_mbus_code_enum`

__u32	pad	Pad number as reported by the media controller API.
__u32	index	Number of the format in the enumeration, set by the application.
__u32	code	The media bus format code, as defined in <i>Media Bus Formats</i> .
__u32	which	Media bus format codes to be enumerated, from enum <code>v4l2_subdev_format_whence</code> .
__u32	flags	See <i>Subdev Media Bus Code Enumerate Flags</i>
__u32	reserved[7]	Reserved for future extensions. Applications and drivers must set the array to zero.

Table 236: Subdev Media Bus Code Enumerate Flags

V4L2_SUBDEV_MBUS_CODE_CSC_COLORSPACE	0x00000001	The driver allows the application to try to change the default colorspace encoding. The application can ask to configure the colorspace of the subdevice when calling the <code>VIDIOC_SUBDEV_S_FMT</code> ioctl with <code>V4L2_MBUS_FRAMEFMT_SET_CSC</code> set. See <i>Media Bus Formats</i> on how to do this.
V4L2_SUBDEV_MBUS_CODE_CSC_XFER_FUNC	0x00000002	The driver allows the application to try to change the default transform function. The application can ask to configure the transform function of the subdevice when calling the <code>VIDIOC_SUBDEV_S_FMT</code> ioctl with <code>V4L2_MBUS_FRAMEFMT_SET_CSC</code> set. See <i>Media Bus Formats</i> on how to do this.
V4L2_SUBDEV_MBUS_CODE_CSC_YCBCR_ENC	0x00000004	The driver allows the application to try to change the default Y'CbCr encoding. The application can ask to configure the Y'CbCr encoding of the subdevice when calling the <code>VIDIOC_SUBDEV_S_FMT</code> ioctl with <code>V4L2_MBUS_FRAMEFMT_SET_CSC</code> set. See <i>Media Bus Formats</i> on how to do this.
V4L2_SUBDEV_MBUS_CODE_CSC_HSV_ENC	0x00000004	The driver allows the application to try to change the default HSV encoding. The application can ask to configure the HSV encoding of the subdevice when calling the <code>VIDIOC_SUBDEV_S_FMT</code> ioctl with <code>V4L2_MBUS_FRAMEFMT_SET_CSC</code> set. See <i>Media Bus Formats</i> on how to do this.
V4L2_SUBDEV_MBUS_CODE_CSC_QUANTIZATION	0x00000008	The driver allows the application to try to change the default quantization. The application can ask to configure the quantization of the subdevice when calling the <code>VIDIOC_SUBDEV_S_FMT</code> ioctl with <code>V4L2_MBUS_FRAMEFMT_SET_CSC</code> set. See <i>Media Bus Formats</i> on how to do this.

Return Value

On success 0 is returned, on error -1 and the `errno` variable is set appropriately. The generic error codes are described at the [Generic Error Codes](#) chapter.

EINVAL The struct `v4l2_subdev_mbus_code_enum` `pad` references a non-existing pad, or the `index` field is out of bounds.

3.2.7.58 ioctl VIDIOC_SUBDEV_G_CROP, VIDIOC_SUBDEV_S_CROP

Name

VIDIOC_SUBDEV_G_CROP - VIDIOC_SUBDEV_S_CROP - Get or set the crop rectangle on a subdev pad

Synopsis

VIDIOC_SUBDEV_G_CROP

```
int ioctl(int fd, VIDIOC_SUBDEV_G_CROP, struct v4l2_subdev_crop *argp)
```

VIDIOC_SUBDEV_S_CROP

```
int ioctl(int fd, VIDIOC_SUBDEV_S_CROP, const struct v4l2_subdev_crop *argp)
```

Arguments

fd File descriptor returned by `open()`.

argp Pointer to struct `v4l2_subdev_crop`.

Description

Note: This is an *Obsolete API Elements* interface and may be removed in the future. It is superseded by *the selection API*.

To retrieve the current crop rectangle applications set the `pad` field of a struct `v4l2_subdev_crop` to the desired pad number as reported by the media API and the `which` field to `V4L2_SUBDEV_FORMAT_ACTIVE`. They then call the `VIDIOC_SUBDEV_G_CROP` `ioctl` with a pointer to this structure. The driver fills the members of the `rect` field or returns `EINVAL` error code if the input arguments are invalid, or if cropping is not supported on the given pad.

To change the current crop rectangle applications set both the `pad` and `which` fields and all members of the `rect` field. They then call the `VIDIOC_SUBDEV_S_CROP` `ioctl` with a pointer to this structure. The driver verifies the requested crop rectangle, adjusts it based on the hardware capabilities and configures the device. Upon return the struct `v4l2_subdev_crop` contains the current format as would be returned by a `VIDIOC_SUBDEV_G_CROP` call.

Applications can query the device capabilities by setting the `which` field to `V4L2_SUBDEV_FORMAT_TRY`. When set, ‘try’ crop rectangles are not applied to the device by the driver, but are mangled exactly as active crop rectangles and stored in the sub-device file handle. Two applications querying the same sub-device would thus not interact with each other.

If the subdev device node has been registered in read-only mode, calls to `VIDIOC_SUBDEV_S_CROP` are only valid if the `which` field is set to `V4L2_SUBDEV_FORMAT_TRY`, otherwise an error is returned and the `errno` variable is set to `-EPERM`.

Drivers must not return an error solely because the requested crop rectangle doesn’t match the device capabilities. They must instead modify the rectangle to match what the hardware can provide. The modified format should be as close as possible to the original request.

v4l2_subdev_crop

Table 237: struct v4l2_subdev_crop

<code>__u32</code>	<code>pad</code>	Pad number as reported by the media framework.
<code>__u32</code>	<code>which</code>	Crop rectangle to get or set, from enum v4l2_subdev_format_whence .
<code>struct v4l2_rect</code>	<code>rect</code>	Crop rectangle boundaries, in pixels.
<code>__u32</code>	<code>reserved[8]</code>	Reserved for future extensions. Applications and drivers must set the array to zero.

Return Value

On success 0 is returned, on error -1 and the `errno` variable is set appropriately. The generic error codes are described at the [Generic Error Codes](#) chapter.

EBUSY The crop rectangle can’t be changed because the pad is currently busy. This can be caused, for instance, by an active video stream on the pad. The `ioctl` must not be retried without performing another action to fix the problem first. Only returned by `VIDIOC_SUBDEV_S_CROP`

EINVAL The struct `v4l2_subdev_crop` `pad` references a non-existing pad, the `which` field references a non-existing format, or cropping is not supported on the given subdev pad.

EPERM The `VIDIOC_SUBDEV_S_CROP` `ioctl` has been called on a read-only subdevice and the `which` field is set to `V4L2_SUBDEV_FORMAT_ACTIVE`.

3.2.7.59 ioctl VIDIOC_SUBDEV_G_FMT, VIDIOC_SUBDEV_S_FMT

Name

`VIDIOC_SUBDEV_G_FMT` - `VIDIOC_SUBDEV_S_FMT` - Get or set the data format on a subdev pad

Synopsis

VIDIOC_SUBDEV_G_FMT

```
int ioctl(int fd, VIDIOC_SUBDEV_G_FMT, struct v4l2_subdev_format *argp)
```

VIDIOC_SUBDEV_S_FMT

```
int ioctl(int fd, VIDIOC_SUBDEV_S_FMT, struct v4l2_subdev_format *argp)
```

Arguments

fd File descriptor returned by `open()`.

argp Pointer to `struct v4l2_subdev_format`.

Description

These ioctls are used to negotiate the frame format at specific subdev pads in the image pipeline.

To retrieve the current format applications set the `pad` field of a `struct v4l2_subdev_format` to the desired pad number as reported by the media API and the `which` field to `V4L2_SUBDEV_FORMAT_ACTIVE`. When they call the `VIDIOC_SUBDEV_G_FMT` ioctl with a pointer to this structure the driver fills the members of the `format` field.

To change the current format applications set both the `pad` and `which` fields and all members of the `format` field. When they call the `VIDIOC_SUBDEV_S_FMT` ioctl with a pointer to this structure the driver verifies the requested format, adjusts it based on the hardware capabilities and configures the device. Upon return the `struct v4l2_subdev_format` contains the current format as would be returned by a `VIDIOC_SUBDEV_G_FMT` call.

Applications can query the device capabilities by setting the `which` to `V4L2_SUBDEV_FORMAT_TRY`. When set, 'try' formats are not applied to the device by the driver, but are changed exactly as active formats and stored in the sub-device file handle. Two applications querying the same sub-device would thus not interact with each other.

For instance, to try a format at the output pad of a sub-device, applications would first set the try format at the sub-device input with the `VIDIOC_SUBDEV_S_FMT` ioctl. They would then either retrieve the default format at the output pad with the `VIDIOC_SUBDEV_G_FMT` ioctl, or set the desired output pad format with the `VIDIOC_SUBDEV_S_FMT` ioctl and check the returned value.

Try formats do not depend on active formats, but can depend on the current links configuration or sub-device controls value. For instance, a low-pass noise filter might crop pixels at the frame boundaries, modifying its output frame size.

If the subdev device node has been registered in read-only mode, calls to `VIDIOC_SUBDEV_S_FMT` are only valid if the `which` field is set to `V4L2_SUBDEV_FORMAT_TRY`, otherwise an error is returned and the `errno` variable is set to `-EPERM`.

Drivers must not return an error solely because the requested format doesn't match the device capabilities. They must instead modify the format to match what the hardware can provide. The modified format should be as close as possible to the original request.

`v4l2_subdev_format`

Table 238: struct v4l2_subdev_format

__u32	pad	Pad number as reported by the media controller API.
__u32	which	Format to modified, from enum v4l2_subdev_format_whence .
struct v4l2_mbus_framefmt	format	Definition of an image format, see v4l2_mbus_framefmt for details.
__u32	reserved[8]	Reserved for future extensions. Applications and drivers must set the array to zero.

Table 239: enum v4l2_subdev_format_whence

V4L2_SUBDEV_FORMAT_TRY	0	Try formats, used for querying device capabilities.
V4L2_SUBDEV_FORMAT_ACTIVE	1	Active formats, applied to the hardware.

Return Value

On success 0 is returned, on error -1 and the `errno` variable is set appropriately. The generic error codes are described at the [Generic Error Codes](#) chapter.

EBUSY The format can't be changed because the pad is currently busy. This can be caused, for instance, by an active video stream on the pad. The `ioctl` must not be retried without performing another action to fix the problem first. Only returned by `VIDIOC_SUBDEV_S_FMT`

EINVAL The struct `v4l2_subdev_format` `pad` references a non-existing pad, or the `which` field references a non-existing format.

EPERM The `VIDIOC_SUBDEV_S_FMT` `ioctl` has been called on a read-only subdevice and the `which` field is set to `V4L2_SUBDEV_FORMAT_ACTIVE`.

On success 0 is returned, on error -1 and the `errno` variable is set appropriately. The generic error codes are described at the [Generic Error Codes](#) chapter.

3.2.7.60 `ioctl` `VIDIOC_SUBDEV_G_FRAME_INTERVAL`, `VIDIOC_SUBDEV_S_FRAME_INTERVAL`

Name

`VIDIOC_SUBDEV_G_FRAME_INTERVAL` - `VIDIOC_SUBDEV_S_FRAME_INTERVAL` - Get or set the frame interval on a subdev pad

Synopsis

VIDIOC_SUBDEV_G_FRAME_INTERVAL

```
int ioctl(int fd, VIDIOC_SUBDEV_G_FRAME_INTERVAL, struct v4l2_subdev_frame_interval *argp)
```

VIDIOC_SUBDEV_S_FRAME_INTERVAL

```
int ioctl(int fd, VIDIOC_SUBDEV_S_FRAME_INTERVAL, struct v4l2_subdev_frame_interval *argp)
```

Arguments

fd File descriptor returned by `open()`.

argp Pointer to `struct v4l2_subdev_frame_interval`.

Description

These `ioctl`s are used to get and set the frame interval at specific subdev pads in the image pipeline. The frame interval only makes sense for sub-devices that can control the frame period on their own. This includes, for instance, image sensors and TV tuners. Sub-devices that don't support frame intervals must not implement these `ioctl`s.

To retrieve the current frame interval applications set the `pad` field of a `struct v4l2_subdev_frame_interval` to the desired pad number as reported by the media controller API. When they call the `VIDIOC_SUBDEV_G_FRAME_INTERVAL` `ioctl` with a pointer to this structure the driver fills the members of the `interval` field.

To change the current frame interval applications set both the `pad` field and all members of the `interval` field. When they call the `VIDIOC_SUBDEV_S_FRAME_INTERVAL` `ioctl` with a pointer to this structure the driver verifies the requested interval, adjusts it based on the hardware capabilities and configures the device. Upon return the `struct v4l2_subdev_frame_interval` contains the current frame interval as would be returned by a `VIDIOC_SUBDEV_G_FRAME_INTERVAL` call.

Calling `VIDIOC_SUBDEV_S_FRAME_INTERVAL` on a subdev device node that has been registered in read-only mode is not allowed. An error is returned and the `errno` variable is set to `-EPERM`.

Drivers must not return an error solely because the requested interval doesn't match the device capabilities. They must instead modify the interval to match what the hardware can provide. The modified interval should be as close as possible to the original request.

Changing the frame interval shall never change the format. Changing the format, on the other hand, may change the frame interval.

Sub-devices that support the frame interval `ioctl`s should implement them on a single pad only. Their behaviour when supported on multiple pads of the same sub-device is not defined.

`v4l2_subdev_frame_interval`

Table 240: struct v4l2_subdev_frame_interval

__u32	pad	Pad number as reported by the media controller API.
struct v4l2_fract	interval	Period, in seconds, between consecutive video frames.
__u32	reserved[9]	Reserved for future extensions. Applications and drivers must set the array to zero.

Return Value

On success 0 is returned, on error -1 and the `errno` variable is set appropriately. The generic error codes are described at the [Generic Error Codes](#) chapter.

EBUSY The frame interval can't be changed because the pad is currently busy. This can be caused, for instance, by an active video stream on the pad. The `ioctl` must not be retried without performing another action to fix the problem first. Only returned by `VIDIOC_SUBDEV_S_FRAME_INTERVAL`

EINVAL The struct `v4l2_subdev_frame_interval` pad references a non-existing pad, or the pad doesn't support frame intervals.

EPERM The `VIDIOC_SUBDEV_S_FRAME_INTERVAL` `ioctl` has been called on a read-only subdevice.

3.2.7.61 `ioctl` `VIDIOC_SUBDEV_G_SELECTION`, `VIDIOC_SUBDEV_S_SELECTION`

Name

`VIDIOC_SUBDEV_G_SELECTION` - `VIDIOC_SUBDEV_S_SELECTION` - Get or set selection rectangles on a subdev pad

Synopsis

`VIDIOC_SUBDEV_G_SELECTION`

```
int ioctl(int fd, VIDIOC_SUBDEV_G_SELECTION, struct v4l2_subdev_selection *argp)
```

`VIDIOC_SUBDEV_S_SELECTION`

```
int ioctl(int fd, VIDIOC_SUBDEV_S_SELECTION, struct v4l2_subdev_selection *argp)
```


Arguments

fd File descriptor returned by `open()`.

argp Pointer to struct `v4l2_subdev_selection`.

Description

The selections are used to configure various image processing functionality performed by the subdevs which affect the image size. This currently includes cropping, scaling and composition.

The selection API replaces *the old subdev crop API*. All the function of the crop API, and more, are supported by the selections API.

See *Sub-device Interface* for more information on how each selection target affects the image processing pipeline inside the subdevice.

If the subdev device node has been registered in read-only mode, calls to `VIDIOC_SUBDEV_S_SELECTION` are only valid if the `which` field is set to `V4L2_SUBDEV_FORMAT_TRY`, otherwise an error is returned and the `errno` variable is set to `-EPERM`.

Types of selection targets

There are two types of selection targets: actual and bounds. The actual targets are the targets which configure the hardware. The `BOUNDS` target will return a rectangle that contain all possible actual rectangles.

Discovering supported features

To discover which targets are supported, the user can perform `VIDIOC_SUBDEV_G_SELECTION` on them. Any unsupported target will return `EINVAL`.

Selection targets and flags are documented in *Common selection definitions*.

`v4l2_subdev_selection`

Table 241: struct `v4l2_subdev_selection`

<code>__u32</code>	<code>which</code>	Active or try selection, from enum <i>v4l2_subdev_format_whence</i> .
<code>__u32</code>	<code>pad</code>	Pad number as reported by the media framework.
<code>__u32</code>	<code>target</code>	Target selection rectangle. See <i>Common selection definitions</i> .
<code>__u32</code>	<code>flags</code>	Flags. See <i>Selection flags</i> .
struct <code>v4l2_rect</code>	<code>r</code>	Selection rectangle, in pixels.
<code>__u32</code>	<code>reserved[8]</code>	Reserved for future extensions. Applications and drivers must set the array to zero.

Return Value

On success 0 is returned, on error -1 and the `errno` variable is set appropriately. The generic error codes are described at the [Generic Error Codes](#) chapter.

EBUSY The selection rectangle can't be changed because the pad is currently busy. This can be caused, for instance, by an active video stream on the pad. The `ioctl` must not be retried without performing another action to fix the problem first. Only returned by `VIDIOC_SUBDEV_S_SELECTION`

EINVAL The struct `v4l2_subdev_selection` pad references a non-existing pad, the `which` field references a non-existing format, or the selection target is not supported on the given subdev pad.

EPERM The `VIDIOC_SUBDEV_S_SELECTION` `ioctl` has been called on a read-only subdevice and the `which` field is set to `V4L2_SUBDEV_FORMAT_ACTIVE`.

3.2.7.62 `ioctl` `VIDIOC_SUBDEV_QUERYCAP`

Name

`VIDIOC_SUBDEV_QUERYCAP` - Query sub-device capabilities

Synopsis

`VIDIOC_SUBDEV_QUERYCAP`

```
int ioctl(int fd, VIDIOC_SUBDEV_QUERYCAP, struct v4l2_subdev_capability *argp)
```

Arguments

fd File descriptor returned by `open()`.

argp Pointer to struct `v4l2_subdev_capability`.

Description

All V4L2 sub-devices support the `VIDIOC_SUBDEV_QUERYCAP` `ioctl`. It is used to identify kernel devices compatible with this specification and to obtain information about driver and hardware capabilities. The `ioctl` takes a pointer to a struct `v4l2_subdev_capability` which is filled by the driver. When the driver is not compatible with this specification the `ioctl` returns `ENOTTY` error code.

`v4l2_subdev_capability`

Table 242: struct v4l2_subdev_capability

__u32	version	Version number of the driver. The version reported is provided by the V4L2 subsystem following the kernel numbering scheme. However, it may not always return the same version as the kernel if, for example, a stable or distribution-modified kernel uses the V4L2 stack from a newer kernel. The version number is formatted using the <code>KERNEL_VERSION()</code> macro:
<pre>#define KERNEL_VERSION(a,b,c) (((a) << 16) + ((b) << 8) + (c)) __u32 version = KERNEL_VERSION(0, 8, 1); printf ("Version: %u.%u.%u\\n", (version >> 16) & 0xFF, (version >> 8) & 0xFF, version & 0xFF);</pre>		
__u32	capabilities	Sub-device capabilities of the opened device, see <i>Sub-Device Capabilities Flags</i> .
__u32	reserved[14]	Reserved for future extensions. Set to 0 by the V4L2 core.

Table 243: Sub-Device Capabilities Flags

V4L2_SUBDEV_CAP_RO_SUBDEV	0x00000001	The sub-device device node is registered in read-only mode. Access to the sub-device ioctls that modify the device state is restricted. Refer to each individual sub-device ioctl documentation for a description of which restrictions apply to a read-only sub-device.
---------------------------	------------	--

Return Value

On success 0 is returned, on error -1 and the `errno` variable is set appropriately. The generic error codes are described at the *Generic Error Codes* chapter.

ENOTTY The device node is not a V4L2 sub-device.

3.2.7.63 ioctl VIDIIOC_SUBSCRIBE_EVENT, VIDIIOC_UNSUBSCRIBE_EVENT

Name

VIDIIOC_SUBSCRIBE_EVENT - VIDIIOC_UNSUBSCRIBE_EVENT - Subscribe or unsubscribe event

Synopsis

VIDIOC_SUBSCRIBE_EVENT

```
int ioctl(int fd, VIDIOC_SUBSCRIBE_EVENT, struct v4l2_event_subscription *argp)
```

VIDIOC_UNSUBSCRIBE_EVENT

```
int ioctl(int fd, VIDIOC_UNSUBSCRIBE_EVENT, struct v4l2_event_subscription *argp)
```

Arguments

fd File descriptor returned by `open()`.

argp Pointer to struct `v4l2_event_subscription`.

Description

Subscribe or unsubscribe V4L2 event. Subscribed events are dequeued by using the *ioctl VID-IOC_DQEVENT* ioctl.

`v4l2_event_subscription`

Table 244: struct `v4l2_event_subscription`

__u32	type	Type of the event, see <i>Event Types</i> . Note: <code>V4L2_EVENT_ALL</code> can be used with <i>VID-IOC_UNSUBSCRIBE_EVENT</i> for unsubscribing all events at once.
__u32	id	ID of the event source. If there is no ID associated with the event source, then set this to 0. Whether or not an event needs an ID depends on the event type.
__u32	flags	Event flags, see <i>Event Flags</i> .
__u32	reserved[5]	Reserved for future extensions. Drivers and applications must set the array to zero.

Table 245: Event Flags

V4L2_EVENT_SUB_FL_SEND_INITIAL	0x0001	When this event is subscribed an initial event will be sent containing the current status. This only makes sense for events that are triggered by a status change such as V4L2_EVENT_CTRL. Other events will ignore this flag.
V4L2_EVENT_SUB_FL_ALLOW_FEEDBACK	0x0002	If set, then events directly caused by an ioctl will also be sent to the filehandle that called that ioctl. For example, changing a control using <code>VIDIOC_S_CTRL</code> will cause a V4L2_EVENT_CTRL to be sent back to that same filehandle. Normally such events are suppressed to prevent feedback loops where an application changes a control to a one value and then another, and then receives an event telling it that that control has changed to the first value. Since it can't tell whether that event was caused by another application or by the <code>VIDIOC_S_CTRL</code> call it is hard to decide whether to set the control to the value in the event, or ignore it. Think carefully when you set this flag so you won't get into situations like that.

Return Value

On success 0 is returned, on error -1 and the `errno` variable is set appropriately. The generic error codes are described at the [Generic Error Codes](#) chapter.

3.2.7.64 V4L2 mmap()

Name

v4l2-mmap - Map device memory into application address space

Synopsis

```
#include <unistd.h>
#include <sys/mman.h>
```

```
void *mmap(void *start, size_t length, int prot, int flags, int fd, off_t offset)
```

Arguments

start Map the buffer to this address in the application's address space. When the `MAP_FIXED` flag is specified, `start` must be a multiple of the `pagesize` and `mmap` will fail when the specified address cannot be used. Use of this option is discouraged; applications should just specify a `NULL` pointer here.

length Length of the memory area to map. This must be the same value as returned by the driver in the struct `v4l2_buffer` `length` field for the single-planar API, and the same value as returned by the driver in the struct `v4l2_plane` `length` field for the multi-planar API.

prot The `prot` argument describes the desired memory protection. Regardless of the device type and the direction of data exchange it should be set to `PROT_READ | PROT_WRITE`, permitting read and write access to image buffers. Drivers should support at least this combination of flags.

Note:

1. The Linux `videobuf` kernel module, which is used by some drivers supports only `PROT_READ | PROT_WRITE`. When the driver does not support the desired protection, the `mmap()` function fails.
2. Device memory accesses (e. g. the memory on a graphics card with video capturing hardware) may incur a performance penalty compared to main memory accesses, or reads may be significantly slower than writes or vice versa. Other I/O methods may be more efficient in such case.

flags The `flags` parameter specifies the type of the mapped object, mapping options and whether modifications made to the mapped copy of the page are private to the process or are to be shared with other references.

`MAP_FIXED` requests that the driver selects no other address than the one specified. If the specified address cannot be used, `mmap()` will fail. If `MAP_FIXED` is specified, `start` must be a multiple of the `pagesize`. Use of this option is discouraged.

One of the `MAP_SHARED` or `MAP_PRIVATE` flags must be set. `MAP_SHARED` allows applications to share the mapped memory with other (e. g. child-) processes.

Note: The Linux `videobuf` module which is used by some drivers supports only `MAP_SHARED`. `MAP_PRIVATE` requests copy-on-write semantics. V4L2 applications should not set the `MAP_PRIVATE`, `MAP_DENYWRITE`, `MAP_EXECUTABLE` or `MAP_ANON` flags.

fd File descriptor returned by `open()`.

offset Offset of the buffer in device memory. This must be the same value as returned by the driver in the struct `v4l2_buffer` `m` union `offset` field for the single-planar API, and the same value as returned by the driver in the struct `v4l2_plane` `m` union `mem_offset` field for the multi-planar API.

Description

The `mmap()` function asks to map `length` bytes starting at `offset` in the memory of the device specified by `fd` into the application address space, preferably at address `start`. This latter address is a hint only, and is usually specified as 0.

Suitable length and offset parameters are queried with the *ioctl* `VIDIOC_QUERYBUF` ioctl. Buffers must be allocated with the *ioctl* `VIDIOC_REQBUFS` ioctl before they can be queried.

To unmap buffers the `munmap()` function is used.

Return Value

On success `mmap()` returns a pointer to the mapped buffer. On error `MAP_FAILED` (-1) is returned, and the `errno` variable is set appropriately. Possible error codes are:

EBADF `fd` is not a valid file descriptor.

EACCES `fd` is not open for reading and writing.

EINVAL The `start` or `length` or `offset` are not suitable. (E. g. they are too large, or not aligned on a `PAGESIZE` boundary.)

The `flags` or `prot` value is not supported.

No buffers have been allocated with the *ioctl* `VIDIOC_REQBUFS` ioctl.

ENOMEM Not enough physical or virtual memory was available to complete the request.

3.2.7.65 V4L2 munmap()

Name

v4l2-munmap - Unmap device memory

Synopsis

```
#include <unistd.h>
#include <sys/mman.h>
```

```
int munmap(void *start, size_t length)
```

Arguments

start Address of the mapped buffer as returned by the `mmap()` function.

length Length of the mapped buffer. This must be the same value as given to `mmap()` and returned by the driver in the struct `v4l2_buffer` `length` field for the single-planar API and in the struct `v4l2_plane` `length` field for the multi-planar API.

Description

Unmaps a previously with the `mmap()` function mapped buffer and frees it, if possible.

Return Value

On success `munmap()` returns 0, on failure -1 and the `errno` variable is set appropriately:

EINVAL The start or length is incorrect, or no buffers have been mapped yet.

3.2.7.66 V4L2 open()

Name

v4l2-open - Open a V4L2 device

Synopsis

```
#include <fcntl.h>
```

```
int open(const char *device_name, int flags)
```

Arguments

device_name Device to be opened.

flags Open flags. Access mode must be `O_RDWR`. This is just a technicality, input devices still support only reading and output devices only writing.

When the `O_NONBLOCK` flag is given, the `read()` function and the `VIDIOC_DQBUF` ioctl will return the `EAGAIN` error code when no data is available or no buffer is in the driver outgoing queue, otherwise these functions block until data becomes available. All V4L2 drivers exchanging data with applications must support the `O_NONBLOCK` flag.

Other flags have no effect.

Description

To open a V4L2 device applications call `open()` with the desired device name. This function has no side effects; all data format parameters, current input or output, control values or other properties remain unchanged. At the first `open()` call after loading the driver they will be reset to default values, drivers are never in an undefined state.

Return Value

On success `open()` returns the new file descriptor. On error -1 is returned, and the `errno` variable is set appropriately. Possible error codes are:

EACCES The caller has no permission to access the device.

EBUSY The driver does not support multiple opens and the device is already in use.

ENXIO No device corresponding to this device special file exists.

ENOMEM Not enough kernel memory was available to complete the request.

EMFILE The process already has the maximum number of files open.

ENFILE The limit on the total number of files open on the system has been reached.

3.2.7.67 V4L2 `poll()`

Name

`v4l2-poll` - Wait for some event on a file descriptor

Synopsis

```
#include <sys/poll.h>
```

```
int poll(struct pollfd *ufds, unsigned int nfds, int timeout)
```

Arguments

Description

With the `poll()` function applications can suspend execution until the driver has captured data or is ready to accept data for output.

When streaming I/O has been negotiated this function waits until a buffer has been filled by the capture device and can be dequeued with the `VIDIOC_DQBUF` ioctl. For output devices this function waits until the device is ready to accept a new buffer to be queued up with the `VIDIOC_QBUF` ioctl for display. When buffers are already in the outgoing queue of the driver (capture) or the incoming queue isn't full (display) the function returns immediately.

On success `poll()` returns the number of file descriptors that have been selected (that is, file descriptors for which the `revents` field of the respective `struct pollfd` structure is non-zero).

Capture devices set the `POLLIN` and `POLLRDNORM` flags in the `revents` field, output devices the `POLLOUT` and `POLLWRNORM` flags. When the function timed out it returns a value of zero, on failure it returns -1 and the `errno` variable is set appropriately. When the application did not call `VIDIOC_STREAMON` the `poll()` function succeeds, but sets the `POLLERR` flag in the `revents` field. When the application has called `VIDIOC_STREAMON` for a capture device but hasn't yet called `VIDIOC_QBUF`, the `poll()` function succeeds and sets the `POLLERR` flag in the `revents` field. For output devices this same situation will cause `poll()` to succeed as well, but it sets the `POLLOUT` and `POLLWRNORM` flags in the `revents` field.

If an event occurred (see *ioctl* `VIDIOC_DQEVENT`) then `POLLPRI` will be set in the `revents` field and `poll()` will return.

When use of the `read()` function has been negotiated and the driver does not capture yet, the `poll()` function starts capturing. When that fails it returns a `POLLERR` as above. Otherwise it waits until data has been captured and can be read. When the driver captures continuously (as opposed to, for example, still images) the function may return immediately.

When use of the `write()` function has been negotiated and the driver does not stream yet, the `poll()` function starts streaming. When that fails it returns a `POLLERR` as above. Otherwise it waits until the driver is ready for a non-blocking `write()` call.

If the caller is only interested in events (just `POLLPRI` is set in the `events` field), then `poll()` will *not* start streaming if the driver does not stream yet. This makes it possible to just poll for events and not for buffers.

All drivers implementing the `read()` or `write()` function or streaming I/O must also support the `poll()` function.

For more details see the `poll()` manual page.

Return Value

On success, `poll()` returns the number structures which have non-zero `revents` fields, or zero if the call timed out. On error -1 is returned, and the `errno` variable is set appropriately:

EBADF One or more of the `ufds` members specify an invalid file descriptor.

EBUSY The driver does not support multiple read or write streams and the device is already in use.

EFAULT `ufds` references an inaccessible memory area.

EINTR The call was interrupted by a signal.

EINVAL The `nfds` value exceeds the `RLIMIT_NOFILE` value. Use `getrlimit()` to obtain this value.

3.2.7.68 V4L2 read()

Name

v4l2-read - Read from a V4L2 device

Synopsis

```
#include <unistd.h>
```

```
ssize_t read(int fd, void *buf, size_t count)
```

Arguments

fd File descriptor returned by `open()`.

buf Buffer to be filled

count Max number of bytes to read

Description

`read()` attempts to read up to `count` bytes from file descriptor `fd` into the buffer starting at `buf`. The layout of the data in the buffer is discussed in the respective device interface section, see `##`. If `count` is zero, `read()` returns zero and has no other results. If `count` is greater than `SSIZE_MAX`, the result is unspecified. Regardless of the `count` value each `read()` call will provide at most one frame (two fields) worth of data.

By default `read()` blocks until data becomes available. When the `O_NONBLOCK` flag was given to the `open()` function it returns immediately with an `EAGAIN` error code when no data is available. The `select()` or `poll()` functions can always be used to suspend execution until data becomes available. All drivers supporting the `read()` function must also support `select()` and `poll()`.

Drivers can implement read functionality in different ways, using a single or multiple buffers and discarding the oldest or newest frames once the internal buffers are filled.

`read()` never returns a “snapshot” of a buffer being filled. Using a single buffer the driver will stop capturing when the application starts reading the buffer until the read is finished. Thus only the period of the vertical blanking interval is available for reading, or the capture rate must fall below the nominal frame rate of the video standard.

The behavior of `read()` when called during the active picture period or the vertical blanking separating the top and bottom field depends on the discarding policy. A driver discarding the oldest frames keeps capturing into an internal buffer, continuously overwriting the previously, not read frame, and returns the frame being received at the time of the `read()` call as soon as it is complete.

A driver discarding the newest frames stops capturing until the next `read()` call. The frame being received at `read()` time is discarded, returning the following frame instead. Again this implies a reduction of the capture rate to one half or less of the nominal frame rate. An example of this model is the video read mode of the `bttv` driver, initiating a DMA to user memory when `read()` is called and returning when the DMA finished.

In the multiple buffer model drivers maintain a ring of internal buffers, automatically advancing to the next free buffer. This allows continuous capturing when the application can empty the buffers fast enough. Again, the behavior when the driver runs out of free buffers depends on the discarding policy.

Applications can get and set the number of buffers used internally by the driver with the `VIDIOC_G_PARM` and `VIDIOC_S_PARM` ioctls. They are optional, however. The discarding policy is not reported and cannot be changed. For minimum requirements see *Interfaces*.

Return Value

On success, the number of bytes read is returned. It is not an error if this number is smaller than the number of bytes requested, or the amount of data required for one frame. This may happen for example because `read()` was interrupted by a signal. On error, -1 is returned, and the `errno` variable is set appropriately. In this case the next read will start at the beginning of a new frame. Possible error codes are:

EAGAIN Non-blocking I/O has been selected using `O_NONBLOCK` and no data was immediately available for reading.

EBADF `fd` is not a valid file descriptor or is not open for reading, or the process already has the maximum number of files open.

EBUSY The driver does not support multiple read streams and the device is already in use.

EFAULT `buf` references an inaccessible memory area.

EINTR The call was interrupted by a signal before any data was read.

EIO I/O error. This indicates some hardware problem or a failure to communicate with a remote device (USB camera etc.).

EINVAL The `read()` function is not supported by this driver, not on this device, or generally not on this type of device.

3.2.7.69 V4L2 select()

Name

v4l2-select - Synchronous I/O multiplexing

Synopsis

```
#include <sys/time.h>
#include <sys/types.h>
#include <unistd.h>
```

```
int select(int nfds, fd_set *readfds, fd_set *writefds, fd_set *exceptfds, struct
           timeval *timeout)
```

Arguments

nfds The highest-numbered file descriptor in any of the three sets, plus 1.

readfds File descriptions to be watched if a `read()` call won't block.

writelfds File descriptions to be watched if a `write()` won't block.

exceptfds File descriptions to be watched for V4L2 events.

timeout Maximum time to wait.

Description

With the `select()` function applications can suspend execution until the driver has captured data or is ready to accept data for output.

When streaming I/O has been negotiated this function waits until a buffer has been filled or displayed and can be dequeued with the `VIDIOC_DQBUF` ioctl. When buffers are already in the outgoing queue of the driver the function returns immediately.

On success `select()` returns the total number of bits set in `fd_set`. When the function timed out it returns a value of zero. On failure it returns -1 and the `errno` variable is set appropriately. When the application did not call `ioctl VIDIOC_QBUF`, `VIDIOC_DQBUF` or `ioctl VIDIOC_STREAMON`, `VIDIOC_STREAMOFF` yet the `select()` function succeeds, setting the bit of the file descriptor in `readfds` or `writelfds`, but subsequent `VIDIOC_DQBUF` calls will fail.¹

When use of the `read()` function has been negotiated and the driver does not capture yet, the `select()` function starts capturing. When that fails, `select()` returns successful and a subsequent `read()` call, which also attempts to start capturing, will return an appropriate error code. When the driver captures continuously (as opposed to, for example, still images) and data is already available the `select()` function returns immediately.

When use of the `write()` function has been negotiated the `select()` function just waits until the driver is ready for a non-blocking `write()` call.

All drivers implementing the `read()` or `write()` function or streaming I/O must also support the `select()` function.

For more details see the `select()` manual page.

Return Value

On success, `select()` returns the number of descriptors contained in the three returned descriptor sets, which will be zero if the timeout expired. On error -1 is returned, and the `errno` variable is set appropriately; the sets and `timeout` are undefined. Possible error codes are:

EBADF One or more of the file descriptor sets specified a file descriptor that is not open.

EBUSY The driver does not support multiple read or write streams and the device is already in use.

EFAULT The `readfds`, `writelfds`, `exceptfds` or `timeout` pointer references an inaccessible memory area.

¹ The Linux kernel implements `select()` like the `poll()` function, but `select()` cannot return a `POLLERR`.

EINTR The call was interrupted by a signal.

EINVAL The `nfds` argument is less than zero or greater than `FD_SETSIZE`.

3.2.7.70 V4L2 write()

Name

v4l2-write - Write to a V4L2 device

Synopsis

```
#include <unistd.h>
```

```
ssize_t write(int fd, void *buf, size_t count)
```

Arguments

fd File descriptor returned by `open()`.

buf Buffer with data to be written

count Number of bytes at the buffer

Description

`write()` writes up to `count` bytes to the device referenced by the file descriptor `fd` from the buffer starting at `buf`. When the hardware outputs are not active yet, this function enables them. When `count` is zero, `write()` returns 0 without any other effect.

When the application does not provide more data in time, the previous video frame, raw VBI image, sliced VPS or WSS data is displayed again. Sliced Teletext or Closed Caption data is not repeated, the driver inserts a blank line instead.

Return Value

On success, the number of bytes written are returned. Zero indicates nothing was written. On error, -1 is returned, and the `errno` variable is set appropriately. In this case the next write will start at the beginning of a new frame. Possible error codes are:

EAGAIN Non-blocking I/O has been selected using the `O_NONBLOCK` flag and no buffer space was available to write the data immediately.

EBADF `fd` is not a valid file descriptor or is not open for writing.

EBUSY The driver does not support multiple write streams and the device is already in use.

EFAULT `buf` references an inaccessible memory area.

EINTR The call was interrupted by a signal before any data was written.

EIO I/O error. This indicates some hardware problem.

EINVAL The `write()` function is not supported by this driver, not on this device, or generally not on this type of device.

3.2.8 Common definitions for V4L2 and V4L2 subdev interfaces

3.2.8.1 Common selection definitions

While the *V4L2 selection API* and *V4L2 subdev selection APIs* are very similar, there's one fundamental difference between the two. On sub-device API, the selection rectangle refers to the media bus format, and is bound to a sub-device's pad. On the V4L2 interface the selection rectangles refer to the in-memory pixel format.

This section defines the common definitions of the selection interfaces on the two APIs.

Selection targets

The precise meaning of the selection targets may be dependent on which of the two interfaces they are used.

Table 246: Selection target definitions

Target name	id	Definition	Valid for V4L2	Valid for V4L2 sub-dev
V4L2_SEL_TGT_CROP	0x0000	Crop rectangle. Defines the cropped area.	Yes	Yes
V4L2_SEL_TGT_CROP_DEFAULT	0x0001	Suggested cropping rectangle that covers the "whole picture". This includes only active pixels and excludes other non-active pixels such as black pixels.	Yes	Yes
V4L2_SEL_TGT_CROP_BOUNDS	0x0002	Bounds of the crop rectangle. All valid crop rectangles fit inside the crop bounds rectangle.	Yes	Yes
V4L2_SEL_TGT_NATIVE_SIZE	0x0003	The native size of the device, e.g. a sensor's pixel array. left and top fields are zero for this target.	Yes	Yes
V4L2_SEL_TGT_COMPOSE	0x0100	Compose rectangle. Used to configure scaling and composition.	Yes	Yes
V4L2_SEL_TGT_COMPOSE_DEFAULT	0x0101	Suggested composition rectangle that covers the "whole picture".	Yes	No
V4L2_SEL_TGT_COMPOSE_BOUNDS	0x0102	Bounds of the compose rectangle. All valid compose rectangles fit inside the compose bounds rectangle.	Yes	Yes
V4L2_SEL_TGT_COMPOSE_PADDED	0x0103	The active area and all padding pixels that are inserted or modified by hardware.	Yes	No

Selection flags

Table 247: Selection flag definitions

Flag name	id	Definition	Valid for V4L2	Valid for V4L2 sub-dev
V4L2_SEL_FLAG_GE	(1 << 0)	Suggest the driver it should choose greater or equal rectangle (in size) than was requested. Albeit the driver may choose a lesser size, it will only do so due to hardware limitations. Without this flag (and V4L2_SEL_FLAG_LE) the behaviour is to choose the closest possible rectangle.	Yes	Yes
V4L2_SEL_FLAG_LE	(1 << 1)	Suggest the driver it should choose lesser or equal rectangle (in size) than was requested. Albeit the driver may choose a greater size, it will only do so due to hardware limitations.	Yes	Yes
V4L2_SEL_FLAG_KEEP_CONFIG	(1 << 2)	The configuration must not be propagated to any further processing steps. If this flag is not given, the configuration is propagated inside the subdevice to all further processing steps.	No	Yes

3.2.9 Video For Linux Two Header File

3.2.9.1 videodev2.h

```

/* SPDX-License-Identifier: ((GPL-2.0+ WITH Linux-syscall-note) OR
↳ BSD-3-Clause) */
/*
 * Video for Linux Two header file
 *
 * Copyright (C) 1999-2012 the contributors
 *
 * This program is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation; either version 2 of the License, or
 * (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
 * GNU General Public License for more details.
 *
 * Alternatively you can redistribute this file under the terms of the

```



```

* BSD license as stated below:
*
* Redistribution and use in source and binary forms, with or without
* modification, are permitted provided that the following conditions
* are met:
* 1. Redistributions of source code must retain the above copyright
*   notice, this list of conditions and the following disclaimer.
* 2. Redistributions in binary form must reproduce the above copyright
*   notice, this list of conditions and the following disclaimer in
*   the documentation and/or other materials provided with the
*   distribution.
* 3. The names of its contributors may not be used to endorse or promote
*   products derived from this software without specific prior written
*   permission.
*
* THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
* "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
* LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR
* A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT
* OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
* SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED
* TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR
* PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF
* LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING
* NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS
* SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
*
*   Header file for v4l or V4L2 drivers and applications
* with public API.
* All kernel-specific stuff were moved to media/v4l2-dev.h, so
* no #if __KERNEL tests are allowed here
*
*   See https://linuxtv.org for more info
*
*   Author: Bill Dirks <bill@thedirks.org>
*           Justin Schoeman
*           Hans Verkuil <hverkuil@xs4all.nl>
*           et al.
*/
#endif __UAPI__LINUX_VIDEODEV2_H
#define __UAPI__LINUX_VIDEODEV2_H

#ifndef __KERNEL__
#include <sys/time.h>
#endif
#include <linux/compiler.h>
#include <linux/ioctl.h>
#include <linux/types.h>
#include <linux/v4l2-common.h>
#include <linux/v4l2-controls.h>

```

```

/*
 * Common stuff for both V4L1 and V4L2
 * Moved from videodev.h
 */
#define VIDEO_MAX_FRAME          32
#define VIDEO_MAX_PLANES        8

/*
 * M I S C E L L A N E O U S
 */

/* Four-character-code (FOURCC) */
#define v4l2_fourcc(a, b, c, d)\
    (((u32)(a) | ((u32)(b) << 8) | ((u32)(c) << 16) | ((u32)(d) << 24))
    < 24))
#define v4l2_fourcc_be(a, b, c, d)    (v4l2_fourcc(a, b, c, d) | (1U << 31))

/*
 * E N U M S
 */
enum v4l2_field {
    V4L2_FIELD_ANY                = 0, /* driver can choose from none,
                                        top, bottom, interlaced
                                        depending on whatever it thinks
                                        is approximate ... */
    V4L2_FIELD_NONE               = 1, /* this device has no fields ... */
    V4L2_FIELD_TOP                = 2, /* top field only */
    V4L2_FIELD_BOTTOM             = 3, /* bottom field only */
    V4L2_FIELD_INTERLACED        = 4, /* both fields interlaced */
    V4L2_FIELD_SEQ_TB            = 5, /* both fields sequential into one
                                        buffer, top-bottom order */
    V4L2_FIELD_SEQ_BT            = 6, /* same as above + bottom-top order */
    V4L2_FIELD_ALTERNATE         = 7, /* both fields alternating into
                                        separate buffers */
    V4L2_FIELD_INTERLACED_TB     = 8, /* both fields interlaced, top field
                                        first and the top field is
                                        transmitted first */
    V4L2_FIELD_INTERLACED_BT     = 9, /* both fields interlaced, top field
                                        first and the bottom field is
                                        transmitted first */
};
#define V4L2_FIELD_HAS_TOP(field) \
    ((field) == V4L2_FIELD_TOP ||\
     (field) == V4L2_FIELD_INTERLACED ||\
     (field) == V4L2_FIELD_INTERLACED_TB ||\
     (field) == V4L2_FIELD_INTERLACED_BT ||\
     (field) == V4L2_FIELD_SEQ_TB ||\
     (field) == V4L2_FIELD_SEQ_BT)
#define V4L2_FIELD_HAS_BOTTOM(field) \
    ((field) == V4L2_FIELD_BOTTOM ||\
     (field) == V4L2_FIELD_INTERLACED ||\

```

```

        (field) == V4L2_FIELD_INTERLACED_TB ||\
        (field) == V4L2_FIELD_INTERLACED_BT ||\
        (field) == V4L2_FIELD_SEQ_TB    ||\
        (field) == V4L2_FIELD_SEQ_BT)
#define V4L2_FIELD_HAS_BOTH(field)      \
    ((field) == V4L2_FIELD_INTERLACED ||\
     (field) == V4L2_FIELD_INTERLACED_TB ||\
     (field) == V4L2_FIELD_INTERLACED_BT ||\
     (field) == V4L2_FIELD_SEQ_TB ||\
     (field) == V4L2_FIELD_SEQ_BT)
#define V4L2_FIELD_HAS_T_OR_B(field)    \
    ((field) == V4L2_FIELD_BOTTOM ||\
     (field) == V4L2_FIELD_TOP ||\
     (field) == V4L2_FIELD_ALTERNATE)
#define V4L2_FIELD_IS_INTERLACED(field) \
    ((field) == V4L2_FIELD_INTERLACED ||\
     (field) == V4L2_FIELD_INTERLACED_TB ||\
     (field) == V4L2_FIELD_INTERLACED_BT)
#define V4L2_FIELD_IS_SEQUENTIAL(field) \
    ((field) == V4L2_FIELD_SEQ_TB ||\
     (field) == V4L2_FIELD_SEQ_BT)

enum v4l2_buf_type {
    V4L2_BUF_TYPE_VIDEO_CAPTURE          = 1,
    V4L2_BUF_TYPE_VIDEO_OUTPUT           = 2,
    V4L2_BUF_TYPE_VIDEO_OVERLAY          = 3,
    V4L2_BUF_TYPE_VBI_CAPTURE            = 4,
    V4L2_BUF_TYPE_VBI_OUTPUT             = 5,
    V4L2_BUF_TYPE_SLICED_VBI_CAPTURE     = 6,
    V4L2_BUF_TYPE_SLICED_VBI_OUTPUT      = 7,
    V4L2_BUF_TYPE_VIDEO_OUTPUT_OVERLAY   = 8,
    V4L2_BUF_TYPE_VIDEO_CAPTURE_MPLANE   = 9,
    V4L2_BUF_TYPE_VIDEO_OUTPUT_MPLANE    = 10,
    V4L2_BUF_TYPE_SDR_CAPTURE             = 11,
    V4L2_BUF_TYPE_SDR_OUTPUT              = 12,
    V4L2_BUF_TYPE_META_CAPTURE           = 13,
    V4L2_BUF_TYPE_META_OUTPUT            = 14,
    /* Deprecated, do not use */
    V4L2_BUF_TYPE_PRIVATE                 = 0x80,
};

#define V4L2_TYPE_IS_MULTIPLANAR(type)    \
    ((type) == V4L2_BUF_TYPE_VIDEO_CAPTURE_MPLANE \
     || (type) == V4L2_BUF_TYPE_VIDEO_OUTPUT_MPLANE)

#define V4L2_TYPE_IS_OUTPUT(type)        \
    ((type) == V4L2_BUF_TYPE_VIDEO_OUTPUT \
     || (type) == V4L2_BUF_TYPE_VIDEO_OUTPUT_MPLANE \
     || (type) == V4L2_BUF_TYPE_VIDEO_OVERLAY \
     || (type) == V4L2_BUF_TYPE_VIDEO_OUTPUT_OVERLAY \
     || (type) == V4L2_BUF_TYPE_VBI_OUTPUT)

```

```
    || (type) == V4L2_BUF_TYPE_SLICED_VBI_OUTPUT          \
    || (type) == V4L2_BUF_TYPE_SDR_OUTPUT                \
    || (type) == V4L2_BUF_TYPE_META_OUTPUT)

#define V4L2_TYPE_IS_CAPTURE(type) (!V4L2_TYPE_IS_OUTPUT(type))

enum v4l2_tuner_type {
    V4L2_TUNER_RADIO          = 1,
    V4L2_TUNER_ANALOG_TV     = 2,
    V4L2_TUNER_DIGITAL_TV    = 3,
    V4L2_TUNER_SDR           = 4,
    V4L2_TUNER_RF            = 5,
};

/* Deprecated, do not use */
#define V4L2_TUNER_ADC V4L2_TUNER_SDR

enum v4l2_memory {
    V4L2_MEMORY_MMAP          = 1,
    V4L2_MEMORY_USERPTR      = 2,
    V4L2_MEMORY_OVERLAY      = 3,
    V4L2_MEMORY_DMABUF       = 4,
};

/* see also http://vektor.theorem.ca/graphics/ycbcr/ */
enum v4l2_colorspace {
    /*
     * Default colorspace, i.e. let the driver figure it out.
     * Can only be used with video capture.
     */
    V4L2_COLORSPACE_DEFAULT    = 0,

    /* SMPTE 170M: used for broadcast NTSC/PAL SDTV */
    V4L2_COLORSPACE_SMPTE170M = 1,

    /* Obsolete pre-1998 SMPTE 240M HDTV standard, superseded by Rec 709 */
    V4L2_COLORSPACE_SMPTE240M = 2,

    /* Rec.709: used for HDTV */
    V4L2_COLORSPACE_REC709    = 3,

    /*
     * Deprecated, do not use. No driver will ever return this. This was
     * based on a misunderstanding of the bt878 datasheet.
     */
    V4L2_COLORSPACE_BT878     = 4,

    /*
     * NTSC 1953 colorspace. This only makes sense when dealing with
     * really, really old NTSC recordings. Superseded by SMPTE 170M.
     */
}
```

```

V4L2_COLORSPACE_470_SYSTEM_M = 5,

/*
 * EBU Tech 3213 PAL/SECAM colorspace.
 */
V4L2_COLORSPACE_470_SYSTEM_BG = 6,

/*
 * Effectively shorthand for V4L2_COLORSPACE_SRGB, V4L2_YCBCR_ENC_601
 * and V4L2_QUANTIZATION_FULL_RANGE. To be used for (Motion-)JPEG.
 */
V4L2_COLORSPACE_JPEG = 7,

/* For RGB colorspace such as produces by most webcams. */
V4L2_COLORSPACE_SRGB = 8,

/* opRGB colorspace */
V4L2_COLORSPACE_OPRGB = 9,

/* BT.2020 colorspace, used for UHDTV. */
V4L2_COLORSPACE_BT2020 = 10,

/* Raw colorspace: for RAW unprocessed images */
V4L2_COLORSPACE_RAW = 11,

/* DCI-P3 colorspace, used by cinema projectors */
V4L2_COLORSPACE_DCI_P3 = 12,
};

/*
 * Determine how COLORSPACE_DEFAULT should map to a proper colorspace.
 * This depends on whether this is a SDTV image (use SMPTE 170M), an
 * HDTV image (use Rec. 709), or something else (use sRGB).
 */
#define V4L2_MAP_COLORSPACE_DEFAULT(is_sdtv, is_hdtv) \
    ((is_sdtv) ? V4L2_COLORSPACE_SMPTE170M : \
     ((is_hdtv) ? V4L2_COLORSPACE_REC709 : V4L2_COLORSPACE_SRGB))

enum v4l2_xfer_func {
    /*
     * Mapping of V4L2_XFER_FUNC_DEFAULT to actual transfer functions
     * for the various colorspace:
     *
     * V4L2_COLORSPACE_SMPTE170M, V4L2_COLORSPACE_470_SYSTEM_M,
     * V4L2_COLORSPACE_470_SYSTEM_BG, V4L2_COLORSPACE_REC709 and
     * V4L2_COLORSPACE_BT2020: V4L2_XFER_FUNC_709
     *
     * V4L2_COLORSPACE_SRGB, V4L2_COLORSPACE_JPEG: V4L2_XFER_FUNC_SRGB
     *
     * V4L2_COLORSPACE_OPRGB: V4L2_XFER_FUNC_OPRGB
     *
    */

```

```

* V4L2_COLORSPACE_SMPTE240M: V4L2_XFER_FUNC_SMPTE240M
*
* V4L2_COLORSPACE_RAW: V4L2_XFER_FUNC_NONE
*
* V4L2_COLORSPACE_DCI_P3: V4L2_XFER_FUNC_DCI_P3
*/
V4L2_XFER_FUNC_DEFAULT      = 0,
V4L2_XFER_FUNC_709         = 1,
V4L2_XFER_FUNC_SRGB        = 2,
V4L2_XFER_FUNC_OPRGB       = 3,
V4L2_XFER_FUNC_SMPTE240M   = 4,
V4L2_XFER_FUNC_NONE        = 5,
V4L2_XFER_FUNC_DCI_P3      = 6,
V4L2_XFER_FUNC_SMPTE2084   = 7,
};

/*
 * Determine how XFER_FUNC_DEFAULT should map to a proper transfer function.
 * This depends on the colorspace.
 */
#define V4L2_MAP_XFER_FUNC_DEFAULT(colsp) \
    ((colsp) == V4L2_COLORSPACE_OPRGB ? V4L2_XFER_FUNC_OPRGB : \
     ((colsp) == V4L2_COLORSPACE_SMPTE240M ? V4L2_XFER_FUNC_SMPTE240M : \
      ((colsp) == V4L2_COLORSPACE_DCI_P3 ? V4L2_XFER_FUNC_DCI_P3 : \
       ((colsp) == V4L2_COLORSPACE_RAW ? V4L2_XFER_FUNC_NONE : \
        (colsp) == V4L2_COLORSPACE_SRGB || (colsp) == V4L2_COLORSPACE_
→JPEG ? \
         V4L2_XFER_FUNC_SRGB : V4L2_XFER_FUNC_709))))

enum v4l2_ycbcr_encoding {
    /*
     * Mapping of V4L2_YCBCR_ENC_DEFAULT to actual encodings for the
     * various colorspace:
     *
     * V4L2_COLORSPACE_SMPTE170M, V4L2_COLORSPACE_470_SYSTEM_M,
     * V4L2_COLORSPACE_470_SYSTEM_BG, V4L2_COLORSPACE_SRGB,
     * V4L2_COLORSPACE_OPRGB and V4L2_COLORSPACE_JPEG: V4L2_YCBCR_ENC_601
     *
     * V4L2_COLORSPACE_REC709 and V4L2_COLORSPACE_DCI_P3: V4L2_YCBCR_ENC_
→709
     *
     * V4L2_COLORSPACE_BT2020: V4L2_YCBCR_ENC_BT2020
     *
     * V4L2_COLORSPACE_SMPTE240M: V4L2_YCBCR_ENC_SMPTE240M
     */
    V4L2_YCBCR_ENC_DEFAULT      = 0,

    /* ITU-R 601 -- SDTV */
    V4L2_YCBCR_ENC_601         = 1,

    /* Rec. 709 -- HDTV */

```

```

V4L2_YCBCR_ENC_709                = 2,

/* ITU-R 601/EN 61966-2-4 Extended Gamut -- SDTV */
V4L2_YCBCR_ENC_XV601              = 3,

/* Rec. 709/EN 61966-2-4 Extended Gamut -- HDTV */
V4L2_YCBCR_ENC_XV709              = 4,

#ifndef __KERNEL__
/*
 * sYCC (Y'CbCr encoding of sRGB), identical to ENC_601. It was added
 * originally due to a misunderstanding of the sYCC standard. It should
 * not be used, instead use V4L2_YCBCR_ENC_601.
 */
V4L2_YCBCR_ENC_SYCC                = 5,
#endif

/* BT.2020 Non-constant Luminance Y'CbCr */
V4L2_YCBCR_ENC_BT2020              = 6,

/* BT.2020 Constant Luminance Y'CbCr */
V4L2_YCBCR_ENC_BT2020_CONST_LUM   = 7,

/* SMPTE 240M -- Obsolete HDTV */
V4L2_YCBCR_ENC_SMPTE240M          = 8,
};

/*
 * enum v4l2_hsv_encoding values should not collide with the ones from
 * enum v4l2_ycbcr_encoding.
 */
enum v4l2_hsv_encoding {

    /* Hue mapped to 0 - 179 */
    V4L2_HSV_ENC_180                = 128,

    /* Hue mapped to 0-255 */
    V4L2_HSV_ENC_256                = 129,
};

/*
 * Determine how YCBCR_ENC_DEFAULT should map to a proper Y'CbCr encoding.
 * This depends on the colorspace.
 */
#define V4L2_MAP_YCBCR_ENC_DEFAULT(colsp) \
    (((colsp) == V4L2_COLORSPACE_REC709 || \
     (colsp) == V4L2_COLORSPACE_DCI_P3) ? V4L2_YCBCR_ENC_709 : \
     ((colsp) == V4L2_COLORSPACE_BT2020 ? V4L2_YCBCR_ENC_BT2020 : \
     ((colsp) == V4L2_COLORSPACE_SMPTE240M ? V4L2_YCBCR_ENC_SMPTE240M : \
     V4L2_YCBCR_ENC_601)))

```

```
enum v4l2_quantization {
    /*
     * The default for R'G'B' quantization is always full range.
     * For Y'CbCr the quantization is always limited range, except
     * for COLORSPACE_JPEG: this is full range.
     */
    V4L2_QUANTIZATION_DEFAULT      = 0,
    V4L2_QUANTIZATION_FULL_RANGE  = 1,
    V4L2_QUANTIZATION_LIM_RANGE   = 2,
};

/*
 * Determine how QUANTIZATION_DEFAULT should map to a proper quantization.
 * This depends on whether the image is RGB or not, the colorspace.
 * The Y'CbCr encoding is not used anymore, but is still there for backwards
 * compatibility.
 */
#define V4L2_MAP_QUANTIZATION_DEFAULT(is_rgb_or_hsv, colsp, ycbcr_enc) \
    (((is_rgb_or_hsv) || (colsp) == V4L2_COLORSPACE_JPEG) ? \
     V4L2_QUANTIZATION_FULL_RANGE : V4L2_QUANTIZATION_LIM_RANGE)

/*
 * Deprecated names for opRGB colorspace (IEC 61966-2-5)
 *
 * WARNING: Please don't use these deprecated defines in your code, as
 * there is a chance we have to remove them in the future.
 */
#ifdef __KERNEL__
#define V4L2_COLORSPACE_ADOBERGB V4L2_COLORSPACE_OPRGB
#define V4L2_XFER_FUNC_ADOBERGB V4L2_XFER_FUNC_OPRGB
#endif

enum v4l2_priority {
    V4L2_PRIORITY_UNSET          = 0, /* not initialized */
    V4L2_PRIORITY_BACKGROUND    = 1,
    V4L2_PRIORITY_INTERACTIVE   = 2,
    V4L2_PRIORITY_RECORD        = 3,
    V4L2_PRIORITY_DEFAULT       = V4L2_PRIORITY_INTERACTIVE,
};

struct v4l2_rect {
    __s32  left;
    __s32  top;
    __u32  width;
    __u32  height;
};

struct v4l2_fract {
    __u32  numerator;
    __u32  denominator;
};
```



```

struct v4l2_area {
    __u32    width;
    __u32    height;
};

/**
 * struct v4l2_capability - Describes V4L2 device caps returned by VIDIOC_
 * → QUERYCAP
 *
 * @driver:      name of the driver module (e.g. "bttv")
 * @card:       name of the card (e.g. "Hauppauge WinTV")
 * @bus_info:   name of the bus (e.g. "PCI:" + pci_name(pci_dev) )
 * @version:    KERNEL_VERSION
 * @capabilities: capabilities of the physical device as a whole
 * @device_caps: capabilities accessed via this particular device (node)
 * @reserved:   reserved fields for future extensions
 */
struct v4l2_capability {
    __u8    driver[16];
    __u8    card[32];
    __u8    bus_info[32];
    __u32    version;
    __u32    capabilities;
    __u32    device_caps;
    __u32    reserved[3];
};

/* Values for 'capabilities' field */
#define V4L2_CAP_VIDEO_CAPTURE    0x00000001 /* Is a video capture_
 * → device */
#define V4L2_CAP_VIDEO_OUTPUT    0x00000002 /* Is a video output_
 * → device */
#define V4L2_CAP_VIDEO_OVERLAY    0x00000004 /* Can do video overlay */
#define V4L2_CAP_VBI_CAPTURE    0x00000010 /* Is a raw VBI capture_
 * → device */
#define V4L2_CAP_VBI_OUTPUT    0x00000020 /* Is a raw VBI output_
 * → device */
#define V4L2_CAP_SLICED_VBI_CAPTURE    0x00000040 /* Is a sliced VBI capture_
 * → device */
#define V4L2_CAP_SLICED_VBI_OUTPUT    0x00000080 /* Is a sliced VBI output_
 * → device */
#define V4L2_CAP_RDS_CAPTURE    0x00000100 /* RDS data capture */
#define V4L2_CAP_VIDEO_OUTPUT_OVERLAY    0x00000200 /* Can do video output_
 * → overlay */
#define V4L2_CAP_HW_FREQ_SEEK    0x00000400 /* Can do hardware_
 * → frequency seek */
#define V4L2_CAP_RDS_OUTPUT    0x00000800 /* Is an RDS encoder */

/* Is a video capture device that supports multiplanar formats */
#define V4L2_CAP_VIDEO_CAPTURE_MPLANE    0x00001000

```

```

/* Is a video output device that supports multiplanar formats */
#define V4L2_CAP_VIDEO_OUTPUT_MPLANE    0x00002000
/* Is a video mem-to-mem device that supports multiplanar formats */
#define V4L2_CAP_VIDEO_M2M_MPLANE      0x00004000
/* Is a video mem-to-mem device */
#define V4L2_CAP_VIDEO_M2M              0x00008000

#define V4L2_CAP_TUNER                  0x00010000 /* has a tuner */
#define V4L2_CAP_AUDIO                  0x00020000 /* has audio support */
#define V4L2_CAP_RADIO                  0x00040000 /* is a radio device */
#define V4L2_CAP_MODULATOR             0x00080000 /* has a modulator */

#define V4L2_CAP_SDR_CAPTURE             0x00100000 /* Is a SDR capture device,
↳ */
#define V4L2_CAP_EXT_PIX_FORMAT         0x00200000 /* Supports the extended,
↳ pixel format */
#define V4L2_CAP_SDR_OUTPUT             0x00400000 /* Is a SDR output device,
↳ */
#define V4L2_CAP_META_CAPTURE           0x00800000 /* Is a metadata capture,
↳ device */

#define V4L2_CAP_READWRITE              0x01000000 /* read/write systemcalls,
↳ */
#define V4L2_CAP_ASYNCIO                0x02000000 /* async I/O */
#define V4L2_CAP_STREAMING              0x04000000 /* streaming I/O ioctls */
#define V4L2_CAP_META_OUTPUT            0x08000000 /* Is a metadata output,
↳ device */

#define V4L2_CAP_TOUCH                  0x10000000 /* Is a touch device */

#define V4L2_CAP_IO_MC                  0x20000000 /* Is input/output,
↳ controlled by the media controller */

#define V4L2_CAP_DEVICE_CAPS            0x80000000 /* sets device,
↳ capabilities field */

/*
 *   V I D E O   I M A G E   F O R M A T
 */
struct v4l2_pix_format {
    __u32          width;
    __u32          height;
    __u32          pixelformat;
    __u32          field; /* enum v4l2_field */
    __u32          bytesperline; /* for padding, zero if unused,
↳ */
    __u32          sizeimage;
    __u32          colorspace; /* enum v4l2_colorspace */
    __u32          priv; /* private data, depends on,
↳ pixelformat */
    __u32          flags; /* format flags (V4L2_PIX_FMT_

```

```

→FLAG_*) */
    union {
        /* enum v4l2_ycbcr_encoding */
        __u32          ycbcr_enc;
        /* enum v4l2_hsv_encoding */
        __u32          hsv_enc;
    };
    __u32             quantization; /* enum v4l2_quantization */
    __u32             xfer_func;    /* enum v4l2_xfer_func */
};

/*      Pixel format      FOURCC      depth
→Description */

/* RGB formats (1 or 2 bytes per pixel) */
#define V4L2_PIX_FMT_RGB332 v4l2_fourcc('R', 'G', 'B', '1') /* 8  RGB-3-3-2
→ */
#define V4L2_PIX_FMT_RGB444 v4l2_fourcc('R', '4', '4', '4') /* 16  xxxrrrr
→gggbbbb */
#define V4L2_PIX_FMT_ARGB444 v4l2_fourcc('A', 'R', '1', '2') /* 16  aaaarrrr
→gggbbbb */
#define V4L2_PIX_FMT_XRGB444 v4l2_fourcc('X', 'R', '1', '2') /* 16  xxxrrrr
→gggbbbb */
#define V4L2_PIX_FMT_RGBA444 v4l2_fourcc('R', 'A', '1', '2') /* 16  rrrrgggg
→bbbbaaaa */
#define V4L2_PIX_FMT_RGBX444 v4l2_fourcc('R', 'X', '1', '2') /* 16  rrrrgggg
→bbbxxxx */
#define V4L2_PIX_FMT_ABGR444 v4l2_fourcc('A', 'B', '1', '2') /* 16  aaaabbbb
→gggrrrr */
#define V4L2_PIX_FMT_XBGR444 v4l2_fourcc('X', 'B', '1', '2') /* 16  xxxxbbbb
→gggrrrr */
#define V4L2_PIX_FMT_BGRA444 v4l2_fourcc('G', 'A', '1', '2') /* 16  bbbbgggg
→rrrraaaa */
#define V4L2_PIX_FMT_BGRX444 v4l2_fourcc('B', 'X', '1', '2') /* 16  bbbbgggg
→rrrrxxxx */
#define V4L2_PIX_FMT_RGB555 v4l2_fourcc('R', 'G', 'B', '0') /* 16  RGB-5-5-5
→ */
#define V4L2_PIX_FMT_ARGB555 v4l2_fourcc('A', 'R', '1', '5') /* 16
→ARGB-1-5-5-5 */
#define V4L2_PIX_FMT_XRGB555 v4l2_fourcc('X', 'R', '1', '5') /* 16
→XRGB-1-5-5-5 */
#define V4L2_PIX_FMT_RGBA555 v4l2_fourcc('R', 'A', '1', '5') /* 16
→RGBA-5-5-5-1 */
#define V4L2_PIX_FMT_RGBX555 v4l2_fourcc('R', 'X', '1', '5') /* 16
→RGBX-5-5-5-1 */
#define V4L2_PIX_FMT_ABGR555 v4l2_fourcc('A', 'B', '1', '5') /* 16
→ABGR-1-5-5-5 */
#define V4L2_PIX_FMT_XBGR555 v4l2_fourcc('X', 'B', '1', '5') /* 16
→XBGR-1-5-5-5 */
#define V4L2_PIX_FMT_BGRA555 v4l2_fourcc('B', 'A', '1', '5') /* 16
→BGRA-5-5-5-1 */

```

```

#define V4L2_PIX_FMT_BGRX555 v4l2_fourcc('B', 'X', '1', '5') /* 16
↳BGRX-5-5-5-1 */
#define V4L2_PIX_FMT_RGB565 v4l2_fourcc('R', 'G', 'B', 'P') /* 16 RGB-5-6-5
↳ */
#define V4L2_PIX_FMT_RGB555X v4l2_fourcc('R', 'G', 'B', 'Q') /* 16 RGB-5-5-5
↳BE */
#define V4L2_PIX_FMT_ARGB555X v4l2_fourcc_be('A', 'R', '1', '5') /* 16
↳ARGB-5-5-5 BE */
#define V4L2_PIX_FMT_XRGB555X v4l2_fourcc_be('X', 'R', '1', '5') /* 16
↳XRGB-5-5-5 BE */
#define V4L2_PIX_FMT_RGB565X v4l2_fourcc('R', 'G', 'B', 'R') /* 16 RGB-5-6-5
↳BE */

/* RGB formats (3 or 4 bytes per pixel) */
#define V4L2_PIX_FMT_BGR666 v4l2_fourcc('B', 'G', 'R', 'H') /* 18 BGR-6-6-6
↳ */
#define V4L2_PIX_FMT_BGR24 v4l2_fourcc('B', 'G', 'R', '3') /* 24 BGR-8-8-8
↳ */
#define V4L2_PIX_FMT_RGB24 v4l2_fourcc('R', 'G', 'B', '3') /* 24 RGB-8-8-8
↳ */
#define V4L2_PIX_FMT_BGR32 v4l2_fourcc('B', 'G', 'R', '4') /* 32
↳BGR-8-8-8-8 */
#define V4L2_PIX_FMT_ABGR32 v4l2_fourcc('A', 'R', '2', '4') /* 32
↳BGRA-8-8-8-8 */
#define V4L2_PIX_FMT_XBGR32 v4l2_fourcc('X', 'R', '2', '4') /* 32
↳BGRX-8-8-8-8 */
#define V4L2_PIX_FMT_BGRA32 v4l2_fourcc('R', 'A', '2', '4') /* 32
↳ABGR-8-8-8-8 */
#define V4L2_PIX_FMT_BGRX32 v4l2_fourcc('R', 'X', '2', '4') /* 32
↳XBGR-8-8-8-8 */
#define V4L2_PIX_FMT_RGB32 v4l2_fourcc('R', 'G', 'B', '4') /* 32
↳RGB-8-8-8-8 */
#define V4L2_PIX_FMT_RGBA32 v4l2_fourcc('A', 'B', '2', '4') /* 32
↳RGBA-8-8-8-8 */
#define V4L2_PIX_FMT_RGBX32 v4l2_fourcc('X', 'B', '2', '4') /* 32
↳RGBX-8-8-8-8 */
#define V4L2_PIX_FMT_ARGB32 v4l2_fourcc('B', 'A', '2', '4') /* 32
↳ARGB-8-8-8-8 */
#define V4L2_PIX_FMT_XRGB32 v4l2_fourcc('B', 'X', '2', '4') /* 32
↳XRGB-8-8-8-8 */

/* Grey formats */
#define V4L2_PIX_FMT_GREY v4l2_fourcc('G', 'R', 'E', 'Y') /* 8 Greyscale
↳ */
#define V4L2_PIX_FMT_Y4 v4l2_fourcc('Y', '0', '4', ' ') /* 4 Greyscale
↳ */
#define V4L2_PIX_FMT_Y6 v4l2_fourcc('Y', '0', '6', ' ') /* 6 Greyscale
↳ */
#define V4L2_PIX_FMT_Y10 v4l2_fourcc('Y', '1', '0', ' ') /* 10 Greyscale
↳ */
#define V4L2_PIX_FMT_Y12 v4l2_fourcc('Y', '1', '2', ' ') /* 12 Greyscale
↳

```

```

→ */
#define V4L2_PIX_FMT_Y14      v4l2_fourcc('Y', '1', '4', ' ') /* 14 Greyscale  ␣
→ */
#define V4L2_PIX_FMT_Y16      v4l2_fourcc('Y', '1', '6', ' ') /* 16 Greyscale  ␣
→ */
#define V4L2_PIX_FMT_Y16_BE  v4l2_fourcc_be('Y', '1', '6', ' ') /* 16
→Greyscale BE */

/* Grey bit-packed formats */
#define V4L2_PIX_FMT_Y10BPACK v4l2_fourcc('Y', '1', '0', 'B') /* 10  ␣
→Greyscale bit-packed */
#define V4L2_PIX_FMT_Y10P    v4l2_fourcc('Y', '1', '0', 'P') /* 10 Greyscale, ␣
→MIPI RAW10 packed */

/* Palette formats */
#define V4L2_PIX_FMT_PAL8    v4l2_fourcc('P', 'A', 'L', '8') /* 8 8-bit ␣
→palette */

/* Chrominance formats */
#define V4L2_PIX_FMT_UV8     v4l2_fourcc('U', 'V', '8', ' ') /* 8 UV 4:4 */

/* Luminance+Chrominance formats */
#define V4L2_PIX_FMT_YUYV    v4l2_fourcc('Y', 'U', 'Y', 'V') /* 16 YUV 4:2:2  ␣
→ */
#define V4L2_PIX_FMT_YUUV    v4l2_fourcc('Y', 'U', 'U', 'V') /* 16 YUV 4:2:2  ␣
→ */
#define V4L2_PIX_FMT_YVYU    v4l2_fourcc('Y', 'V', 'Y', 'U') /* 16 YVU 4:2:2 */
#define V4L2_PIX_FMT_UYVY    v4l2_fourcc('U', 'Y', 'V', 'Y') /* 16 YUV 4:2:2  ␣
→ */
#define V4L2_PIX_FMT_VYUY    v4l2_fourcc('V', 'Y', 'U', 'Y') /* 16 YUV 4:2:2  ␣
→ */
#define V4L2_PIX_FMT_Y41P    v4l2_fourcc('Y', '4', '1', 'P') /* 12 YUV 4:1:1  ␣
→ */
#define V4L2_PIX_FMT_YUV444  v4l2_fourcc('Y', '4', '4', '4') /* 16 xxxxyyyy ␣
→uuuuvvvv */
#define V4L2_PIX_FMT_YUV555  v4l2_fourcc('Y', 'U', 'V', '0') /* 16 YUV-5-5-5  ␣
→ */
#define V4L2_PIX_FMT_YUV565  v4l2_fourcc('Y', 'U', 'V', 'P') /* 16 YUV-5-6-5  ␣
→ */
#define V4L2_PIX_FMT_YUV24   v4l2_fourcc('Y', 'U', 'V', '3') /* 24 YUV-8-8-8  ␣
→ */
#define V4L2_PIX_FMT_YUV32   v4l2_fourcc('Y', 'U', 'V', '4') /* 32  ␣
→YUV-8-8-8-8 */
#define V4L2_PIX_FMT_AYUV32  v4l2_fourcc('A', 'Y', 'U', 'V') /* 32  ␣
→AYUV-8-8-8-8 */
#define V4L2_PIX_FMT_XYUV32  v4l2_fourcc('X', 'Y', 'U', 'V') /* 32  ␣
→XYUV-8-8-8-8 */
#define V4L2_PIX_FMT_VUYA32  v4l2_fourcc('V', 'U', 'Y', 'A') /* 32  ␣
→VUYA-8-8-8-8 */
#define V4L2_PIX_FMT_VUYX32  v4l2_fourcc('V', 'U', 'Y', 'X') /* 32  ␣
→VUYX-8-8-8-8 */

```

```

#define V4L2_PIX_FMT_M420    v4l2_fourcc('M', '4', '2', '0') /* 12  YUV 4:2:0
↳2 lines y, 1 line uv interleaved */

/* two planes -- one Y, one Cr + Cb interleaved */
#define V4L2_PIX_FMT_NV12    v4l2_fourcc('N', 'V', '1', '2') /* 12  Y/CbCr
↳4:2:0 */
#define V4L2_PIX_FMT_NV21    v4l2_fourcc('N', 'V', '2', '1') /* 12  Y/CrCb
↳4:2:0 */
#define V4L2_PIX_FMT_NV16    v4l2_fourcc('N', 'V', '1', '6') /* 16  Y/CbCr
↳4:2:2 */
#define V4L2_PIX_FMT_NV61    v4l2_fourcc('N', 'V', '6', '1') /* 16  Y/CrCb
↳4:2:2 */
#define V4L2_PIX_FMT_NV24    v4l2_fourcc('N', 'V', '2', '4') /* 24  Y/CbCr
↳4:4:4 */
#define V4L2_PIX_FMT_NV42    v4l2_fourcc('N', 'V', '4', '2') /* 24  Y/CrCb
↳4:4:4 */

/* two non contiguous planes - one Y, one Cr + Cb interleaved */
#define V4L2_PIX_FMT_NV12M    v4l2_fourcc('N', 'M', '1', '2') /* 12  Y/CbCr
↳4:2:0 */
#define V4L2_PIX_FMT_NV21M    v4l2_fourcc('N', 'M', '2', '1') /* 21  Y/CrCb
↳4:2:0 */
#define V4L2_PIX_FMT_NV16M    v4l2_fourcc('N', 'M', '1', '6') /* 16  Y/CbCr
↳4:2:2 */
#define V4L2_PIX_FMT_NV61M    v4l2_fourcc('N', 'M', '6', '1') /* 16  Y/CrCb
↳4:2:2 */

/* three planes - Y Cb, Cr */
#define V4L2_PIX_FMT_YUV410    v4l2_fourcc('Y', 'U', 'V', '9') /* 9  YUV 4:1:0
↳ */
#define V4L2_PIX_FMT_YVU410    v4l2_fourcc('Y', 'V', 'U', '9') /* 9  YVU 4:1:0
↳ */
#define V4L2_PIX_FMT_YUV411P    v4l2_fourcc('4', '1', '1', 'P') /* 12  YVU411
↳planar */
#define V4L2_PIX_FMT_YUV420    v4l2_fourcc('Y', 'U', '1', '2') /* 12  YUV 4:2:0
↳ */
#define V4L2_PIX_FMT_YVU420    v4l2_fourcc('Y', 'V', '1', '2') /* 12  YVU 4:2:0
↳ */
#define V4L2_PIX_FMT_YUV422P    v4l2_fourcc('4', '2', '2', 'P') /* 16  YVU422
↳planar */

/* three non contiguous planes - Y, Cb, Cr */
#define V4L2_PIX_FMT_YUV420M    v4l2_fourcc('Y', 'M', '1', '2') /* 12  YUV420
↳planar */
#define V4L2_PIX_FMT_YVU420M    v4l2_fourcc('Y', 'M', '2', '1') /* 12  YVU420
↳planar */
#define V4L2_PIX_FMT_YUV422M    v4l2_fourcc('Y', 'M', '1', '6') /* 16  YUV422
↳planar */
#define V4L2_PIX_FMT_YVU422M    v4l2_fourcc('Y', 'M', '6', '1') /* 16  YVU422
↳planar */
#define V4L2_PIX_FMT_YUV444M    v4l2_fourcc('Y', 'M', '2', '4') /* 24  YUV444
↳planar */

```



```

↳ planar */
#define V4L2_PIX_FMT_YVU444M v4l2_fourcc('Y', 'M', '4', '2') /* 24 YVU444
↳ planar */

/* Tiled YUV formats */
#define V4L2_PIX_FMT_NV12_4L4 v4l2_fourcc('V', 'T', '1', '2') /* 12 Y/CbCr
↳4:2:0 4x4 tiles */
#define V4L2_PIX_FMT_NV12_16L16 v4l2_fourcc('H', 'M', '1', '2') /* 12 Y/CbCr
↳4:2:0 16x16 tiles */
#define V4L2_PIX_FMT_NV12_32L32 v4l2_fourcc('S', 'T', '1', '2') /* 12 Y/CbCr
↳4:2:0 32x32 tiles */

/* Tiled YUV formats, non contiguous planes */
#define V4L2_PIX_FMT_NV12MT v4l2_fourcc('T', 'M', '1', '2') /* 12 Y/CbCr
↳4:2:0 64x32 tiles */
#define V4L2_PIX_FMT_NV12MT_16X16 v4l2_fourcc('V', 'M', '1', '2') /* 12 Y/
↳CbCr 4:2:0 16x16 tiles */

/* Bayer formats - see http://www.siliconimaging.com/RGB%20Bayer.htm */
#define V4L2_PIX_FMT_SBGGR8 v4l2_fourcc('B', 'A', '8', '1') /* 8 BGBG..
↳GRGR.. */
#define V4L2_PIX_FMT_SGBRG8 v4l2_fourcc('G', 'B', 'R', 'G') /* 8 GBGB..
↳RGRG.. */
#define V4L2_PIX_FMT_SGRBG8 v4l2_fourcc('G', 'R', 'B', 'G') /* 8 GRGR..
↳BGBG.. */
#define V4L2_PIX_FMT_SRGGB8 v4l2_fourcc('R', 'G', 'G', 'B') /* 8 RGRG..
↳GBGB.. */
#define V4L2_PIX_FMT_SBGGR10 v4l2_fourcc('B', 'G', '1', '0') /* 10 BGBG..
↳GRGR.. */
#define V4L2_PIX_FMT_SGBRG10 v4l2_fourcc('G', 'B', '1', '0') /* 10 GBGB..
↳RGRG.. */
#define V4L2_PIX_FMT_SGRBG10 v4l2_fourcc('B', 'A', '1', '0') /* 10 GRGR..
↳BGBG.. */
#define V4L2_PIX_FMT_SRGGB10 v4l2_fourcc('R', 'G', '1', '0') /* 10 RGRG..
↳GBGB.. */
/* 10bit raw bayer packed, 5 bytes for every 4 pixels */
#define V4L2_PIX_FMT_SBGGR10P v4l2_fourcc('p', 'B', 'A', 'A')
#define V4L2_PIX_FMT_SGBRG10P v4l2_fourcc('p', 'G', 'A', 'A')
#define V4L2_PIX_FMT_SGRBG10P v4l2_fourcc('p', 'g', 'A', 'A')
#define V4L2_PIX_FMT_SRGGB10P v4l2_fourcc('p', 'R', 'A', 'A')
/* 10bit raw bayer a-law compressed to 8 bits */
#define V4L2_PIX_FMT_SBGGR10ALAW8 v4l2_fourcc('a', 'B', 'A', '8')
#define V4L2_PIX_FMT_SGBRG10ALAW8 v4l2_fourcc('a', 'G', 'A', '8')
#define V4L2_PIX_FMT_SGRBG10ALAW8 v4l2_fourcc('a', 'g', 'A', '8')
#define V4L2_PIX_FMT_SRGGB10ALAW8 v4l2_fourcc('a', 'R', 'A', '8')
/* 10bit raw bayer DPCM compressed to 8 bits */
#define V4L2_PIX_FMT_SBGGR10DPCM8 v4l2_fourcc('b', 'B', 'A', '8')
#define V4L2_PIX_FMT_SGBRG10DPCM8 v4l2_fourcc('b', 'G', 'A', '8')
#define V4L2_PIX_FMT_SGRBG10DPCM8 v4l2_fourcc('B', 'D', '1', '0')
#define V4L2_PIX_FMT_SRGGB10DPCM8 v4l2_fourcc('b', 'R', 'A', '8')
#define V4L2_PIX_FMT_SBGGR12 v4l2_fourcc('B', 'G', '1', '2') /* 12 BGBG..

```

```

↳GRGR.. */
#define V4L2_PIX_FMT_SGBRG12 v4l2_fourcc('G', 'B', '1', '2') /* 12  GBGB..
↳RGRG.. */
#define V4L2_PIX_FMT_SGRBG12 v4l2_fourcc('B', 'A', '1', '2') /* 12  GRGR..
↳BGBG.. */
#define V4L2_PIX_FMT_SRGGB12 v4l2_fourcc('R', 'G', '1', '2') /* 12  RGRG..
↳GBGB.. */
/* 12bit raw bayer packed, 6 bytes for every 4 pixels */
#define V4L2_PIX_FMT_SBGGR12P v4l2_fourcc('p', 'B', 'C', 'C')
#define V4L2_PIX_FMT_SGBRG12P v4l2_fourcc('p', 'G', 'C', 'C')
#define V4L2_PIX_FMT_SGRBG12P v4l2_fourcc('p', 'g', 'C', 'C')
#define V4L2_PIX_FMT_SRGGB12P v4l2_fourcc('p', 'R', 'C', 'C')
#define V4L2_PIX_FMT_SBGGR14 v4l2_fourcc('B', 'G', '1', '4') /* 14  BGBG..
↳GRGR.. */
#define V4L2_PIX_FMT_SGBRG14 v4l2_fourcc('G', 'B', '1', '4') /* 14  GBGB..
↳RGRG.. */
#define V4L2_PIX_FMT_SGRBG14 v4l2_fourcc('G', 'R', '1', '4') /* 14  GRGR..
↳BGBG.. */
#define V4L2_PIX_FMT_SRGGB14 v4l2_fourcc('R', 'G', '1', '4') /* 14  RGRG..
↳GBGB.. */
/* 14bit raw bayer packed, 7 bytes for every 4 pixels */
#define V4L2_PIX_FMT_SBGGR14P v4l2_fourcc('p', 'B', 'E', 'E')
#define V4L2_PIX_FMT_SGBRG14P v4l2_fourcc('p', 'G', 'E', 'E')
#define V4L2_PIX_FMT_SGRBG14P v4l2_fourcc('p', 'g', 'E', 'E')
#define V4L2_PIX_FMT_SRGGB14P v4l2_fourcc('p', 'R', 'E', 'E')
#define V4L2_PIX_FMT_SBGGR16 v4l2_fourcc('B', 'Y', 'R', '2') /* 16  BGBG..
↳GRGR.. */
#define V4L2_PIX_FMT_SGBRG16 v4l2_fourcc('G', 'B', '1', '6') /* 16  GBGB..
↳RGRG.. */
#define V4L2_PIX_FMT_SGRBG16 v4l2_fourcc('G', 'R', '1', '6') /* 16  GRGR..
↳BGBG.. */
#define V4L2_PIX_FMT_SRGGB16 v4l2_fourcc('R', 'G', '1', '6') /* 16  RGRG..
↳GBGB.. */

/* HSV formats */
#define V4L2_PIX_FMT_HSV24 v4l2_fourcc('H', 'S', 'V', '3')
#define V4L2_PIX_FMT_HSV32 v4l2_fourcc('H', 'S', 'V', '4')

/* compressed formats */
#define V4L2_PIX_FMT_MJPEG v4l2_fourcc('M', 'J', 'P', 'G') /* Motion-JPEG
↳*/
#define V4L2_PIX_FMT_JPEG v4l2_fourcc('J', 'P', 'E', 'G') /* JFIF JPEG
↳*/
#define V4L2_PIX_FMT_DV v4l2_fourcc('d', 'v', 's', 'd') /* 1394
↳*/
#define V4L2_PIX_FMT_MPEG v4l2_fourcc('M', 'P', 'E', 'G') /* MPEG-1/2/4
↳Multiplexed */
#define V4L2_PIX_FMT_H264 v4l2_fourcc('H', '2', '6', '4') /* H264 with
↳start codes */
#define V4L2_PIX_FMT_H264_NO_SC v4l2_fourcc('A', 'V', 'C', '1') /* H264
↳without start codes */

```



```

#define V4L2_PIX_FMT_H264_MVC v4l2_fourcc('M', '2', '6', '4') /* H264 MVC */
#define V4L2_PIX_FMT_H263     v4l2_fourcc('H', '2', '6', '3') /* H263
↳ */
#define V4L2_PIX_FMT_MPEG1    v4l2_fourcc('M', 'P', 'G', '1') /* MPEG-1 ES
↳ */
#define V4L2_PIX_FMT_MPEG2    v4l2_fourcc('M', 'P', 'G', '2') /* MPEG-2 ES
↳ */
#define V4L2_PIX_FMT_MPEG2_SLICE v4l2_fourcc('M', 'G', '2', 'S') /* MPEG-2
↳ parsed slice data */
#define V4L2_PIX_FMT_MPEG4     v4l2_fourcc('M', 'P', 'G', '4') /* MPEG-4 part 2
↳ ES */
#define V4L2_PIX_FMT_XVID      v4l2_fourcc('X', 'V', 'I', 'D') /* Xvid
↳ */
#define V4L2_PIX_FMT_VC1_ANNEX_G v4l2_fourcc('V', 'C', '1', 'G') /* SMPTE 421M
↳ Annex G compliant stream */
#define V4L2_PIX_FMT_VC1_ANNEX_L v4l2_fourcc('V', 'C', '1', 'L') /* SMPTE 421M
↳ Annex L compliant stream */
#define V4L2_PIX_FMT_VP8       v4l2_fourcc('V', 'P', '8', '0') /* VP8 */
#define V4L2_PIX_FMT_VP8_FRAME v4l2_fourcc('V', 'P', '8', 'F') /* VP8 parsed
↳ frame */
#define V4L2_PIX_FMT_VP9       v4l2_fourcc('V', 'P', '9', '0') /* VP9 */
#define V4L2_PIX_FMT_VP9_FRAME v4l2_fourcc('V', 'P', '9', 'F') /* VP9 parsed
↳ frame */
#define V4L2_PIX_FMT_HEVC      v4l2_fourcc('H', 'E', 'V', 'C') /* HEVC aka H.
↳ 265 */
#define V4L2_PIX_FMT_FWHT      v4l2_fourcc('F', 'W', 'H', 'T') /* Fast Walsh
↳ Hadamard Transform (vicodec) */
#define V4L2_PIX_FMT_FWHT_STATELESS v4l2_fourcc('S', 'F', 'W', 'H') /*
↳ Stateless FWHT (vicodec) */
#define V4L2_PIX_FMT_H264_SLICE v4l2_fourcc('S', '2', '6', '4') /* H264 parsed
↳ slices */

/* Vendor-specific formats */
#define V4L2_PIX_FMT_CPIA1     v4l2_fourcc('C', 'P', 'I', 'A') /* cpia1 YUV */
#define V4L2_PIX_FMT_WNVA      v4l2_fourcc('W', 'N', 'V', 'A') /* Winnov hw
↳ compress */
#define V4L2_PIX_FMT_SN9C10X    v4l2_fourcc('S', '9', '1', '0') /* SN9C10x
↳ compression */
#define V4L2_PIX_FMT_SN9C20X_I420 v4l2_fourcc('S', '9', '2', '0') /* SN9C20x
↳ YUV 4:2:0 */
#define V4L2_PIX_FMT_PWC1      v4l2_fourcc('P', 'W', 'C', '1') /* pwc older
↳ webcam */
#define V4L2_PIX_FMT_PWC2      v4l2_fourcc('P', 'W', 'C', '2') /* pwc newer
↳ webcam */
#define V4L2_PIX_FMT_ET61X251  v4l2_fourcc('E', '6', '2', '5') /* ET61X251
↳ compression */
#define V4L2_PIX_FMT_SPCA501    v4l2_fourcc('S', '5', '0', '1') /* YUYV per line
↳ */
#define V4L2_PIX_FMT_SPCA505    v4l2_fourcc('S', '5', '0', '5') /* YYUV per line
↳ */
#define V4L2_PIX_FMT_SPCA508    v4l2_fourcc('S', '5', '0', '8') /* YUVY per line
↳

```

```

↳ */
#define V4L2_PIX_FMT_SPCA561 v4l2_fourcc('S', '5', '6', '1') /* compressed
↳GBRG bayer */
#define V4L2_PIX_FMT_PAC207 v4l2_fourcc('P', '2', '0', '7') /* compressed
↳BGGR bayer */
#define V4L2_PIX_FMT_MR97310A v4l2_fourcc('M', '3', '1', '0') /* compressed
↳BGGR bayer */
#define V4L2_PIX_FMT_JL2005BCD v4l2_fourcc('J', 'L', '2', '0') /* compressed
↳RGGG bayer */
#define V4L2_PIX_FMT_SN9C2028 v4l2_fourcc('S', '0', 'N', 'X') /* compressed
↳GBRG bayer */
#define V4L2_PIX_FMT_S0905C v4l2_fourcc('9', '0', '5', 'C') /* compressed
↳RGGG bayer */
#define V4L2_PIX_FMT_PJPG v4l2_fourcc('P', 'J', 'P', 'G') /* Pixart 73xx
↳JPEG */
#define V4L2_PIX_FMT_OV511 v4l2_fourcc('0', '5', '1', '1') /* ov511 JPEG */
#define V4L2_PIX_FMT_OV518 v4l2_fourcc('0', '5', '1', '8') /* ov518 JPEG */
#define V4L2_PIX_FMT_STV0680 v4l2_fourcc('S', '6', '8', '0') /* stv0680 bayer
↳ */
#define V4L2_PIX_FMT_TM6000 v4l2_fourcc('T', 'M', '6', '0') /* tm5600/tm60x0
↳ */
#define V4L2_PIX_FMT_CIT_YYVYUY v4l2_fourcc('C', 'I', 'T', 'V') /* one line of
↳Y then 1 line of VYUY */
#define V4L2_PIX_FMT_KONICA420 v4l2_fourcc('K', '0', 'N', 'I') /* YUV420
↳planar in blocks of 256 pixels */
#define V4L2_PIX_FMT_JPGL v4l2_fourcc('J', 'P', 'G', 'L') /* JPEG-Lite */
#define V4L2_PIX_FMT_SE401 v4l2_fourcc('S', '4', '0', '1') /* se401
↳janggu compressed rgb */
#define V4L2_PIX_FMT_S5C_UYVY_JPG v4l2_fourcc('S', '5', 'C', 'I') /* S5C73M3
↳interleaved UYVY/JPEG */
#define V4L2_PIX_FMT_Y8I v4l2_fourcc('Y', '8', 'I', ' ') /* Greyscale
↳8-bit L/R interleaved */
#define V4L2_PIX_FMT_Y12I v4l2_fourcc('Y', '1', '2', 'I') /* Greyscale
↳12-bit L/R interleaved */
#define V4L2_PIX_FMT_Z16 v4l2_fourcc('Z', '1', '6', ' ') /* Depth data
↳16-bit */
#define V4L2_PIX_FMT_MT21C v4l2_fourcc('M', 'T', '2', '1') /* Mediatek
↳compressed block mode */
#define V4L2_PIX_FMT_MM21 v4l2_fourcc('M', 'M', '2', '1') /* Mediatek
↳8-bit block mode, two non-contiguous planes */
#define V4L2_PIX_FMT_INZI v4l2_fourcc('I', 'N', 'Z', 'I') /* Intel Planar
↳Greyscale 10-bit and Depth 16-bit */
#define V4L2_PIX_FMT_CNF4 v4l2_fourcc('C', 'N', 'F', '4') /* Intel 4-bit
↳packed depth confidence information */
#define V4L2_PIX_FMT_HI240 v4l2_fourcc('H', 'I', '2', '4') /* BTTV 8-bit
↳dithered RGB */

/* 10bit raw bayer packed, 32 bytes for every 25 pixels, last LSB 6 bits
↳unused */
#define V4L2_PIX_FMT_IPU3_SBGGR10 v4l2_fourcc('i', 'p', '3', 'b') /*
↳IPU3 packed 10-bit BGGR bayer */

```

```

#define V4L2_PIX_FMT_IPU3_SGBRG10    v4l2_fourcc('i', 'p', '3', 'g') /*
↳ IPU3 packed 10-bit GBRG bayer */
#define V4L2_PIX_FMT_IPU3_SGRBG10    v4l2_fourcc('i', 'p', '3', 'G') /*
↳ IPU3 packed 10-bit GRBG bayer */
#define V4L2_PIX_FMT_IPU3_SRGG10     v4l2_fourcc('i', 'p', '3', 'r') /*
↳ IPU3 packed 10-bit RGG bayer */

/* SDR formats - used only for Software Defined Radio devices */
#define V4L2_SDR_FMT_CU8              v4l2_fourcc('C', 'U', '0', '8') /* IQ u8 */
#define V4L2_SDR_FMT_CU16LE          v4l2_fourcc('C', 'U', '1', '6') /* IQ u16le
↳ */
#define V4L2_SDR_FMT_CS8              v4l2_fourcc('C', 'S', '0', '8') /* complex
↳ s8 */
#define V4L2_SDR_FMT_CS14LE          v4l2_fourcc('C', 'S', '1', '4') /* complex
↳ s14le */
#define V4L2_SDR_FMT_RU12LE          v4l2_fourcc('R', 'U', '1', '2') /* real
↳ u12le */
#define V4L2_SDR_FMT_PCU16BE          v4l2_fourcc('P', 'C', '1', '6') /* planar
↳ complex u16be */
#define V4L2_SDR_FMT_PCU18BE          v4l2_fourcc('P', 'C', '1', '8') /* planar
↳ complex u18be */
#define V4L2_SDR_FMT_PCU20BE          v4l2_fourcc('P', 'C', '2', '0') /* planar
↳ complex u20be */

/* Touch formats - used for Touch devices */
#define V4L2_TCH_FMT_DELTA_TD16      v4l2_fourcc('T', 'D', '1', '6') /* 16-bit
↳ signed deltas */
#define V4L2_TCH_FMT_DELTA_TD08      v4l2_fourcc('T', 'D', '0', '8') /* 8-bit
↳ signed deltas */
#define V4L2_TCH_FMT_TU16            v4l2_fourcc('T', 'U', '1', '6') /* 16-bit
↳ unsigned touch data */
#define V4L2_TCH_FMT_TU08            v4l2_fourcc('T', 'U', '0', '8') /* 8-bit
↳ unsigned touch data */

/* Meta-data formats */
#define V4L2_META_FMT_VSP1_HGO        v4l2_fourcc('V', 'S', 'P', 'H') /* R-Car
↳ VSP1 1-D Histogram */
#define V4L2_META_FMT_VSP1_HGT        v4l2_fourcc('V', 'S', 'P', 'T') /* R-Car
↳ VSP1 2-D Histogram */
#define V4L2_META_FMT_UVC             v4l2_fourcc('U', 'V', 'C', 'H') /* UVC
↳ Payload Header metadata */
#define V4L2_META_FMT_D4XX            v4l2_fourcc('D', '4', 'X', 'X') /* D4XX
↳ Payload Header metadata */
#define V4L2_META_FMT_VIVID           v4l2_fourcc('V', 'I', 'V', 'D') /* Vivid
↳ Metadata */

/* Vendor specific - used for RK_ISP1 camera sub-system */
#define V4L2_META_FMT_RK_ISP1_PARAMS  v4l2_fourcc('R', 'K', '1', 'P') /*
↳ Rockchip ISP1 3A Parameters */
#define V4L2_META_FMT_RK_ISP1_STAT_3A v4l2_fourcc('R', 'K', '1', 'S') /*
↳ Rockchip ISP1 3A Statistics */

```

```
/* priv field value to indicates that subsequent fields are valid. */
#define V4L2_PIX_FMT_PRIV_MAGIC          0xfeedcafe

/* Flags */
#define V4L2_PIX_FMT_FLAG_PREMUL_ALPHA  0x00000001
#define V4L2_PIX_FMT_FLAG_SET_CSC      0x00000002

/*
 *   F O R M A T   E N U M E R A T I O N
 */
struct v4l2_fmtdesc {
    __u32          index;          /* Format number      */
    __u32          type;          /* enum v4l2_buf_type */
    __u32          flags;
    __u8           description[32]; /* Description string */
    __u32          pixelformat;   /* Format fourcc      */
    __u32          mbus_code;     /* Media bus code    */
    __u32          reserved[3];
};

#define V4L2_FMT_FLAG_COMPRESSED          0x0001
#define V4L2_FMT_FLAG_EMULATED           0x0002
#define V4L2_FMT_FLAG_CONTINUOUS_BYTESTREAM 0x0004
#define V4L2_FMT_FLAG_DYN_RESOLUTION     0x0008
#define V4L2_FMT_FLAG_ENC_CAP_FRAME_INTERVAL 0x0010
#define V4L2_FMT_FLAG_CSC_COLORSPACE     0x0020
#define V4L2_FMT_FLAG_CSC_XFER_FUNC      0x0040
#define V4L2_FMT_FLAG_CSC_YCBCR_ENC      0x0080
#define V4L2_FMT_FLAG_CSC_HSV_ENC        V4L2_FMT_FLAG_CSC_YCBCR_ENC
#define V4L2_FMT_FLAG_CSC_QUANTIZATION   0x0100

/* Frame Size and frame rate enumeration */
/*
 *   F R A M E   S I Z E   E N U M E R A T I O N
 */
enum v4l2_frmsizetypes {
    V4L2_FRMSIZE_TYPE_DISCRETE          = 1,
    V4L2_FRMSIZE_TYPE_CONTINUOUS        = 2,
    V4L2_FRMSIZE_TYPE_STEPWISE          = 3,
};

struct v4l2_frmsize_discrete {
    __u32          width;         /* Frame width [pixel] */
    __u32          height;        /* Frame height [pixel] */
};

struct v4l2_frmsize_stepwise {
    __u32          min_width;     /* Minimum frame width [pixel]
→ */
    __u32          max_width;     /* Maximum frame width [pixel]

```

```

↪ */
    __u32                step_width;    /* Frame width step size_
↪ [pixel] */
    __u32                min_height;    /* Minimum frame height_
↪ [pixel] */
    __u32                max_height;    /* Maximum frame height_
↪ [pixel] */
    __u32                step_height;   /* Frame height step size_
↪ [pixel] */
};

struct v4l2_frmsizeenum {
    __u32                index;         /* Frame size number */
    __u32                pixel_format;  /* Pixel format */
    __u32                type;         /* Frame size type the device_
↪ supports. */

    union {                /* Frame size */
        struct v4l2_frmsize_discrete discrete;
        struct v4l2_frmsize_stepwise stepwise;
    };

    __u32                reserved[2];   /* Reserved space for future_
↪ use */
};

/*
 *   F R A M E   R A T E   E N U M E R A T I O N
 */
enum v4l2_frmivaltypes {
    V4L2_FRMIVAL_TYPE_DISCRETE    = 1,
    V4L2_FRMIVAL_TYPE_CONTINUOUS  = 2,
    V4L2_FRMIVAL_TYPE_STEPWISE    = 3,
};

struct v4l2_frmival_stepwise {
    struct v4l2_fract    min;          /* Minimum frame interval [s]_
↪ */
    struct v4l2_fract    max;          /* Maximum frame interval [s]_
↪ */
    struct v4l2_fract    step;         /* Frame interval step size_
↪ [s] */
};

struct v4l2_frmivalenum {
    __u32                index;         /* Frame format index */
    __u32                pixel_format;  /* Pixel format */
    __u32                width;        /* Frame width */
    __u32                height;       /* Frame height */
    __u32                type;         /* Frame interval type the_
↪ device supports. */
};

```

```

        union {
            struct v4l2_fract
            struct v4l2_frmival_stepwise
        };
        __u32 reserved[2];
    use */
};

/*
 *   T I M E C O D E
 */
struct v4l2_timecode {
    __u32 type;
    __u32 flags;
    __u8 frames;
    __u8 seconds;
    __u8 minutes;
    __u8 hours;
    __u8 userbits[4];
};

/* Type */
#define V4L2_TC_TYPE_24FPS 1
#define V4L2_TC_TYPE_25FPS 2
#define V4L2_TC_TYPE_30FPS 3
#define V4L2_TC_TYPE_50FPS 4
#define V4L2_TC_TYPE_60FPS 5

/* Flags */
#define V4L2_TC_FLAG_DROPFRAME 0x0001 /* "drop-frame" mode */
#define V4L2_TC_FLAG_COLORFRAME 0x0002
#define V4L2_TC_USERBITS_field 0x000C
#define V4L2_TC_USERBITS_USERDEFINED 0x0000
#define V4L2_TC_USERBITS_8BITCHARS 0x0008
/* The above is based on SMPTE timecodes */

struct v4l2_jpegcompression {
    int quality;

    int APPn; /* Number of APP segment to be written,
              * must be 0..15 */
    int APP_len; /* Length of data in JPEG APPn segment */
    char APP_data[60]; /* Data in the JPEG APPn segment. */

    int COM_len; /* Length of data in JPEG COM segment */
    char COM_data[60]; /* Data in JPEG COM segment */

    __u32 jpeg_markers; /* Which markers should go into the JPEG
                       * output. Unless you exactly know what

```

```

* you do, leave them untouched.
* Including less markers will make the
* resulting code smaller, but there will
* be fewer applications which can read it.
* The presence of the APP and COM marker
* is influenced by APP_len and COM_len
* ONLY, not by this property! */

#define V4L2_JPEG_MARKER_DHT (1<<3)    /* Define Huffman Tables */
#define V4L2_JPEG_MARKER_DQT (1<<4)    /* Define Quantization Tables */
#define V4L2_JPEG_MARKER_DRI (1<<5)    /* Define Restart Interval */
#define V4L2_JPEG_MARKER_COM (1<<6)    /* Comment segment */
#define V4L2_JPEG_MARKER_APP (1<<7)    /* App segment, driver will
* always use APP0 */

};

/*
 *   M E M O R Y - M A P P I N G   B U F F E R S
 */

#ifdef __KERNEL__
/*
 * This corresponds to the user space version of timeval
 * for 64-bit time_t. sparc64 is different from everyone
 * else, using the microseconds in the wrong half of the
 * second 64-bit word.
 */
struct __kernel_v4l2_timeval {
    long long    tv_sec;
#ifdef __sparc__ && defined(__arch64__)
    int          tv_usec;
    int          __pad;
#else
    long long    tv_usec;
#endif
};
#endif

struct v4l2_requestbuffers {
    __u32        count;
    __u32        type;          /* enum v4l2_buf_type */
    __u32        memory;       /* enum v4l2_memory */
    __u32        capabilities;
    __u8         flags;
    __u8         reserved[3];
};

#define V4L2_MEMORY_FLAG_NON_COHERENT    (1 << 0)

/* capabilities for struct v4l2_requestbuffers and v4l2_create_buffers */
#define V4L2_BUF_CAP_SUPPORTS_MMAP      (1 << 0)

```



```
#define V4L2_BUF_CAP_SUPPORTS_USERPTR          (1 << 1)
#define V4L2_BUF_CAP_SUPPORTS_DMABUF         (1 << 2)
#define V4L2_BUF_CAP_SUPPORTS_REQUESTS      (1 << 3)
#define V4L2_BUF_CAP_SUPPORTS_ORPHANED_BUFS (1 << 4)
#define V4L2_BUF_CAP_SUPPORTS_M2M_HOLD_CAPTURE_BUF (1 << 5)
#define V4L2_BUF_CAP_SUPPORTS_MMAP_CACHE_HINTS (1 << 6)

/**
 * struct v4l2_plane - plane info for multi-planar buffers
 * @bytesused:      number of bytes occupied by data in the plane (payload)
 * @length:         size of this plane (NOT the payload) in bytes
 * @mem_offset:     when memory in the associated struct v4l2_buffer is
 *                 V4L2_MEMORY_MMAP, equals the offset from the start of
 *                 the device memory for this plane (or is a "cookie" that
 *                 should be passed to mmap() called on the video node)
 * @userptr:        when memory is V4L2_MEMORY_USERPTR, a userspace pointer
 *                 pointing to this plane
 * @fd:             when memory is V4L2_MEMORY_DMABUF, a userspace file
 *                 descriptor associated with this plane
 * @m:              union of @mem_offset, @userptr and @fd
 * @data_offset:    offset in the plane to the start of data; usually 0,
 *                 unless there is a header in front of the data
 * @reserved:       drivers and applications must zero this array
 *
 * Multi-planar buffers consist of one or more planes, e.g. an YCbCr buffer
 * with two planes can have one plane for Y, and another for interleaved CbCr
 * components. Each plane can reside in a separate memory buffer, or even in
 * a completely separate memory node (e.g. in embedded devices).
 */
struct v4l2_plane {
    __u32          bytesused;
    __u32          length;
    union {
        __u32      mem_offset;
        unsigned long userptr;
        __s32      fd;
    } m;
    __u32          data_offset;
    __u32          reserved[11];
};

/**
 * struct v4l2_buffer - video buffer info
 * @index:          id number of the buffer
 * @type:           enum v4l2_buf_type; buffer type (type == *_MPLANE for
 *                 multiplanar buffers);
 * @bytesused:      number of bytes occupied by data in the buffer (payload);
 *                 unused (set to 0) for multiplanar buffers
 * @flags:          buffer informational flags
 * @field:          enum v4l2_field; field order of the image in the buffer
 * @timestamp:      frame timestamp
 */
```



```

* @timecode:    frame timecode
* @sequence:   sequence count of this frame
* @memory:     enum v4l2_memory; the method, in which the actual video data is
*             passed
* @offset:     for non-multiplanar buffers with memory == V4L2_MEMORY_MMAP;
*             offset from the start of the device memory for this plane,
*             (or a "cookie" that should be passed to mmap() as offset)
* @userptr:    for non-multiplanar buffers with memory == V4L2_MEMORY_USERPTR;
*             a userspace pointer pointing to this buffer
* @fd:         for non-multiplanar buffers with memory == V4L2_MEMORY_DMABUF;
*             a userspace file descriptor associated with this buffer
* @planes:     for multiplanar buffers; userspace pointer to the array of
→ plane
*             info structs for this buffer
* @m:         union of @offset, @userptr, @planes and @fd
* @length:     size in bytes of the buffer (NOT its payload) for single-plane
*             buffers (when type != *_MPLANE); number of elements in the
*             planes array for multi-plane buffers
* @reserved2:  drivers and applications must zero this field
* @request_fd: fd of the request that this buffer should use
* @reserved:   for backwards compatibility with applications that do not know
*             about @request_fd
*
* Contains data exchanged by application and driver using one of the Streaming
* I/O methods.
*/
struct v4l2_buffer {
    __u32                index;
    __u32                type;
    __u32                bytesused;
    __u32                flags;
    __u32                field;
#ifdef __KERNEL__
    struct __kernel_v4l2_timeval timestamp;
#else
    struct timeval        timestamp;
#endif
    struct v4l2_timecode  timecode;
    __u32                sequence;

    /* memory location */
    __u32                memory;
    union {
        __u32            offset;
        unsigned long    userptr;
        struct v4l2_plane *planes;
        __s32            fd;
    } m;
    __u32                length;
    __u32                reserved2;
    union {

```

```
        __s32      request_fd;
        __u32      reserved;
};

};

#ifdef __KERNEL__
/**
 * v4l2_timeval_to_ns - Convert timeval to nanoseconds
 * @tv:                pointer to the timeval variable to be converted
 *
 * Returns the scalar nanosecond representation of the timeval
 * parameter.
 */
static inline __u64 v4l2_timeval_to_ns(const struct timeval *tv)
{
    return (__u64)tv->tv_sec * 1000000000ULL + tv->tv_usec * 1000;
}
#endif

/* Flags for 'flags' field */
/* Buffer is mapped (flag) */
#define V4L2_BUF_FLAG_MAPPED                0x00000001
/* Buffer is queued for processing */
#define V4L2_BUF_FLAG_QUEUED                0x00000002
/* Buffer is ready */
#define V4L2_BUF_FLAG_DONE                  0x00000004
/* Image is a keyframe (I-frame) */
#define V4L2_BUF_FLAG_KEYFRAME              0x00000008
/* Image is a P-frame */
#define V4L2_BUF_FLAG_PFRAME                0x00000010
/* Image is a B-frame */
#define V4L2_BUF_FLAG_BFRAME                0x00000020
/* Buffer is ready, but the data contained within is corrupted. */
#define V4L2_BUF_FLAG_ERROR                  0x00000040
/* Buffer is added to an unqueued request */
#define V4L2_BUF_FLAG_IN_REQUEST            0x00000080
/* timecode field is valid */
#define V4L2_BUF_FLAG_TIMECODE              0x00000100
/* Don't return the capture buffer until OUTPUT timestamp changes */
#define V4L2_BUF_FLAG_M2M_HOLD_CAPTURE_BUF  0x00000200
/* Buffer is prepared for queuing */
#define V4L2_BUF_FLAG_PREPARED              0x00000400
/* Cache handling flags */
#define V4L2_BUF_FLAG_NO_CACHE_INVALIDATE    0x00000800
#define V4L2_BUF_FLAG_NO_CACHE_CLEAN        0x00001000
/* Timestamp type */
#define V4L2_BUF_FLAG_TIMESTAMP_MASK        0x0000e000
#define V4L2_BUF_FLAG_TIMESTAMP_UNKNOWN     0x00000000
#define V4L2_BUF_FLAG_TIMESTAMP_MONOTONIC   0x00002000
#define V4L2_BUF_FLAG_TIMESTAMP_COPY        0x00004000
/* Timestamp sources. */
```

```

#define V4L2_BUF_FLAG_TSTAMP_SRC_MASK          0x00070000
#define V4L2_BUF_FLAG_TSTAMP_SRC_EOF          0x00000000
#define V4L2_BUF_FLAG_TSTAMP_SRC_SOE          0x00010000
/* mem2mem encoder/decoder */
#define V4L2_BUF_FLAG_LAST                      0x00100000
/* request_fd is valid */
#define V4L2_BUF_FLAG_REQUEST_FD              0x00800000

/**
 * struct v4l2_exportbuffer - export of video buffer as DMABUF file descriptor
 *
 * @index:      id number of the buffer
 * @type:       enum v4l2_buf_type; buffer type (type == *_MPLANE for
 *             multiplanar buffers);
 * @plane:     index of the plane to be exported, 0 for single plane queues
 * @flags:     flags for newly created file, currently only O_CLOEXEC is
 *             supported, refer to manual of open syscall for more details
 * @fd:       file descriptor associated with DMABUF (set by driver)
 * @reserved:  drivers and applications must zero this array
 *
 * Contains data used for exporting a video buffer as DMABUF file descriptor.
 * The buffer is identified by a 'cookie' returned by VIDIOC_QUERYBUF
 * (identical to the cookie used to mmap() the buffer to userspace). All
 * reserved fields must be set to zero. The field reserved0 is expected to
 * become a structure 'type' allowing an alternative layout of the structure
 * content. Therefore this field should not be used for any other extensions.
 */
struct v4l2_exportbuffer {
    __u32          type; /* enum v4l2_buf_type */
    __u32          index;
    __u32          plane;
    __u32          flags;
    __s32          fd;
    __u32          reserved[11];
};

/*
 *   O V E R L A Y   P R E V I E W
 */
struct v4l2_framebuffer {
    __u32          capability;
    __u32          flags;
/* FIXME: in theory we should pass something like PCI device + memory
 * region + offset instead of some physical address */
    void          *base;
    struct {
        __u32      width;
        __u32      height;
        __u32      pixelformat;
        __u32      field;          /* enum v4l2_field */
        __u32      bytesperline; /* for padding, zero if unused */
    };
};

```

```

→ */
        __u32          sizeimage;
        __u32          colorspace; /* enum v4l2_colorspace */
        __u32          priv;      /* reserved field, set to 0 */
    } fmt;
};
/* Flags for the 'capability' field. Read only */
#define V4L2_FBUF_CAP_EXTERNOVERLAY    0x0001
#define V4L2_FBUF_CAP_CHROMAKEY       0x0002
#define V4L2_FBUF_CAP_LIST_CLIPPING   0x0004
#define V4L2_FBUF_CAP_BITMAP_CLIPPING 0x0008
#define V4L2_FBUF_CAP_LOCAL_ALPHA      0x0010
#define V4L2_FBUF_CAP_GLOBAL_ALPHA     0x0020
#define V4L2_FBUF_CAP_LOCAL_INV_ALPHA  0x0040
#define V4L2_FBUF_CAP_SRC_CHROMAKEY    0x0080
/* Flags for the 'flags' field. */
#define V4L2_FBUF_FLAG_PRIMARY         0x0001
#define V4L2_FBUF_FLAG_OVERLAY        0x0002
#define V4L2_FBUF_FLAG_CHROMAKEY      0x0004
#define V4L2_FBUF_FLAG_LOCAL_ALPHA     0x0008
#define V4L2_FBUF_FLAG_GLOBAL_ALPHA    0x0010
#define V4L2_FBUF_FLAG_LOCAL_INV_ALPHA 0x0020
#define V4L2_FBUF_FLAG_SRC_CHROMAKEY   0x0040

struct v4l2_clip {
    struct v4l2_rect    c;
    struct v4l2_clip   __user *next;
};

struct v4l2_window {
    struct v4l2_rect    w;
    __u32              field; /* enum v4l2_field */
    __u32              chromakey;
    struct v4l2_clip   *clips;
    __u32              clipcount;
    void               __user *bitmap;
    __u8               global_alpha;
};

/*
 *   C A P T U R E   P A R A M E T E R S
 */
struct v4l2_captureparm {
    __u32              capability; /* Supported modes */
    __u32              capturemode; /* Current mode */
    struct v4l2_fract  timeperframe; /* Time per frame in seconds */
    __u32              extendedmode; /* Driver-specific extensions */
    __u32              readbuffers; /* # of buffers for read */
    __u32              reserved[4];
};

```

```

/* Flags for 'capability' and 'capturemode' fields */
#define V4L2_MODE_HIGHQUALITY 0x0001 /* High quality imaging mode */
#define V4L2_CAP_TIMEPERFRAME 0x1000 /* timeperframe field is supported */

struct v4l2_outputparm {
    __u32          capability; /* Supported modes */
    __u32          outputmode; /* Current mode */
    struct v4l2_fract timeperframe; /* Time per frame in seconds */
    __u32          extendedmode; /* Driver-specific extensions */
    __u32          writebuffers; /* # of buffers for write */
    __u32          reserved[4];
};

/*
 *   I N P U T   I M A G E   C R O P P I N G
 */
struct v4l2_cropcap {
    __u32          type; /* enum v4l2_buf_type */
    struct v4l2_rect bounds;
    struct v4l2_rect defrect;
    struct v4l2_fract pixelaspect;
};

struct v4l2_crop {
    __u32          type; /* enum v4l2_buf_type */
    struct v4l2_rect c;
};

/**
 * struct v4l2_selection - selection info
 * @type:          buffer type (do not use *_MPLANE types)
 * @target:        Selection target, used to choose one of possible rectangles;
 *                 defined in v4l2-common.h; V4L2_SEL_TGT_* .
 * @flags:         constraints flags, defined in v4l2-common.h; V4L2_SEL_FLAG_*.
 * @r:             coordinates of selection window
 * @reserved:      for future use, rounds structure size to 64 bytes, set to zero
 *
 * Hardware may use multiple helper windows to process a video stream.
 * The structure is used to exchange this selection areas between
 * an application and a driver.
 */
struct v4l2_selection {
    __u32          type;
    __u32          target;
    __u32          flags;
    struct v4l2_rect r;
    __u32          reserved[9];
};

/*
 *   A N A L O G   V I D E O   S T A N D A R D
 */

```

```
*/  
  
typedef __u64 v4l2_std_id;  
  
/*  
 * Attention: Keep the V4L2_STD_* bit definitions in sync with  
 * include/dt-bindings/display/sdtv-standards.h SDTV_STD_* bit definitions.  
 */  
/* one bit for each */  
#define V4L2_STD_PAL_B ((v4l2_std_id)0x00000001)  
#define V4L2_STD_PAL_B1 ((v4l2_std_id)0x00000002)  
#define V4L2_STD_PAL_G ((v4l2_std_id)0x00000004)  
#define V4L2_STD_PAL_H ((v4l2_std_id)0x00000008)  
#define V4L2_STD_PAL_I ((v4l2_std_id)0x00000010)  
#define V4L2_STD_PAL_D ((v4l2_std_id)0x00000020)  
#define V4L2_STD_PAL_D1 ((v4l2_std_id)0x00000040)  
#define V4L2_STD_PAL_K ((v4l2_std_id)0x00000080)  
  
#define V4L2_STD_PAL_M ((v4l2_std_id)0x00000100)  
#define V4L2_STD_PAL_N ((v4l2_std_id)0x00000200)  
#define V4L2_STD_PAL_Nc ((v4l2_std_id)0x00000400)  
#define V4L2_STD_PAL_60 ((v4l2_std_id)0x00000800)  
  
#define V4L2_STD_NTSC_M ((v4l2_std_id)0x00001000) /* BTSC */  
#define V4L2_STD_NTSC_M_JP ((v4l2_std_id)0x00002000) /* EIA-J */  
#define V4L2_STD_NTSC_443 ((v4l2_std_id)0x00004000)  
#define V4L2_STD_NTSC_M_KR ((v4l2_std_id)0x00008000) /* FM A2 */  
  
#define V4L2_STD_SECAM_B ((v4l2_std_id)0x00010000)  
#define V4L2_STD_SECAM_D ((v4l2_std_id)0x00020000)  
#define V4L2_STD_SECAM_G ((v4l2_std_id)0x00040000)  
#define V4L2_STD_SECAM_H ((v4l2_std_id)0x00080000)  
#define V4L2_STD_SECAM_K ((v4l2_std_id)0x00100000)  
#define V4L2_STD_SECAM_K1 ((v4l2_std_id)0x00200000)  
#define V4L2_STD_SECAM_L ((v4l2_std_id)0x00400000)  
#define V4L2_STD_SECAM_LC ((v4l2_std_id)0x00800000)  
  
/* ATSC/HDTV */  
#define V4L2_STD_ATSC_8_VSB ((v4l2_std_id)0x01000000)  
#define V4L2_STD_ATSC_16_VSB ((v4l2_std_id)0x02000000)  
  
/* FIXME:  
Although std_id is 64 bits, there is an issue on PPC32 architecture that  
makes switch(__u64) to break. So, there's a hack on v4l2-common.c rounding  
this value to 32 bits.  
As, currently, the max value is for V4L2_STD_ATSC_16_VSB (30 bits wide),  
it should work fine. However, if needed to add more than two standards,  
v4l2-common.c should be fixed.  
*/  
  
/*
```

```

* Some macros to merge video standards in order to make live easier for the
* drivers and V4L2 applications
*/

/*
* "Common" NTSC/M - It should be noticed that V4L2_STD_NTSC_443 is
* Missing here.
*/
#define V4L2_STD_NTSC          (V4L2_STD_NTSC_M          |\
                               V4L2_STD_NTSC_M_JP       |\
                               V4L2_STD_NTSC_M_KR)

/* Secam macros */
#define V4L2_STD_SECAM_DK     (V4L2_STD_SECAM_D          |\
                               V4L2_STD_SECAM_K           |\
                               V4L2_STD_SECAM_K1)

/* All Secam Standards */
#define V4L2_STD_SECAM        (V4L2_STD_SECAM_B          |\
                               V4L2_STD_SECAM_G           |\
                               V4L2_STD_SECAM_H           |\
                               V4L2_STD_SECAM_DK          |\
                               V4L2_STD_SECAM_L           |\
                               V4L2_STD_SECAM_LC)

/* PAL macros */
#define V4L2_STD_PAL_BG       (V4L2_STD_PAL_B           |\
                               V4L2_STD_PAL_B1            |\
                               V4L2_STD_PAL_G)
#define V4L2_STD_PAL_DK       (V4L2_STD_PAL_D           |\
                               V4L2_STD_PAL_D1            |\
                               V4L2_STD_PAL_K)

/*
* "Common" PAL - This macro is there to be compatible with the old
* V4L1 concept of "PAL": /BGDKHI.
* Several PAL standards are missing here: /M, /N and /Nc
*/
#define V4L2_STD_PAL          (V4L2_STD_PAL_BG          |\
                               V4L2_STD_PAL_DK           |\
                               V4L2_STD_PAL_H             |\
                               V4L2_STD_PAL_I)

/* Chroma "agnostic" standards */
#define V4L2_STD_B            (V4L2_STD_PAL_B           |\
                               V4L2_STD_PAL_B1            |\
                               V4L2_STD_SECAM_B)
#define V4L2_STD_G            (V4L2_STD_PAL_G           |\
                               V4L2_STD_SECAM_G)
#define V4L2_STD_H            (V4L2_STD_PAL_H           |\
                               V4L2_STD_SECAM_H)
#define V4L2_STD_L            (V4L2_STD_SECAM_L         |\
                               V4L2_STD_SECAM_LC)
#define V4L2_STD_GH           (V4L2_STD_G              |\
                               V4L2_STD_H)
#define V4L2_STD_DK           (V4L2_STD_PAL_DK)

```

```

        V4L2_STD_SECAM_DK)
#define V4L2_STD_BG      (V4L2_STD_B          |\
        V4L2_STD_G)
#define V4L2_STD_MN      (V4L2_STD_PAL_M       |\
        V4L2_STD_PAL_N   |\
        V4L2_STD_PAL_Nc  |\
        V4L2_STD_NTSC)

/* Standards where MTS/BTSC stereo could be found */
#define V4L2_STD_MTS     (V4L2_STD_NTSC_M      |\
        V4L2_STD_PAL_M   |\
        V4L2_STD_PAL_N   |\
        V4L2_STD_PAL_Nc)

/* Standards for Countries with 60Hz Line frequency */
#define V4L2_STD_525_60 (V4L2_STD_PAL_M       |\
        V4L2_STD_PAL_60  |\
        V4L2_STD_NTSC    |\
        V4L2_STD_NTSC_443)

/* Standards for Countries with 50Hz Line frequency */
#define V4L2_STD_625_50 (V4L2_STD_PAL        |\
        V4L2_STD_PAL_N   |\
        V4L2_STD_PAL_Nc  |\
        V4L2_STD_SECAM)

#define V4L2_STD_ATSC    (V4L2_STD_ATSC_8_VSB  |\
        V4L2_STD_ATSC_16_VSB)

/* Macros with none and all analog standards */
#define V4L2_STD_UNKNOWN 0
#define V4L2_STD_ALL     (V4L2_STD_525_60     |\
        V4L2_STD_625_50)

struct v4l2_standard {
    __u32          index;
    v4l2_std_id    id;
    __u8          name[24];
    struct v4l2_fract frameperiod; /* Frames, not fields */
    __u32          framelines;
    __u32          reserved[4];
};

/*
 *      D V      B T      T I M I N G S
 */

/** struct v4l2_bt_timings - BT.656/BT.1120 timing data
 * @width:      total width of the active video in pixels
 * @height:     total height of the active video in lines
 * @interlaced: Interlaced or progressive
 * @polarities: Positive or negative polarities
 * @pixelclock: Pixel clock in HZ. Ex. 74.25MHz->74250000

```



```

* @hfrontporch: Horizontal front porch in pixels
* @hsync:       Horizontal Sync length in pixels
* @hbackporch: Horizontal back porch in pixels
* @vfrontporch: Vertical front porch in lines
* @vsync:       Vertical Sync length in lines
* @vbackporch: Vertical back porch in lines
* @il_vfrontporch: Vertical front porch for the even field
*                (aka field 2) of interlaced field formats
* @il_vsync:    Vertical Sync length for the even field
*                (aka field 2) of interlaced field formats
* @il_vbackporch: Vertical back porch for the even field
*                (aka field 2) of interlaced field formats
* @standards:   Standards the timing belongs to
* @flags:       Flags
* @picture_aspect: The picture aspect ratio (hor/vert).
* @cea861_vic:  VIC code as per the CEA-861 standard.
* @hdmi_vic:    VIC code as per the HDMI standard.
* @reserved:    Reserved fields, must be zeroed.
*
* A note regarding vertical interlaced timings: height refers to the total
* height of the active video frame (= two fields). The blanking timings refer
* to the blanking of each field. So the height of the total frame is
* calculated as follows:
*
* tot_height = height + vfrontporch + vsync + vbackporch +
*                il_vfrontporch + il_vsync + il_vbackporch
*
* The active height of each field is height / 2.
*/
struct v4l2_bt_timings {
    __u32    width;
    __u32    height;
    __u32    interlaced;
    __u32    polarities;
    __u64    pixelclock;
    __u32    hfrontporch;
    __u32    hsync;
    __u32    hbackporch;
    __u32    vfrontporch;
    __u32    vsync;
    __u32    vbackporch;
    __u32    il_vfrontporch;
    __u32    il_vsync;
    __u32    il_vbackporch;
    __u32    standards;
    __u32    flags;
    struct v4l2_fract picture_aspect;
    __u8     cea861_vic;
    __u8     hdmi_vic;
    __u8     reserved[46];
} __attribute__((packed));

```

```
/* Interlaced or progressive format */
#define V4L2_DV_PROGRESSIVE      0
#define V4L2_DV_INTERLACED      1

/* Polarities. If bit is not set, it is assumed to be negative polarity */
#define V4L2_DV_VSYNC_POS_POL    0x00000001
#define V4L2_DV_HSYNC_POS_POL    0x00000002

/* Timings standards */
#define V4L2_DV_BT_STD_CEA861    (1 << 0) /* CEA-861 Digital TV Profile */
#define V4L2_DV_BT_STD_DMT       (1 << 1) /* VESA Discrete Monitor Timings */
#define V4L2_DV_BT_STD_CVT       (1 << 2) /* VESA Coordinated Video Timings */
#define V4L2_DV_BT_STD_GTF       (1 << 3) /* VESA Generalized Timings Formula
↳*/
#define V4L2_DV_BT_STD_SDI       (1 << 4) /* SDI Timings */

/* Flags */

/*
 * CVT/GTF specific: timing uses reduced blanking (CVT) or the 'Secondary
 * GTF' curve (GTF). In both cases the horizontal and/or vertical blanking
 * intervals are reduced, allowing a higher resolution over the same
 * bandwidth. This is a read-only flag.
 */
#define V4L2_DV_FL_REDUCED_BLANKING (1 << 0)
/*
 * CEA-861 specific: set for CEA-861 formats with a framerate of a multiple
 * of six. These formats can be optionally played at 1 / 1.001 speed.
 * This is a read-only flag.
 */
#define V4L2_DV_FL_CAN_REDUCE_FPS   (1 << 1)
/*
 * CEA-861 specific: only valid for video transmitters, the flag is cleared
 * by receivers.
 * If the framerate of the format is a multiple of six, then the pixelclock
 * used to set up the transmitter is divided by 1.001 to make it compatible
 * with 60 Hz based standards such as NTSC and PAL-M that use a framerate of
 * 29.97 Hz. Otherwise this flag is cleared. If the transmitter can't generate
 * such frequencies, then the flag will also be cleared.
 */
#define V4L2_DV_FL_REDUCED_FPS     (1 << 2)
/*
 * Specific to interlaced formats: if set, then field 1 is really one half-line
 * longer and field 2 is really one half-line shorter, so each field has
 * exactly the same number of half-lines. Whether half-lines can be detected
 * or used depends on the hardware.
 */
#define V4L2_DV_FL_HALF_LINE       (1 << 3)
/*
 * If set, then this is a Consumer Electronics (CE) video format. Such formats
```

```

* differ from other formats (commonly called IT formats) in that if RGB
* encoding is used then by default the RGB values use limited range (i.e.
* use the range 16-235) as opposed to 0-255. All formats defined in CEA-861
* except for the 640x480 format are CE formats.
*/
#define V4L2_DV_FL_IS_CE_VIDEO                (1 << 4)
/* Some formats like SMPTE-125M have an interlaced signal with a odd
* total height. For these formats, if this flag is set, the first
* field has the extra line. If not, it is the second field.
*/
#define V4L2_DV_FL_FIRST_FIELD_EXTRA_LINE    (1 << 5)
/*
* If set, then the picture_aspect field is valid. Otherwise assume that the
* pixels are square, so the picture aspect ratio is the same as the width to
* height ratio.
*/
#define V4L2_DV_FL_HAS_PICTURE_ASPECT        (1 << 6)
/*
* If set, then the cea861_vic field is valid and contains the Video
* Identification Code as per the CEA-861 standard.
*/
#define V4L2_DV_FL_HAS_CEA861_VIC           (1 << 7)
/*
* If set, then the hdmi_vic field is valid and contains the Video
* Identification Code as per the HDMI standard (HDMI Vendor Specific
* InfoFrame).
*/
#define V4L2_DV_FL_HAS_HDMI_VIC             (1 << 8)
/*
* CEA-861 specific: only valid for video receivers.
* If set, then HW can detect the difference between regular FPS and
* 1000/1001 FPS. Note: This flag is only valid for HDMI VIC codes with
* the V4L2_DV_FL_CAN_REDUCE_FPS flag set.
*/
#define V4L2_DV_FL_CAN_DETECT_REDUCED_FPS   (1 << 9)

/* A few useful defines to calculate the total blanking and frame sizes */
#define V4L2_DV_BT_BLANKING_WIDTH(bt) \
    ((bt)->hfrontporch + (bt)->hsync + (bt)->hbackporch)
#define V4L2_DV_BT_FRAME_WIDTH(bt) \
    ((bt)->width + V4L2_DV_BT_BLANKING_WIDTH(bt))
#define V4L2_DV_BT_BLANKING_HEIGHT(bt) \
    ((bt)->vfrontporch + (bt)->vsync + (bt)->vbackporch + \
    (bt)->il_vfrontporch + (bt)->il_vsync + (bt)->il_vbackporch)
#define V4L2_DV_BT_FRAME_HEIGHT(bt) \
    ((bt)->height + V4L2_DV_BT_BLANKING_HEIGHT(bt))

/** struct v4l2_dv_timings - DV timings
 * @type:          the type of the timings
 * @bt: BT656/1120 timings
 */

```

```
struct v4l2_dv_timings {
    __u32 type;
    union {
        struct v4l2_bt_timings bt;
        __u32 reserved[32];
    };
} __attribute__((packed));

/* Values for the type field */
#define V4L2_DV_BT_656_1120 0 /* BT.656/1120 timing type */

/** struct v4l2_enum_dv_timings - DV timings enumeration
 * @index: enumeration index
 * @pad: the pad number for which to enumerate timings (used with
 * v4l-subdev nodes only)
 * @reserved: must be zeroed
 * @timings: the timings for the given index
 */
struct v4l2_enum_dv_timings {
    __u32 index;
    __u32 pad;
    __u32 reserved[2];
    struct v4l2_dv_timings timings;
};

/** struct v4l2_bt_timings_cap - BT.656/BT.1120 timing capabilities
 * @min_width: width in pixels
 * @max_width: width in pixels
 * @min_height: height in lines
 * @max_height: height in lines
 * @min_pixelclock: Pixel clock in HZ. Ex. 74.25MHz->74250000
 * @max_pixelclock: Pixel clock in HZ. Ex. 74.25MHz->74250000
 * @standards: Supported standards
 * @capabilities: Supported capabilities
 * @reserved: Must be zeroed
 */
struct v4l2_bt_timings_cap {
    __u32 min_width;
    __u32 max_width;
    __u32 min_height;
    __u32 max_height;
    __u64 min_pixelclock;
    __u64 max_pixelclock;
    __u32 standards;
    __u32 capabilities;
    __u32 reserved[16];
} __attribute__((packed));

/* Supports interlaced formats */
#define V4L2_DV_BT_CAP_INTERLACED (1 << 0)
/* Supports progressive formats */
```

```

#define V4L2_DV_BT_CAP_PROGRESSIVE      (1 << 1)
/* Supports CVT/GTF reduced blanking */
#define V4L2_DV_BT_CAP_REDUCED_BLANKING (1 << 2)
/* Supports custom formats */
#define V4L2_DV_BT_CAP_CUSTOM           (1 << 3)

/** struct v4l2_dv_timings_cap - DV timings capabilities
 * @type:      the type of the timings (same as in struct v4l2_dv_timings)
 * @pad:      the pad number for which to query capabilities (used with
 *            v4l-subdev nodes only)
 * @bt:      the BT656/1120 timings capabilities
 */
struct v4l2_dv_timings_cap {
    __u32 type;
    __u32 pad;
    __u32 reserved[2];
    union {
        struct v4l2_bt_timings_cap bt;
        __u32 raw_data[32];
    };
};

/*
 *   V I D E O   I N P U T S
 */
struct v4l2_input {
    __u32      index;           /* Which input */
    __u8       name[32];       /* Label */
    __u32      type;           /* Type of input */
    __u32      audioset;       /* Associated audios (bitfield) */
    __u32      tuner;         /* enum v4l2_tuner_type */
    v4l2_std_id std;
    __u32      status;
    __u32      capabilities;
    __u32      reserved[3];
};

/* Values for the 'type' field */
#define V4L2_INPUT_TYPE_TUNER          1
#define V4L2_INPUT_TYPE_CAMERA        2
#define V4L2_INPUT_TYPE_TOUCH         3

/* field 'status' - general */
#define V4L2_IN_ST_NO_POWER            0x00000001 /* Attached device is off */
#define V4L2_IN_ST_NO_SIGNAL          0x00000002
#define V4L2_IN_ST_NO_COLOR           0x00000004

/* field 'status' - sensor orientation */
/* If sensor is mounted upside down set both bits */
#define V4L2_IN_ST_HFLIP               0x00000010 /* Frames are flipped horizontally */
#define V4L2_IN_ST_VFLIP               0x00000020 /* Frames are flipped vertically */

```

```

/* field 'status' - analog */
#define V4L2_IN_ST_NO_H_LOCK      0x00000100 /* No horizontal sync lock */
#define V4L2_IN_ST_COLOR_KILL    0x00000200 /* Color killer is active */
#define V4L2_IN_ST_NO_V_LOCK     0x00000400 /* No vertical sync lock */
#define V4L2_IN_ST_NO_STD_LOCK   0x00000800 /* No standard format lock */

/* field 'status' - digital */
#define V4L2_IN_ST_NO_SYNC       0x00010000 /* No synchronization lock */
#define V4L2_IN_ST_NO_EQU       0x00020000 /* No equalizer lock */
#define V4L2_IN_ST_NO_CARRIER  0x00040000 /* Carrier recovery failed */

/* field 'status' - VCR and set-top box */
#define V4L2_IN_ST_MACROVISION  0x01000000 /* Macrovision detected */
#define V4L2_IN_ST_NO_ACCESS    0x02000000 /* Conditional access denied */
#define V4L2_IN_ST_VTR          0x04000000 /* VTR time constant */

/* capabilities flags */
#define V4L2_IN_CAP_DV_TIMINGS   0x00000002 /* Supports S_DV_TIMINGS */
#define V4L2_IN_CAP_CUSTOM_TIMINGS /* For
↳ compatibility */
#define V4L2_IN_CAP_STD          0x00000004 /* Supports S_STD */
#define V4L2_IN_CAP_NATIVE_SIZE /* Supports setting native
↳ size */

/*
 *   V I D E O   O U T P U T S
 */
struct v4l2_output {
    __u32      index; /* Which output */
    __u8       name[32]; /* Label */
    __u32      type; /* Type of output */
    __u32      audioset; /* Associated audios (bitfield) */
    __u32      modulator; /* Associated modulator */
    v4l2_std_id std;
    __u32      capabilities;
    __u32      reserved[3];
};
/* Values for the 'type' field */
#define V4L2_OUTPUT_TYPE_MODULATOR 1
#define V4L2_OUTPUT_TYPE_ANALOG 2
#define V4L2_OUTPUT_TYPE_ANALOGVGAOVERLAY 3

/* capabilities flags */
#define V4L2_OUT_CAP_DV_TIMINGS   0x00000002 /* Supports S_DV_TIMINGS */
#define V4L2_OUT_CAP_CUSTOM_TIMINGS /* For
↳ compatibility */
#define V4L2_OUT_CAP_STD          0x00000004 /* Supports S_STD */
#define V4L2_OUT_CAP_NATIVE_SIZE /* Supports setting native
↳ size */

```

```

/*
 *      C O N T R O L S
 */
struct v4l2_control {
    __u32          id;
    __s32          value;
};

struct v4l2_ext_control {
    __u32 id;
    __u32 size;
    __u32 reserved2[1];
    union {
        __s32 value;
        __s64 value64;
        char __user *string;
        __u8 __user *p_u8;
        __u16 __user *p_u16;
        __u32 __user *p_u32;
        struct v4l2_area __user *p_area;
        struct v4l2_ctrl_h264_sps __user *p_h264_sps;
        struct v4l2_ctrl_h264_pps *p_h264_pps;
        struct v4l2_ctrl_h264_scaling_matrix __user *p_h264_scaling_
↪matrix;
        struct v4l2_ctrl_h264_pred_weights __user *p_h264_pred_weights;
        struct v4l2_ctrl_h264_slice_params __user *p_h264_slice_params;
        struct v4l2_ctrl_h264_decode_params __user *p_h264_decode_
↪params;
        struct v4l2_ctrl_fwht_params __user *p_fwht_params;
        struct v4l2_ctrl_vp8_frame __user *p_vp8_frame;
        struct v4l2_ctrl_mpeg2_sequence __user *p_mpeg2_sequence;
        struct v4l2_ctrl_mpeg2_picture __user *p_mpeg2_picture;
        struct v4l2_ctrl_mpeg2_quantisation __user *p_mpeg2_
↪quantisation;
        struct v4l2_ctrl_vp9_compressed_hdr __user *p_vp9_compressed_
↪hdr_probs;
        struct v4l2_ctrl_vp9_frame __user *p_vp9_frame;
        void __user *ptr;
    };
} __attribute__((packed));

struct v4l2_ext_controls {
    union {
#ifdef __KERNEL__
        __u32 ctrl_class;
#endif
        __u32 which;
    };
    __u32 count;
    __u32 error_idx;
    __s32 request_fd;
};

```

```
    __u32 reserved[1];
    struct v4l2_ext_control *controls;
};

#define V4L2_CTRL_ID_MASK          (0xffffffff)
#ifndef __KERNEL__
#define V4L2_CTRL_ID2CLASS(id)     ((id) & 0xffff0000UL)
#endif
#define V4L2_CTRL_ID2WHICH(id)    ((id) & 0xffff0000UL)
#define V4L2_CTRL_DRIVER_PRIV(id) (((id) & 0xffff) >= 0x1000)
#define V4L2_CTRL_MAX_DIMS        (4)
#define V4L2_CTRL_WHICH_CUR_VAL   0
#define V4L2_CTRL_WHICH_DEF_VAL   0x0f000000
#define V4L2_CTRL_WHICH_REQUEST_VAL 0x0f010000

enum v4l2_ctrl_type {
    V4L2_CTRL_TYPE_INTEGER          = 1,
    V4L2_CTRL_TYPE_BOOLEAN         = 2,
    V4L2_CTRL_TYPE_MENU            = 3,
    V4L2_CTRL_TYPE_BUTTON          = 4,
    V4L2_CTRL_TYPE_INTEGER64       = 5,
    V4L2_CTRL_TYPE_CTRL_CLASS      = 6,
    V4L2_CTRL_TYPE_STRING          = 7,
    V4L2_CTRL_TYPE_BITMASK        = 8,
    V4L2_CTRL_TYPE_INTEGER_MENU    = 9,

    /* Compound types are >= 0x0100 */
    V4L2_CTRL_COMPOUND_TYPES       = 0x0100,
    V4L2_CTRL_TYPE_U8              = 0x0100,
    V4L2_CTRL_TYPE_U16             = 0x0101,
    V4L2_CTRL_TYPE_U32            = 0x0102,
    V4L2_CTRL_TYPE_AREA            = 0x0106,

    V4L2_CTRL_TYPE_HDR10_CLL_INFO   = 0x0110,
    V4L2_CTRL_TYPE_HDR10_MASTERING_DISPLAY = 0x0111,

    V4L2_CTRL_TYPE_H264_SPS        = 0x0200,
    V4L2_CTRL_TYPE_H264_PPS        = 0x0201,
    V4L2_CTRL_TYPE_H264_SCALING_MATRIX = 0x0202,
    V4L2_CTRL_TYPE_H264_SLICE_PARAMS = 0x0203,
    V4L2_CTRL_TYPE_H264_DECODE_PARAMS = 0x0204,
    V4L2_CTRL_TYPE_H264_PRED_WEIGHTS = 0x0205,

    V4L2_CTRL_TYPE_FWHT_PARAMS     = 0x0220,

    V4L2_CTRL_TYPE_VP8_FRAME       = 0x0240,

    V4L2_CTRL_TYPE_MPEG2_QUANTISATION = 0x0250,
    V4L2_CTRL_TYPE_MPEG2_SEQUENCE  = 0x0251,
    V4L2_CTRL_TYPE_MPEG2_PICTURE   = 0x0252,
```



```

        V4L2_CTRL_TYPE_VP9_COMPRESSED_HDR          = 0x0260,
        V4L2_CTRL_TYPE_VP9_FRAME                  = 0x0261,
};

/* Used in the VIDIOC_QUERYCTRL ioctl for querying controls */
struct v4l2_queryctrl {
    __u32          id;
    __u32          type;          /* enum v4l2_ctrl_type */
    __u8           name[32];     /* Whatever */
    __s32          minimum;     /* Note signedness */
    __s32          maximum;
    __s32          step;
    __s32          default_value;
    __u32          flags;
    __u32          reserved[2];
};

/* Used in the VIDIOC_QUERY_EXT_CTRL ioctl for querying extended controls */
struct v4l2_query_ext_ctrl {
    __u32          id;
    __u32          type;
    char           name[32];
    __s64          minimum;
    __s64          maximum;
    __u64          step;
    __s64          default_value;
    __u32          flags;
    __u32          elem_size;
    __u32          elems;
    __u32          nr_of_dims;
    __u32          dims[V4L2_CTRL_MAX_DIMS];
    __u32          reserved[32];
};

/* Used in the VIDIOC_QUERYMENU ioctl for querying menu items */
struct v4l2_querymenu {
    __u32          id;
    __u32          index;
    union {
        __u8       name[32];          /* Whatever */
        __s64      value;
    };
    __u32          reserved;
} __attribute__((packed));

/* Control flags */
#define V4L2_CTRL_FLAG_DISABLED          0x0001
#define V4L2_CTRL_FLAG_GRABBED          0x0002
#define V4L2_CTRL_FLAG_READ_ONLY        0x0004
#define V4L2_CTRL_FLAG_UPDATE           0x0008
#define V4L2_CTRL_FLAG_INACTIVE         0x0010

```

```
#define V4L2_CTRL_FLAG_SLIDER          0x0020
#define V4L2_CTRL_FLAG_WRITE_ONLY     0x0040
#define V4L2_CTRL_FLAG_VOLATILE       0x0080
#define V4L2_CTRL_FLAG_HAS_PAYLOAD    0x0100
#define V4L2_CTRL_FLAG_EXECUTE_ON_WRITE 0x0200
#define V4L2_CTRL_FLAG_MODIFY_LAYOUT  0x0400

/* Query flags, to be ORed with the control ID */
#define V4L2_CTRL_FLAG_NEXT_CTRL      0x80000000
#define V4L2_CTRL_FLAG_NEXT_COMPOUND  0x40000000

/* User-class control IDs defined by V4L2 */
#define V4L2_CID_MAX_CTRL             1024
/* IDs reserved for driver specific controls */
#define V4L2_CID_PRIVATE_BASE         0x08000000

/*
 *   T U N I N G
 */
struct v4l2_tuner {
    __u32          index;
    __u8          name[32];
    __u32          type; /* enum v4l2_tuner_type */
    __u32          capability;
    __u32          rangelow;
    __u32          rangehigh;
    __u32          rxsubchans;
    __u32          audmode;
    __s32          signal;
    __s32          afc;
    __u32          reserved[4];
};

struct v4l2_modulator {
    __u32          index;
    __u8          name[32];
    __u32          capability;
    __u32          rangelow;
    __u32          rangehigh;
    __u32          txsubchans;
    __u32          type; /* enum v4l2_tuner_type */
    __u32          reserved[3];
};

/* Flags for the 'capability' field */
#define V4L2_TUNER_CAP_LOW             0x0001
#define V4L2_TUNER_CAP_NORM           0x0002
#define V4L2_TUNER_CAP_HWSEEK_BOUNDED 0x0004
#define V4L2_TUNER_CAP_HWSEEK_WRAP    0x0008
#define V4L2_TUNER_CAP_STEREO        0x0010
#define V4L2_TUNER_CAP_LANG2         0x0020
```

```

#define V4L2_TUNER_CAP_SAP                0x0020
#define V4L2_TUNER_CAP_LANG1             0x0040
#define V4L2_TUNER_CAP_RDS              0x0080
#define V4L2_TUNER_CAP_RDS_BLOCK_IO     0x0100
#define V4L2_TUNER_CAP_RDS_CONTR0LS    0x0200
#define V4L2_TUNER_CAP_FREQ_BANDS      0x0400
#define V4L2_TUNER_CAP_HWSEEK_PROG_LIM  0x0800
#define V4L2_TUNER_CAP_1HZ              0x1000

/* Flags for the 'rxsubchans' field */
#define V4L2_TUNER_SUB_MONO              0x0001
#define V4L2_TUNER_SUB_STEREO           0x0002
#define V4L2_TUNER_SUB_LANG2            0x0004
#define V4L2_TUNER_SUB_SAP              0x0004
#define V4L2_TUNER_SUB_LANG1            0x0008
#define V4L2_TUNER_SUB_RDS              0x0010

/* Values for the 'audmode' field */
#define V4L2_TUNER_MODE_MONO             0x0000
#define V4L2_TUNER_MODE_STEREO          0x0001
#define V4L2_TUNER_MODE_LANG2           0x0002
#define V4L2_TUNER_MODE_SAP             0x0002
#define V4L2_TUNER_MODE_LANG1           0x0003
#define V4L2_TUNER_MODE_LANG1_LANG2     0x0004

struct v4l2_frequency {
    __u32    tuner;
    __u32    type;    /* enum v4l2_tuner_type */
    __u32    frequency;
    __u32    reserved[8];
};

#define V4L2_BAND_MODULATION_VSB        (1 << 1)
#define V4L2_BAND_MODULATION_FM        (1 << 2)
#define V4L2_BAND_MODULATION_AM        (1 << 3)

struct v4l2_frequency_band {
    __u32    tuner;
    __u32    type;    /* enum v4l2_tuner_type */
    __u32    index;
    __u32    capability;
    __u32    rangelow;
    __u32    rangehigh;
    __u32    modulation;
    __u32    reserved[9];
};

struct v4l2_hw_freq_seek {
    __u32    tuner;
    __u32    type;    /* enum v4l2_tuner_type */
    __u32    seek_upward;
};

```

```
        __u32    wrap_around;
        __u32    spacing;
        __u32    rangelow;
        __u32    rangehigh;
        __u32    reserved[5];
};

/*
 *    R D S
 */

struct v4l2_rds_data {
    __u8    lsb;
    __u8    msb;
    __u8    block;
} __attribute__((packed));

#define V4L2_RDS_BLOCK_MSK        0x7
#define V4L2_RDS_BLOCK_A        0
#define V4L2_RDS_BLOCK_B        1
#define V4L2_RDS_BLOCK_C        2
#define V4L2_RDS_BLOCK_D        3
#define V4L2_RDS_BLOCK_C_ALT    4
#define V4L2_RDS_BLOCK_INVALID  7

#define V4L2_RDS_BLOCK_CORRECTED 0x40
#define V4L2_RDS_BLOCK_ERROR    0x80

/*
 *    A U D I O
 */
struct v4l2_audio {
    __u32    index;
    __u8     name[32];
    __u32    capability;
    __u32    mode;
    __u32    reserved[2];
};

/* Flags for the 'capability' field */
#define V4L2_AUDCAP_STEREO        0x00001
#define V4L2_AUDCAP_AVL          0x00002

/* Flags for the 'mode' field */
#define V4L2_AUDMODE_AVL          0x00001

struct v4l2_audioout {
    __u32    index;
    __u8     name[32];
    __u32    capability;
    __u32    mode;
};
```

```

    __u32 reserved[2];
};

/*
 *      M P E G      S E R V I C E S
 */
#if 1
#define V4L2_ENC_IDX_FRAME_I      (0)
#define V4L2_ENC_IDX_FRAME_P      (1)
#define V4L2_ENC_IDX_FRAME_B      (2)
#define V4L2_ENC_IDX_FRAME_MASK  (0xf)

struct v4l2_enc_idx_entry {
    __u64 offset;
    __u64 pts;
    __u32 length;
    __u32 flags;
    __u32 reserved[2];
};

#define V4L2_ENC_IDX_ENTRIES (64)
struct v4l2_enc_idx {
    __u32 entries;
    __u32 entries_cap;
    __u32 reserved[4];
    struct v4l2_enc_idx_entry entry[V4L2_ENC_IDX_ENTRIES];
};

#define V4L2_ENC_CMD_START      (0)
#define V4L2_ENC_CMD_STOP      (1)
#define V4L2_ENC_CMD_PAUSE      (2)
#define V4L2_ENC_CMD_RESUME      (3)

/* Flags for V4L2_ENC_CMD_STOP */
#define V4L2_ENC_CMD_STOP_AT_GOP_END  (1 << 0)

struct v4l2_encoder_cmd {
    __u32 cmd;
    __u32 flags;
    union {
        struct {
            __u32 data[8];
        } raw;
    };
};

/* Decoder commands */
#define V4L2_DEC_CMD_START      (0)
#define V4L2_DEC_CMD_STOP      (1)
#define V4L2_DEC_CMD_PAUSE      (2)
#define V4L2_DEC_CMD_RESUME      (3)

```

```
#define V4L2_DEC_CMD_FLUSH          (4)

/* Flags for V4L2_DEC_CMD_START */
#define V4L2_DEC_CMD_START_MUTE_AUDIO  (1 << 0)

/* Flags for V4L2_DEC_CMD_PAUSE */
#define V4L2_DEC_CMD_PAUSE_TO_BLACK  (1 << 0)

/* Flags for V4L2_DEC_CMD_STOP */
#define V4L2_DEC_CMD_STOP_TO_BLACK  (1 << 0)
#define V4L2_DEC_CMD_STOP_IMMEDIATELY  (1 << 1)

/* Play format requirements (returned by the driver): */

/* The decoder has no special format requirements */
#define V4L2_DEC_START_FMT_NONE      (0)
/* The decoder requires full GOPs */
#define V4L2_DEC_START_FMT_GOP      (1)

/* The structure must be zeroed before use by the application
   This ensures it can be extended safely in the future. */
struct v4l2_decoder_cmd {
    __u32 cmd;
    __u32 flags;
    union {
        struct {
            __u64 pts;
        } stop;

        struct {
            /* 0 or 1000 specifies normal speed,
               1 specifies forward single stepping,
               -1 specifies backward single stepping,
               >1: playback at speed/1000 of the normal speed,
               <-1: reverse playback at (-speed/1000) of the
↳normal speed. */
            __s32 speed;
            __u32 format;
        } start;

        struct {
            __u32 data[16];
        } raw;
    };
};
#endif

/*
 *   D A T A   S E R V I C E S   ( V B I )
 *
 *   Data services API by Michael Schimek

```

```

*/

/* Raw VBI */
struct v4l2_vbi_format {
    __u32    sampling_rate;        /* in 1 Hz */
    __u32    offset;
    __u32    samples_per_line;
    __u32    sample_format;       /* V4L2_PIX_FMT_* */
    __s32    start[2];
    __u32    count[2];
    __u32    flags;               /* V4L2_VBI_* */
    __u32    reserved[2];        /* must be zero */
};

/* VBI flags */
#define V4L2_VBI_UNSYNC          (1 << 0)
#define V4L2_VBI_INTERLACED    (1 << 1)

/* ITU-R start lines for each field */
#define V4L2_VBI_ITU_525_F1_START (1)
#define V4L2_VBI_ITU_525_F2_START (264)
#define V4L2_VBI_ITU_625_F1_START (1)
#define V4L2_VBI_ITU_625_F2_START (314)

/* Sliced VBI
 *
 * This implements is a proposal V4L2 API to allow SLICED VBI
 * required for some hardware encoders. It should change without
 * notice in the definitive implementation.
 */

struct v4l2_sliced_vbi_format {
    __u16    service_set;
    /* service_lines[0][...] specifies lines 0-23 (1-23 used) of the first
    ↪field
        service_lines[1][...] specifies lines 0-23 (1-23 used) of the
    ↪second field
                                (equals frame lines 313-336 for 625 line video
                                standards, 263-286 for 525 line standards) */
    __u16    service_lines[2][24];
    __u32    io_size;
    __u32    reserved[2];        /* must be zero */
};

/* Teletext World System Teletext
   (WST), defined on ITU-R BT.653-2 */
#define V4L2_SLICED_TELETEXT_B    (0x0001)
/* Video Program System, defined on ETS 300 231*/
#define V4L2_SLICED_VPS          (0x0400)
/* Closed Caption, defined on EIA-608 */
#define V4L2_SLICED_CAPTION_525  (0x1000)

```

```
/* Wide Screen System, defined on ITU-R BT1119.1 */
#define V4L2_SLICED_WSS_625                (0x4000)

#define V4L2_SLICED_VBI_525                (V4L2_SLICED_CAPTION_525)
#define V4L2_SLICED_VBI_625                (V4L2_SLICED_TELETEXT_B | V4L2_SLICED_
↪VPS | V4L2_SLICED_WSS_625)

struct v4l2_sliced_vbi_cap {
    __u16    service_set;
    /* service_lines[0][...] specifies lines 0-23 (1-23 used) of the first ↪
↪field
    service_lines[1][...] specifies lines 0-23 (1-23 used) of the ↪
↪second field
                                (equals frame lines 313-336 for 625 line video
                                standards, 263-286 for 525 line standards) */
    __u16    service_lines[2][24];
    __u32    type;          /* enum v4l2_buf_type */
    __u32    reserved[3];   /* must be 0 */
};

struct v4l2_sliced_vbi_data {
    __u32    id;
    __u32    field;        /* 0: first field, 1: second field */
    __u32    line;         /* 1-23 */
    __u32    reserved;     /* must be 0 */
    __u8     data[48];
};

/*
 * Sliced VBI data inserted into MPEG Streams
 */

/*
 * V4L2_MPEG_STREAM_VBI_FMT_IVTV:
 *
 * Structure of payload contained in an MPEG 2 Private Stream 1 PES Packet in ↪
↪an
 * MPEG-2 Program Pack that contains V4L2_MPEG_STREAM_VBI_FMT_IVTV Sliced VBI
 * data
 *
 * Note, the MPEG-2 Program Pack and Private Stream 1 PES packet header
 * definitions are not included here. See the MPEG-2 specifications for ↪
↪details
 * on these headers.
 */

/* Line type IDs */
#define V4L2_MPEG_VBI_IVTV_TELETEXT_B      (1)
#define V4L2_MPEG_VBI_IVTV_CAPTION_525    (4)
#define V4L2_MPEG_VBI_IVTV_WSS_625        (5)
#define V4L2_MPEG_VBI_IVTV_VPS            (7)
```



```

struct v4l2_mpeg_vbi_itv0_line {
    __u8 id;          /* One of V4L2_MPEG_VBI_IVTV_* above */
    __u8 data[42];   /* Sliced VBI data for the line */
} __attribute__((packed));

struct v4l2_mpeg_vbi_itv0 {
    __le32 linemask[2]; /* Bitmasks of VBI service lines present */
    struct v4l2_mpeg_vbi_itv0_line line[35];
} __attribute__((packed));

struct v4l2_mpeg_vbi_ITV0 {
    struct v4l2_mpeg_vbi_itv0_line line[36];
} __attribute__((packed));

#define V4L2_MPEG_VBI_IVTV_MAGIC0    "itv0"
#define V4L2_MPEG_VBI_IVTV_MAGIC1    "ITV0"

struct v4l2_mpeg_vbi_fmt_itv {
    __u8 magic[4];
    union {
        struct v4l2_mpeg_vbi_itv0 itv0;
        struct v4l2_mpeg_vbi_ITV0 ITV0;
    };
} __attribute__((packed));

/*
 *   A G G R E G A T E   S T R U C T U R E S
 */

/**
 * struct v4l2_plane_pix_format - additional, per-plane format definition
 * @sizeimage:          maximum size in bytes required for data, for which
 *                    this plane will be used
 * @bytesperline:      distance in bytes between the leftmost pixels in two
 *                    adjacent lines
 * @reserved:          drivers and applications must zero this array
 */
struct v4l2_plane_pix_format {
    __u32          sizeimage;
    __u32          bytesperline;
    __u16          reserved[6];
} __attribute__((packed));

/**
 * struct v4l2_pix_format_mplane - multiplanar format definition
 * @width:            image width in pixels
 * @height:           image height in pixels
 * @pixelformat:     little endian four character code (fourcc)
 * @field:           enum v4l2_field; field order (for interlaced video)
 * @colorspace:      enum v4l2_colorspace; supplemental to pixelformat

```

```
* @plane_fmt:          per-plane information
* @num_planes:        number of planes for this format
* @flags:             format flags (V4L2_PIX_FMT_FLAG_*)
* @ycbcr_enc:        enum v4l2_ycbcr_encoding, Y'CbCr encoding
* @hsv_enc:          enum v4l2_hsv_encoding, HSV encoding
* @quantization:     enum v4l2_quantization, colorspace quantization
* @xfer_func:        enum v4l2_xfer_func, colorspace transfer function
* @reserved:         drivers and applications must zero this array
*/
```

```
struct v4l2_pix_format_mplane {
    __u32                width;
    __u32                height;
    __u32                pixelformat;
    __u32                field;
    __u32                colorspace;

    struct v4l2_plane_pix_format plane_fmt[VIDEO_MAX_PLANES];
    __u8                num_planes;
    __u8                flags;
    union {
        __u8                ycbcr_enc;
        __u8                hsv_enc;
    };
    __u8                quantization;
    __u8                xfer_func;
    __u8                reserved[7];
} __attribute__((packed));
```

```
/**
 * struct v4l2_sdr_format - SDR format definition
 * @pixelformat:        little endian four character code (fourcc)
 * @buffersize:        maximum size in bytes required for data
 * @reserved:          drivers and applications must zero this array
 */
```

```
struct v4l2_sdr_format {
    __u32                pixelformat;
    __u32                buffersize;
    __u8                reserved[24];
} __attribute__((packed));
```

```
/**
 * struct v4l2_meta_format - metadata format definition
 * @dataformat:        little endian four character code (fourcc)
 * @buffersize:        maximum size in bytes required for data
 */
```

```
struct v4l2_meta_format {
    __u32                dataformat;
    __u32                buffersize;
} __attribute__((packed));
```

```
/**
```

```

* struct v4l2_format - stream data format
* @type:          enum v4l2_buf_type; type of the data stream
* @pix:          definition of an image format
* @pix_mp:       definition of a multiplanar image format
* @win:          definition of an overlaid image
* @vbi:         raw VBI capture or output parameters
* @sliced:       sliced VBI capture or output parameters
* @raw_data:     placeholder for future extensions and custom formats
* @fmt:         union of @pix, @pix_mp, @win, @vbi, @sliced, @sdr, @meta
*               and @raw_data
*/
struct v4l2_format {
    __u32    type;
    union {
        struct v4l2_pix_format      pix;      /* V4L2_BUF_TYPE_
→VIDEO_CAPTURE */
        struct v4l2_pix_format_mplane pix_mp; /* V4L2_BUF_TYPE_
→VIDEO_CAPTURE_MPLANE */
        struct v4l2_window          win;      /* V4L2_BUF_TYPE_
→VIDEO_OVERLAY */
        struct v4l2_vbi_format      vbi;      /* V4L2_BUF_TYPE_VBI_
→CAPTURE */
        struct v4l2_sliced_vbi_format sliced; /* V4L2_BUF_TYPE_
→SLICED_VBI_CAPTURE */
        struct v4l2_sdr_format      sdr;      /* V4L2_BUF_TYPE_SDR_
→CAPTURE */
        struct v4l2_meta_format     meta;     /* V4L2_BUF_TYPE_META_
→CAPTURE */
        __u8    raw_data[200]; /* user-defined */
    } fmt;
};

/*      Stream type-dependent parameters
*/
struct v4l2_streamparm {
    __u32    type; /* enum v4l2_buf_type */
    union {
        struct v4l2_captureparm capture;
        struct v4l2_outputparm  output;
        __u8    raw_data[200]; /* user-defined */
    } parm;
};

/*
 *      E V E N T S
 */

#define V4L2_EVENT_ALL          0
#define V4L2_EVENT_VSYNC       1
#define V4L2_EVENT_EOS         2
#define V4L2_EVENT_CTRL        3

```

```
#define V4L2_EVENT_FRAME_SYNC 4
#define V4L2_EVENT_SOURCE_CHANGE 5
#define V4L2_EVENT_MOTION_DET 6
#define V4L2_EVENT_PRIVATE_START 0x08000000

/* Payload for V4L2_EVENT_VSYNC */
struct v4l2_event_vsync {
    /* Can be V4L2_FIELD_ANY, _NONE, _TOP or _BOTTOM */
    __u8 field;
} __attribute__((packed));

/* Payload for V4L2_EVENT_CTRL */
#define V4L2_EVENT_CTRL_CH_VALUE (1 << 0)
#define V4L2_EVENT_CTRL_CH_FLAGS (1 << 1)
#define V4L2_EVENT_CTRL_CH_RANGE (1 << 2)

struct v4l2_event_ctrl {
    __u32 changes;
    __u32 type;
    union {
        __s32 value;
        __s64 value64;
    };
    __u32 flags;
    __s32 minimum;
    __s32 maximum;
    __s32 step;
    __s32 default_value;
};

struct v4l2_event_frame_sync {
    __u32 frame_sequence;
};

#define V4L2_EVENT_SRC_CH_RESOLUTION (1 << 0)

struct v4l2_event_src_change {
    __u32 changes;
};

#define V4L2_EVENT_MD_FL_HAVE_FRAME_SEQ (1 << 0)

/**
 * struct v4l2_event_motion_det - motion detection event
 * @flags: if V4L2_EVENT_MD_FL_HAVE_FRAME_SEQ is set, then the
 * frame_sequence field is valid.
 * @frame_sequence: the frame sequence number associated with this event.
 * @region_mask: which regions detected motion.
 */
struct v4l2_event_motion_det {
    __u32 flags;
```

```

    __u32 frame_sequence;
    __u32 region_mask;
};

struct v4l2_event {
    __u32 type;
    union {
        struct v4l2_event_vsync vsync;
        struct v4l2_event_ctrl ctrl;
        struct v4l2_event_frame_sync frame_sync;
        struct v4l2_event_src_change src_change;
        struct v4l2_event_motion_det motion_det;
        __u8 data[64];
    } u;
    __u32 pending;
    __u32 sequence;
#ifdef __KERNEL__
    struct __kernel_timespec timestamp;
#else
    struct timespec timestamp;
#endif
    __u32 id;
    __u32 reserved[8];
};

#define V4L2_EVENT_SUB_FL_SEND_INITIAL (1 << 0)
#define V4L2_EVENT_SUB_FL_ALLOW_FEEDBACK (1 << 1)

struct v4l2_event_subscription {
    __u32 type;
    __u32 id;
    __u32 flags;
    __u32 reserved[5];
};

/*
 * A D V A N C E D   D E B U G G I N G
 *
 * NOTE: EXPERIMENTAL API, NEVER RELY ON THIS IN APPLICATIONS!
 * FOR DEBUGGING, TESTING AND INTERNAL USE ONLY!
 */

/* VIDIOC_DBG_G_REGISTER and VIDIOC_DBG_S_REGISTER */

#define V4L2_CHIP_MATCH_BRIDGE 0 /* Match against chip ID on the bridge,
→ (0 for the bridge) */
#define V4L2_CHIP_MATCH_SUBDEV 4 /* Match against subdev index */

/* The following four defines are no longer in use */
#define V4L2_CHIP_MATCH_HOST V4L2_CHIP_MATCH_BRIDGE
#define V4L2_CHIP_MATCH_I2C_DRIVER 1 /* Match against I2C driver name */

```

```
#define V4L2_CHIP_MATCH_I2C_ADDR 2 /* Match against I2C 7-bit address */
#define V4L2_CHIP_MATCH_AC97 3 /* Match against ancillary AC97 chip */

struct v4l2_dbg_match {
    __u32 type; /* Match type */
    union { /* Match this chip, meaning determined by type */
        __u32 addr;
        char name[32];
    };
} __attribute__((packed));

struct v4l2_dbg_register {
    struct v4l2_dbg_match match;
    __u32 size; /* register size in bytes */
    __u64 reg;
    __u64 val;
} __attribute__((packed));

#define V4L2_CHIP_FL_READABLE (1 << 0)
#define V4L2_CHIP_FL_WRITABLE (1 << 1)

/* VIDIOC_DBG_G_CHIP_INFO */
struct v4l2_dbg_chip_info {
    struct v4l2_dbg_match match;
    char name[32];
    __u32 flags;
    __u32 reserved[32];
} __attribute__((packed));

/**
 * struct v4l2_create_buffers - VIDIOC_CREATE_BUFS argument
 * @index: on return, index of the first created buffer
 * @count: entry: number of requested buffers,
 *         return: number of created buffers
 * @memory: enum v4l2_memory; buffer memory type
 * @format: frame format, for which buffers are requested
 * @capabilities: capabilities of this buffer type.
 * @flags: additional buffer management attributes (ignored unless the
 *         queue has V4L2_BUF_CAP_SUPPORTS_MMAP_CACHE_HINTS capability
 *         and configured for MMAP streaming I/O).
 * @reserved: future extensions
 */
struct v4l2_create_buffers {
    __u32 index;
    __u32 count;
    __u32 memory;
    struct v4l2_format format;
    __u32 capabilities;
    __u32 flags;
    __u32 reserved[6];
};
```

```

/*
 *      I O C T L   C O D E S   F O R   V I D E O   D E V I C E S
 *
 */
#define VIDIOC_QUERYCAP          _IOR('V', 0, struct v4l2_capability)
#define VIDIOC_ENUM_FMT         _IOWR('V', 2, struct v4l2_fmtdesc)
#define VIDIOC_G_FMT           _IOWR('V', 4, struct v4l2_format)
#define VIDIOC_S_FMT           _IOWR('V', 5, struct v4l2_format)
#define VIDIOC_REQBUFS         _IOWR('V', 8, struct v4l2_requestbuffers)
#define VIDIOC_QUERYBUF        _IOWR('V', 9, struct v4l2_buffer)
#define VIDIOC_G_FBUF          _IOR('V', 10, struct v4l2_framebuffer)
#define VIDIOC_S_FBUF          _IOW('V', 11, struct v4l2_framebuffer)
#define VIDIOC_OVERLAY         _IOW('V', 14, int)
#define VIDIOC_QBUF            _IOWR('V', 15, struct v4l2_buffer)
#define VIDIOC_EXPBUF          _IOWR('V', 16, struct v4l2_exportbuffer)
#define VIDIOC_DQBUF           _IOWR('V', 17, struct v4l2_buffer)
#define VIDIOC_STREAMON        _IOW('V', 18, int)
#define VIDIOC_STREAMOFF       _IOW('V', 19, int)
#define VIDIOC_G_PARM          _IOWR('V', 21, struct v4l2_streamparm)
#define VIDIOC_S_PARM          _IOWR('V', 22, struct v4l2_streamparm)
#define VIDIOC_G_STD           _IOR('V', 23, v4l2_std_id)
#define VIDIOC_S_STD           _IOW('V', 24, v4l2_std_id)
#define VIDIOC_ENUMSTD         _IOWR('V', 25, struct v4l2_standard)
#define VIDIOC_ENUMINPUT       _IOWR('V', 26, struct v4l2_input)
#define VIDIOC_G_CTRL          _IOWR('V', 27, struct v4l2_control)
#define VIDIOC_S_CTRL          _IOWR('V', 28, struct v4l2_control)
#define VIDIOC_G_TUNER         _IOWR('V', 29, struct v4l2_tuner)
#define VIDIOC_S_TUNER         _IOW('V', 30, struct v4l2_tuner)
#define VIDIOC_G_AUDIO         _IOR('V', 33, struct v4l2_audio)
#define VIDIOC_S_AUDIO         _IOW('V', 34, struct v4l2_audio)
#define VIDIOC_QUERYCTRL       _IOWR('V', 36, struct v4l2_queryctrl)
#define VIDIOC_QUERYMENU       _IOWR('V', 37, struct v4l2_querymenu)
#define VIDIOC_G_INPUT         _IOR('V', 38, int)
#define VIDIOC_S_INPUT         _IOWR('V', 39, int)
#define VIDIOC_G_EDID          _IOWR('V', 40, struct v4l2_edid)
#define VIDIOC_S_EDID          _IOWR('V', 41, struct v4l2_edid)
#define VIDIOC_G_OUTPUT        _IOR('V', 46, int)
#define VIDIOC_S_OUTPUT        _IOWR('V', 47, int)
#define VIDIOC_ENUMOUTPUT      _IOWR('V', 48, struct v4l2_output)
#define VIDIOC_G_AUDOUT        _IOR('V', 49, struct v4l2_audioout)
#define VIDIOC_S_AUDOUT        _IOW('V', 50, struct v4l2_audioout)
#define VIDIOC_G_MODULATOR    _IOWR('V', 54, struct v4l2_modulator)
#define VIDIOC_S_MODULATOR    _IOW('V', 55, struct v4l2_modulator)
#define VIDIOC_G_FREQUENCY     _IOWR('V', 56, struct v4l2_frequency)
#define VIDIOC_S_FREQUENCY     _IOW('V', 57, struct v4l2_frequency)
#define VIDIOC_CROPCAP         _IOWR('V', 58, struct v4l2_cropcap)
#define VIDIOC_G_CROP          _IOWR('V', 59, struct v4l2_crop)
#define VIDIOC_S_CROP          _IOW('V', 60, struct v4l2_crop)
#define VIDIOC_G_JPEGCOMP       _IOR('V', 61, struct v4l2_jpegcompression)
#define VIDIOC_S_JPEGCOMP       _IOW('V', 62, struct v4l2_jpegcompression)

```



```
#define VIDIOC_QUERYSTD          _IOR('V', 63, v4l2_std_id)
#define VIDIOC_TRY_FMT          _IOWR('V', 64, struct v4l2_format)
#define VIDIOC_ENUMAUDIO        _IOWR('V', 65, struct v4l2_audio)
#define VIDIOC_ENUMAUDOUT      _IOWR('V', 66, struct v4l2_audioout)
#define VIDIOC_G_PRIORITY      _IOR('V', 67, __u32) /* enum v4l2_priority */
#define VIDIOC_S_PRIORITY      _IOW('V', 68, __u32) /* enum v4l2_priority */
#define VIDIOC_G_SLICED_VBI_CAP _IOWR('V', 69, struct v4l2_sliced_vbi_cap)
#define VIDIOC_LOG_STATUS      _IO('V', 70)
#define VIDIOC_G_EXT_CTRLS     _IOWR('V', 71, struct v4l2_ext_controls)
#define VIDIOC_S_EXT_CTRLS     _IOWR('V', 72, struct v4l2_ext_controls)
#define VIDIOC_TRY_EXT_CTRLS   _IOWR('V', 73, struct v4l2_ext_controls)
#define VIDIOC_ENUM_FRAMESIZES _IOWR('V', 74, struct v4l2_frmsizeenum)
#define VIDIOC_ENUM_FRAMEINTERVALS _IOWR('V', 75, struct v4l2_fmivalenum)
#define VIDIOC_G_ENC_INDEX     _IOR('V', 76, struct v4l2_enc_idx)
#define VIDIOC_ENCODER_CMD     _IOWR('V', 77, struct v4l2_encoder_cmd)
#define VIDIOC_TRY_ENCODER_CMD _IOWR('V', 78, struct v4l2_encoder_cmd)

/*
 * Experimental, meant for debugging, testing and internal use.
 * Only implemented if CONFIG_VIDEO_ADV_DEBUG is defined.
 * You must be root to use these ioctls. Never use these in applications!
 */
#define VIDIOC_DBG_S_REGISTER  _IOW('V', 79, struct v4l2_dbg_register)
#define VIDIOC_DBG_G_REGISTER  _IOWR('V', 80, struct v4l2_dbg_register)

#define VIDIOC_S_HW_FREQ_SEEK  _IOW('V', 82, struct v4l2_hw_freq_seek)
#define VIDIOC_S_DV_TIMINGS    _IOWR('V', 87, struct v4l2_dv_timings)
#define VIDIOC_G_DV_TIMINGS    _IOWR('V', 88, struct v4l2_dv_timings)
#define VIDIOC_DQEVENT        _IOR('V', 89, struct v4l2_event)
#define VIDIOC_SUBSCRIBE_EVENT _IOW('V', 90, struct v4l2_event_subscription)
#define VIDIOC_UNSUBSCRIBE_EVENT _IOW('V', 91, struct v4l2_event_subscription)
#define VIDIOC_CREATE_BUFS     _IOWR('V', 92, struct v4l2_create_buffers)
#define VIDIOC_PREPARE_BUF     _IOWR('V', 93, struct v4l2_buffer)
#define VIDIOC_G_SELECTION     _IOWR('V', 94, struct v4l2_selection)
#define VIDIOC_S_SELECTION     _IOWR('V', 95, struct v4l2_selection)
#define VIDIOC_DECODER_CMD     _IOWR('V', 96, struct v4l2_decoder_cmd)
#define VIDIOC_TRY_DECODER_CMD _IOWR('V', 97, struct v4l2_decoder_cmd)
#define VIDIOC_ENUM_DV_TIMINGS _IOWR('V', 98, struct v4l2_enum_dv_timings)
#define VIDIOC_QUERY_DV_TIMINGS _IOR('V', 99, struct v4l2_dv_timings)
#define VIDIOC_DV_TIMINGS_CAP  _IOWR('V', 100, struct v4l2_dv_timings_cap)
#define VIDIOC_ENUM_FREQ_BANDS _IOWR('V', 101, struct v4l2_frequency_band)

/*
 * Experimental, meant for debugging, testing and internal use.
 * Never use this in applications!
 */
#define VIDIOC_DBG_G_CHIP_INFO  _IOWR('V', 102, struct v4l2_dbg_chip_info)

#define VIDIOC_QUERY_EXT_CTRL   _IOWR('V', 103, struct v4l2_query_ext_ctrl)

/* Reminder: when adding new ioctls please add support for them to
```



```

drivers/media/v4l2-core/v4l2-compat-ioctl32.c as well! */

#define BASE_VIDIIOC_PRIVATE      192                /* 192-255 are private */

/* Deprecated definitions kept for backwards compatibility */
#ifndef __KERNEL__
#define V4L2_PIX_FMT_HM12 V4L2_PIX_FMT_NV12_16L16
#define V4L2_PIX_FMT_SUNXI_TILED_NV12 V4L2_PIX_FMT_NV12_32L32
#endif

#endif /* __UAPI__LINUX_VIDEODEV2_H */

```

3.2.10 Video Capture Example

3.2.10.1 file: media/v4l/capture.c

```

/*
 * V4L2 video capture example
 *
 * This program can be used and distributed without restrictions.
 *
 * This program is provided with the V4L2 API
 * see https://linuxtv.org/docs.php for more information
 */

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <assert.h>

#include <getopt.h>          /* getopt_long() */

#include <fcntl.h>          /* low-level i/o */
#include <unistd.h>
#include <errno.h>
#include <sys/stat.h>
#include <sys/types.h>
#include <sys/time.h>
#include <sys/mman.h>
#include <sys/ioctl.h>

#include <linux/videodev2.h>

#define CLEAR(x) memset(&(x), 0, sizeof(x))

enum io_method {
    IO_METHOD_READ,
    IO_METHOD_MMAP,
    IO_METHOD_USERPTR,
};

struct buffer {
    void *start;
    size_t length;
};

```

```
static char      *dev_name;
static enum io_method io = IO_METHOD_MMAP;
static int      fd = -1;
static struct buffer *buffers;
static unsigned int n_buffers;
static int      out_buf;
static int      force_format;
static int      frame_count = 70;

static void errno_exit(const char *s)
{
    fprintf(stderr, "%s error %d, %s\n", s, errno, strerror(errno));
    exit(EXIT_FAILURE);
}

static int xioctl(int fh, int request, void *arg)
{
    int r;

    do {
        r = ioctl(fh, request, arg);
    } while (-1 == r && EINTR == errno);

    return r;
}

static void process_image(const void *p, int size)
{
    if (out_buf)
        fwrite(p, size, 1, stdout);

    fflush(stderr);
    fprintf(stderr, ".");
    fflush(stdout);
}

static int read_frame(void)
{
    struct v4l2_buffer buf;
    unsigned int i;

    switch (io) {
    case IO_METHOD_READ:
        if (-1 == read(fd, buffers[0].start, buffers[0].length)) {
            switch (errno) {
            case EAGAIN:
                return 0;

            case EIO:
                /* Could ignore EIO, see spec. */

                /* fall through */

            default:
                errno_exit("read");
            }
        }
    }
```

```

    }

    process_image(buf[0].start, buf[0].length);
    break;

case IO_METHOD_MMAP:
    CLEAR(buf);

    buf.type = V4L2_BUF_TYPE_VIDEO_CAPTURE;
    buf.memory = V4L2_MEMORY_MMAP;

    if (-1 == xioctl(fd, VIDIOC_DQBUF, &buf)) {
        switch (errno) {
            case EAGAIN:
                return 0;

            case EIO:
                /* Could ignore EIO, see spec. */

                /* fall through */

            default:
                errno_exit("VIDIOC_DQBUF");
        }
    }

    assert(buf.index < n_buffers);

    process_image(buf.index.start, buf.bytesused);

    if (-1 == xioctl(fd, VIDIOC_QBUF, &buf))
        errno_exit("VIDIOC_QBUF");
    break;

case IO_METHOD_USERPTR:
    CLEAR(buf);

    buf.type = V4L2_BUF_TYPE_VIDEO_CAPTURE;
    buf.memory = V4L2_MEMORY_USERPTR;

    if (-1 == xioctl(fd, VIDIOC_DQBUF, &buf)) {
        switch (errno) {
            case EAGAIN:
                return 0;

            case EIO:
                /* Could ignore EIO, see spec. */

                /* fall through */

            default:
                errno_exit("VIDIOC_DQBUF");
        }
    }

    for (i = 0; i < n_buffers; ++i)
        if (buf.m.userptr == (unsigned long)buf[i].start

```

```
        && buf.length == buffers[i].length)
            break;

    assert(i < n_buffers);

    process_image((void *)buf.m.userptr, buf.bytesused);

    if (-1 == xiocctl(fd, VIDIOC_QBUF, &buf))
        errno_exit("VIDIOC_QBUF");
    break;
}

return 1;
}

static void mainloop(void)
{
    unsigned int count;

    count = frame_count;

    while (count-- > 0) {
        for (;;) {
            fd_set fds;
            struct timeval tv;
            int r;

            FD_ZERO(&fds);
            FD_SET(fd, &fds);

            /* Timeout. */
            tv.tv_sec = 2;
            tv.tv_usec = 0;

            r = select(fd + 1, &fds, NULL, NULL, &tv);

            if (-1 == r) {
                if (EINTR == errno)
                    continue;
                errno_exit("select");
            }

            if (0 == r) {
                fprintf(stderr, "select timeout\n");
                exit(EXIT_FAILURE);
            }

            if (read_frame())
                break;
            /* EAGAIN - continue select loop. */
        }
    }
}

static void stop_capturing(void)
{
    enum v4l2_buf_type type;
```

```

switch (io) {
case IO_METHOD_READ:
    /* Nothing to do. */
    break;

case IO_METHOD_MMAP:
case IO_METHOD_USERPTR:
    type = V4L2_BUF_TYPE_VIDEO_CAPTURE;
    if (-1 == xioctl(fd, VIDIOC_STREAMOFF, &type))
        errno_exit("VIDIOC_STREAMOFF");
    break;
}
}

static void start_capturing(void)
{
    unsigned int i;
    enum v4l2_buf_type type;

    switch (io) {
case IO_METHOD_READ:
    /* Nothing to do. */
    break;

case IO_METHOD_MMAP:
    for (i = 0; i < n_buffers; ++i) {
        struct v4l2_buffer buf;

        CLEAR(buf);
        buf.type = V4L2_BUF_TYPE_VIDEO_CAPTURE;
        buf.memory = V4L2_MEMORY_MMAP;
        buf.index = i;

        if (-1 == xioctl(fd, VIDIOC_QBUF, &buf))
            errno_exit("VIDIOC_QBUF");
    }
    type = V4L2_BUF_TYPE_VIDEO_CAPTURE;
    if (-1 == xioctl(fd, VIDIOC_STREAMON, &type))
        errno_exit("VIDIOC_STREAMON");
    break;

case IO_METHOD_USERPTR:
    for (i = 0; i < n_buffers; ++i) {
        struct v4l2_buffer buf;

        CLEAR(buf);
        buf.type = V4L2_BUF_TYPE_VIDEO_CAPTURE;
        buf.memory = V4L2_MEMORY_USERPTR;
        buf.index = i;
        buf.m.userptr = (unsigned long)buffers[i].start;
        buf.length = buffers[i].length;

        if (-1 == xioctl(fd, VIDIOC_QBUF, &buf))
            errno_exit("VIDIOC_QBUF");
    }
    type = V4L2_BUF_TYPE_VIDEO_CAPTURE;

```

```
        if (-1 == xioctl(fd, VIDIOC_STREAMON, &type))
            errno_exit("VIDIOC_STREAMON");
        break;
    }
}

static void uninit_device(void)
{
    unsigned int i;

    switch (io) {
    case IO_METHOD_READ:
        free(buffers[0].start);
        break;

    case IO_METHOD_MMAP:
        for (i = 0; i < n_buffers; ++i)
            if (-1 == munmap(buffers[i].start, buffers[i].length))
                errno_exit("munmap");
        break;

    case IO_METHOD_USERPTR:
        for (i = 0; i < n_buffers; ++i)
            free(buffers[i].start);
        break;
    }

    free(buffers);
}

static void init_read(unsigned int buffer_size)
{
    buffers = calloc(1, sizeof(*buffers));

    if (!buffers) {
        fprintf(stderr, "Out of memory\n");
        exit(EXIT_FAILURE);
    }

    buffers[0].length = buffer_size;
    buffers[0].start = malloc(buffer_size);

    if (!buffers[0].start) {
        fprintf(stderr, "Out of memory\n");
        exit(EXIT_FAILURE);
    }
}

static void init_mmap(void)
{
    struct v4l2_requestbuffers req;

    CLEAR(req);

    req.count = 4;
    req.type = V4L2_BUF_TYPE_VIDEO_CAPTURE;
    req.memory = V4L2_MEMORY_MMAP;
}
```

```

if (-1 == xioctl(fd, VIDIOC_REQBUFS, &req)) {
    if (EINVAL == errno) {
        fprintf(stderr, "%s does not support "
                "memory mapping", dev_name);
        exit(EXIT_FAILURE);
    } else {
        errno_exit("VIDIOC_REQBUFS");
    }
}

if (req.count < 2) {
    fprintf(stderr, "Insufficient buffer memory on %s\n",
            dev_name);
    exit(EXIT_FAILURE);
}

buffers = calloc(req.count, sizeof(*buffers));

if (!buffers) {
    fprintf(stderr, "Out of memory\n");
    exit(EXIT_FAILURE);
}

for (n_buffers = 0; n_buffers < req.count; ++n_buffers) {
    struct v4l2_buffer buf;

    CLEAR(buf);

    buf.type      = V4L2_BUF_TYPE_VIDEO_CAPTURE;
    buf.memory    = V4L2_MEMORY_MMAP;
    buf.index     = n_buffers;

    if (-1 == xioctl(fd, VIDIOC_QUERYBUF, &buf))
        errno_exit("VIDIOC_QUERYBUF");

    buffers[n_buffers].length = buf.length;
    buffers[n_buffers].start =
        mmap(NULL /* start anywhere */,
            buf.length,
            PROT_READ | PROT_WRITE /* required */,
            MAP_SHARED /* recommended */,
            fd, buf.m.offset);

    if (MAP_FAILED == buffers[n_buffers].start)
        errno_exit("mmap");
}
}

static void init_userp(unsigned int buffer_size)
{
    struct v4l2_requestbuffers req;

    CLEAR(req);

    req.count = 4;
    req.type = V4L2_BUF_TYPE_VIDEO_CAPTURE;

```

```
req.memory = V4L2_MEMORY_USERPTR;

if (-1 == xiocctl(fd, VIDIOC_REQBUFS, &req)) {
    if (EINVAL == errno) {
        fprintf(stderr, "%s does not support "
                "user pointer i/on", dev_name);
        exit(EXIT_FAILURE);
    } else {
        errno_exit("VIDIOC_REQBUFS");
    }
}

buffers = calloc(4, sizeof(*buffers));

if (!buffers) {
    fprintf(stderr, "Out of memory\n");
    exit(EXIT_FAILURE);
}

for (n_buffers = 0; n_buffers < 4; ++n_buffers) {
    buffers[n_buffers].length = buffer_size;
    buffers[n_buffers].start = malloc(buffer_size);

    if (!buffers[n_buffers].start) {
        fprintf(stderr, "Out of memory\n");
        exit(EXIT_FAILURE);
    }
}
}

static void init_device(void)
{
    struct v4l2_capability cap;
    struct v4l2_cropcap cropcap;
    struct v4l2_crop crop;
    struct v4l2_format fmt;
    unsigned int min;

    if (-1 == xiocctl(fd, VIDIOC_QUERYCAP, &cap)) {
        if (EINVAL == errno) {
            fprintf(stderr, "%s is no V4L2 device\n",
                    dev_name);
            exit(EXIT_FAILURE);
        } else {
            errno_exit("VIDIOC_QUERYCAP");
        }
    }

    if (!(cap.capabilities & V4L2_CAP_VIDEO_CAPTURE)) {
        fprintf(stderr, "%s is no video capture device\n",
                dev_name);
        exit(EXIT_FAILURE);
    }

    switch (io) {
    case IO_METHOD_READ:
        if (!(cap.capabilities & V4L2_CAP_READWRITE)) {
```



```

        fprintf(stderr, "%s does not support read i/o\n",
                dev_name);
        exit(EXIT_FAILURE);
    }
    break;

case IO_METHOD_MMAP:
case IO_METHOD_USERPTR:
    if (!(cap.capabilities & V4L2_CAP_STREAMING)) {
        fprintf(stderr, "%s does not support streaming i/o\n",
                dev_name);
        exit(EXIT_FAILURE);
    }
    break;
}

/* Select video input, video standard and tune here. */

CLEAR(cropcap);

cropcap.type = V4L2_BUF_TYPE_VIDEO_CAPTURE;

if (0 == xiocctl(fd, VIDIOC_CROPCAP, &cropcap)) {
    crop.type = V4L2_BUF_TYPE_VIDEO_CAPTURE;
    crop.c = cropcap.defrect; /* reset to default */

    if (-1 == xiocctl(fd, VIDIOC_S_CROP, &crop)) {
        switch (errno) {
            case EINVAL:
                /* Cropping not supported. */
                break;
            default:
                /* Errors ignored. */
                break;
        }
    }
} else {
    /* Errors ignored. */
}

CLEAR(fmt);

fmt.type = V4L2_BUF_TYPE_VIDEO_CAPTURE;
if (force_format) {
    fmt.fmt.pix.width      = 640;
    fmt.fmt.pix.height    = 480;
    fmt.fmt.pix.pixelformat = V4L2_PIX_FMT_YUYV;
    fmt.fmt.pix.field      = V4L2_FIELD_INTERLACED;

    if (-1 == xiocctl(fd, VIDIOC_S_FMT, &fmt))
        errno_exit("VIDIOC_S_FMT");

    /* Note VIDIOC_S_FMT may change width and height. */
} else {

```

```
        /* Preserve original settings as set by v4l2-ctl for example */
        if (-1 == xiocctl(fd, VIDIOC_G_FMT, &fmt))
            errno_exit("VIDIOC_G_FMT");
    }

    /* Buggy driver paranoia. */
    min = fmt.fmt.pix.width * 2;
    if (fmt.fmt.pix.bytesperline < min)
        fmt.fmt.pix.bytesperline = min;
    min = fmt.fmt.pix.bytesperline * fmt.fmt.pix.height;
    if (fmt.fmt.pix.sizeimage < min)
        fmt.fmt.pix.sizeimage = min;

    switch (io) {
    case IO_METHOD_READ:
        init_read(fmt.fmt.pix.sizeimage);
        break;

    case IO_METHOD_MMAP:
        init_mmap();
        break;

    case IO_METHOD_USERPTR:
        init_userp(fmt.fmt.pix.sizeimage);
        break;
    }
}

static void close_device(void)
{
    if (-1 == close(fd))
        errno_exit("close");

    fd = -1;
}

static void open_device(void)
{
    struct stat st;

    if (-1 == stat(dev_name, &st)) {
        fprintf(stderr, "Cannot identify '%s': %d, %s\n",
                dev_name, errno, strerror(errno));
        exit(EXIT_FAILURE);
    }

    if (!S_ISCHR(st.st_mode)) {
        fprintf(stderr, "%s is no devicen", dev_name);
        exit(EXIT_FAILURE);
    }

    fd = open(dev_name, O_RDWR /* required */ | O_NONBLOCK, 0);

    if (-1 == fd) {
        fprintf(stderr, "Cannot open '%s': %d, %s\n",
                dev_name, errno, strerror(errno));
        exit(EXIT_FAILURE);
    }
}
```

```

    }
}

static void usage(FILE *fp, int argc, char **argv)
{
    fprintf(fp,
        "Usage: %s [options]\n\n"
        "Version 1.3\n"
        "Options:\n"
        "-d | --device name   Video device name [%s]\n"
        "-h | --help         Print this message\n"
        "-m | --mmap         Use memory mapped buffers [default]\n"
        "-r | --read         Use read() calls\n"
        "-u | --userp        Use application allocated buffers\n"
        "-o | --output        Outputs stream to stdout\n"
        "-f | --format        Force format to 640x480 YUYV\n"
        "-c | --count        Number of frames to grab [%i]\n"
        "",
        argv[0], dev_name, frame_count);
}

static const char short_options[] = "d:hmrufc:";

static const struct option
long_options[] = {
    { "device", required_argument, NULL, 'd' },
    { "help", no_argument, NULL, 'h' },
    { "mmap", no_argument, NULL, 'm' },
    { "read", no_argument, NULL, 'r' },
    { "userp", no_argument, NULL, 'u' },
    { "output", no_argument, NULL, 'o' },
    { "format", no_argument, NULL, 'f' },
    { "count", required_argument, NULL, 'c' },
    { 0, 0, 0, 0 }
};

int main(int argc, char **argv)
{
    dev_name = "/dev/video0";

    for (;;) {
        int idx;
        int c;

        c = getopt_long(argc, argv,
            short_options, long_options, &idx);

        if (-1 == c)
            break;

        switch (c) {
        case 0: /* getopt_long() flag */
            break;

        case 'd':
            dev_name = optarg;
            break;
    }
}

```

```
    case 'h':
        usage(stdout, argc, argv);
        exit(EXIT_SUCCESS);

    case 'm':
        io = IO_METHOD_MMAP;
        break;

    case 'r':
        io = IO_METHOD_READ;
        break;

    case 'u':
        io = IO_METHOD_USERPTR;
        break;

    case 'o':
        out_buf++;
        break;

    case 'f':
        force_format++;
        break;

    case 'c':
        errno = 0;
        frame_count = strtol(optarg, NULL, 0);
        if (errno)
            errno_exit(optarg);
        break;

    default:
        usage(stderr, argc, argv);
        exit(EXIT_FAILURE);
}

open_device();
init_device();
start_capturing();
mainloop();
stop_capturing();
uninit_device();
close_device();
fprintf(stderr, "\n");
return 0;
}
```

3.2.11 Video Grabber example using libv4l

This program demonstrates how to grab V4L2 images in ppm format by using libv4l handlers. The advantage is that this grabber can potentially work with any V4L2 driver.

3.2.11.1 file: media/v4l/v4l2grab.c

```

/* V4L2 video picture grabber
   Copyright (C) 2009 Mauro Carvalho Chehab <mchehab@kernel.org>

   This program is free software; you can redistribute it and/or modify
   it under the terms of the GNU General Public License as published by
   the Free Software Foundation version 2 of the License.

   This program is distributed in the hope that it will be useful,
   but WITHOUT ANY WARRANTY; without even the implied warranty of
   MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
   GNU General Public License for more details.
*/

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <fcntl.h>
#include <errno.h>
#include <sys/ioctl.h>
#include <sys/types.h>
#include <sys/time.h>
#include <sys/mman.h>
#include <linux/videodev2.h>
#include "../libv4l/include/libv4l2.h"

#define CLEAR(x) memset(&(x), 0, sizeof(x))

struct buffer {
    void *start;
    size_t length;
};

static void xioctl(int fh, int request, void *arg)
{
    int r;

    do {
        r = v4l2_ioctl(fh, request, arg);
    } while (r == -1 && ((errno == EINTR) || (errno == EAGAIN)));

    if (r == -1) {
        fprintf(stderr, "error %d, %s\n", errno, strerror(errno));
        exit(EXIT_FAILURE);
    }
}

int main(int argc, char **argv)
{
    struct v4l2_format          fmt;

```

```

struct v4l2_buffer          buf;
struct v4l2_requestbuffers req;
enum v4l2_buf_type         type;
fd_set                    fds;
struct timeval             tv;
int                        r, fd = -1;
unsigned int              i, n_buffers;
char                      *dev_name = "/dev/video0";
char                      out_name[256];
FILE                      *fout;
struct buffer              *buffers;

fd = v4l2_open(dev_name, O_RDWR | O_NONBLOCK, 0);
if (fd < 0) {
    perror("Cannot open device");
    exit(EXIT_FAILURE);
}

CLEAR(fmt);
fmt.type = V4L2_BUF_TYPE_VIDEO_CAPTURE;
fmt.fmt.pix.width      = 640;
fmt.fmt.pix.height     = 480;
fmt.fmt.pix.pixelformat = V4L2_PIX_FMT_RGB24;
fmt.fmt.pix.field      = V4L2_FIELD_INTERLACED;
ioctl(fd, VIDIOC_S_FMT, &fmt);
if (fmt.fmt.pix.pixelformat != V4L2_PIX_FMT_RGB24) {
    printf("Libv4l didn't accept RGB24 format. Can't proceed.\n");
    exit(EXIT_FAILURE);
}
if ((fmt.fmt.pix.width != 640) || (fmt.fmt.pix.height != 480))
    printf("Warning: driver is sending image at %dx%d\n",
           fmt.fmt.pix.width, fmt.fmt.pix.height);

CLEAR(req);
req.count = 2;
req.type = V4L2_BUF_TYPE_VIDEO_CAPTURE;
req.memory = V4L2_MEMORY_MMAP;
ioctl(fd, VIDIOC_REQBUFS, &req);

buffers = calloc(req.count, sizeof(*buffers));
for (n_buffers = 0; n_buffers < req.count; ++n_buffers) {
    CLEAR(buf);

    buf.type      = V4L2_BUF_TYPE_VIDEO_CAPTURE;
    buf.memory    = V4L2_MEMORY_MMAP;
    buf.index     = n_buffers;

    ioctl(fd, VIDIOC_QUERYBUF, &buf);

    buffers[n_buffers].length = buf.length;
    buffers[n_buffers].start = v4l2_mmap(NULL, buf.length,
                                         PROT_READ | PROT_WRITE, MAP_SHARED,
                                         fd, buf.m.offset);

    if (MAP_FAILED == buffers[n_buffers].start) {
        perror("mmap");
        exit(EXIT_FAILURE);
    }
}

```

```

    }
}

for (i = 0; i < n_buffers; ++i) {
    CLEAR(buf);
    buf.type = V4L2_BUF_TYPE_VIDEO_CAPTURE;
    buf.memory = V4L2_MEMORY_MMAP;
    buf.index = i;
    xioctl(fd, VIDIOC_QBUF, &buf);
}
type = V4L2_BUF_TYPE_VIDEO_CAPTURE;
xioctl(fd, VIDIOC_STREAMON, &type);
for (i = 0; i < 20; i++) {
    do {
        FD_ZERO(&fds);
        FD_SET(fd, &fds);

        /* Timeout. */
        tv.tv_sec = 2;
        tv.tv_usec = 0;

        r = select(fd + 1, &fds, NULL, NULL, &tv);
    } while ((r == -1 && (errno = EINTR)));
    if (r == -1) {
        perror("select");
        return errno;
    }

    CLEAR(buf);
    buf.type = V4L2_BUF_TYPE_VIDEO_CAPTURE;
    buf.memory = V4L2_MEMORY_MMAP;
    xioctl(fd, VIDIOC_DQBUF, &buf);

    sprintf(out_name, "out%03d.ppm", i);
    fout = fopen(out_name, "w");
    if (!fout) {
        perror("Cannot open image");
        exit(EXIT_FAILURE);
    }
    fprintf(fout, "P6\n%d %d 255\n",
            fmt.fmt.pix.width, fmt.fmt.pix.height);
    fwrite(bufs[buf.index].start, buf.bytesused, 1, fout);
    fclose(fout);

    xioctl(fd, VIDIOC_QBUF, &buf);
}

type = V4L2_BUF_TYPE_VIDEO_CAPTURE;
xioctl(fd, VIDIOC_STREAMOFF, &type);
for (i = 0; i < n_buffers; ++i)
    v4l2_munmap(bufs[i].start, buf[i].length);
v4l2_close(fd);

return 0;
}

```

3.2.12 References

3.2.12.1 CEA 608-E

title CEA-608-E R-2014 “Line 21 Data Services”

author Consumer Electronics Association (<http://www.ce.org>)

3.2.12.2 EN 300 294

title EN 300 294 “625-line television Wide Screen Signalling (WSS)”

author European Telecommunication Standards Institute (<http://www.etsi.org>)

3.2.12.3 ETS 300 231

title ETS 300 231 “Specification of the domestic video Programme Delivery Control system (PDC)”

author European Telecommunication Standards Institute (<http://www.etsi.org>)

3.2.12.4 ETS 300 706

title ETS 300 706 “Enhanced Teletext specification”

author European Telecommunication Standards Institute (<http://www.etsi.org>)

3.2.12.5 ISO 13818-1

title ITU-T Rec. H.222.0 | ISO/IEC 13818-1 “Information technology — Generic coding of moving pictures and associated audio information: Systems”

author International Telecommunication Union (<http://www.itu.ch>), International Organisation for Standardisation (<http://www.iso.ch>)

3.2.12.6 ISO 13818-2

title ITU-T Rec. H.262 | ISO/IEC 13818-2 “Information technology — Generic coding of moving pictures and associated audio information: Video”

author International Telecommunication Union (<http://www.itu.ch>), International Organisation for Standardisation (<http://www.iso.ch>)

3.2.12.7 ITU BT.470

title ITU-R Recommendation BT.470-6 “Conventional Television Systems”

author International Telecommunication Union (<http://www.itu.ch>)

3.2.12.8 ITU BT.601

title ITU-R Recommendation BT.601-5 “Studio Encoding Parameters of Digital Television for Standard 4:3 and Wide-Screen 16:9 Aspect Ratios”

author International Telecommunication Union (<http://www.itu.ch>)

3.2.12.9 ITU BT.653

title ITU-R Recommendation BT.653-3 “Teletext systems”

author International Telecommunication Union (<http://www.itu.ch>)

3.2.12.10 ITU BT.709

title ITU-R Recommendation BT.709-5 “Parameter values for the HDTV standards for production and international programme exchange”

author International Telecommunication Union (<http://www.itu.ch>)

3.2.12.11 ITU BT.1119

title ITU-R Recommendation BT.1119 “625-line television Wide Screen Signalling (WSS)”

author International Telecommunication Union (<http://www.itu.ch>)

3.2.12.12 ITU-T Rec. H.264 Specification (04/2017 Edition)

title ITU-T Recommendation H.264 “Advanced Video Coding for Generic Audiovisual Services”

author International Telecommunication Union (<http://www.itu.ch>)

3.2.12.13 ITU H.265/HEVC

title ITU-T Rec. H.265 | ISO/IEC 23008-2 “High Efficiency Video Coding”

author International Telecommunication Union (<http://www.itu.ch>), International Organisation for Standardisation (<http://www.iso.ch>)

3.2.12.14 JFIF

title JPEG File Interchange Format

subtitle Version 1.02

author Independent JPEG Group (<http://www.ijg.org>)

3.2.12.15 ITU-T.81

title ITU-T Recommendation T.81 “Information Technology — Digital Compression and Coding of Continuous-Tone Still Images — Requirements and Guidelines”

author International Telecommunication Union (<http://www.itu.int>)

3.2.12.16 W3C JPEG JFIF

title JPEG JFIF

author The World Wide Web Consortium (<http://www.w3.org>)

3.2.12.17 SMPTE 12M

title SMPTE 12M-1999 “Television, Audio and Film - Time and Control Code”

author Society of Motion Picture and Television Engineers (<http://www.smpte.org>)

3.2.12.18 SMPTE 170M

title SMPTE 170M-1999 “Television - Composite Analog Video Signal - NTSC for Studio Applications”

author Society of Motion Picture and Television Engineers (<http://www.smpte.org>)

3.2.12.19 SMPTE 240M

title SMPTE 240M-1999 “Television - Signal Parameters - 1125-Line High-Definition Production”

author Society of Motion Picture and Television Engineers (<http://www.smpte.org>)

3.2.12.20 SMPTE RP 431-2

title SMPTE RP 431-2:2011 “D-Cinema Quality - Reference Projector and Environment”

author Society of Motion Picture and Television Engineers (<http://www.smpte.org>)

3.2.12.21 SMPTE ST 2084

title SMPTE ST 2084:2014 “High Dynamic Range Electro-Optical Transfer Function of Master Reference Displays”

author Society of Motion Picture and Television Engineers (<http://www.smpte.org>)

3.2.12.22 sRGB

title IEC 61966-2-1 ed1.0 “Multimedia systems and equipment - Colour measurement and management - Part 2-1: Colour management - Default RGB colour space - sRGB”

author International Electrotechnical Commission (<http://www.iec.ch>)

3.2.12.23 sYCC

title IEC 61966-2-1-am1 ed1.0 “Amendment 1 - Multimedia systems and equipment - Colour measurement and management - Part 2-1: Colour management - Default RGB colour space - sRGB”

author International Electrotechnical Commission (<http://www.iec.ch>)

3.2.12.24 xvYCC

title IEC 61966-2-4 ed1.0 “Multimedia systems and equipment - Colour measurement and management - Part 2-4: Colour management - Extended-gamut YCC colour space for video applications - xvYCC”

author International Electrotechnical Commission (<http://www.iec.ch>)

3.2.12.25 opRGB

title IEC 61966-2-5 “Multimedia systems and equipment - Colour measurement and management - Part 2-5: Colour management - Optional RGB colour space - opRGB”

author International Electrotechnical Commission (<http://www.iec.ch>)

3.2.12.26 ITU BT.2020

title ITU-R Recommendation BT.2020 (08/2012) “Parameter values for ultra-high definition television systems for production and international programme exchange”

author International Telecommunication Union (<http://www.itu.ch>)

3.2.12.27 EBU Tech 3213

title E.B.U. Standard for Chromaticity Tolerances for Studio Monitors

author European Broadcast Union (<http://www.ebu.ch>)

3.2.12.28 EBU Tech 3321

title E.B.U. guidelines for Consumer Flat Panel Displays (FPDs)

author European Broadcast Union (<http://www.ebu.ch>)

3.2.12.29 IEC 62106

title Specification of the radio data system (RDS) for VHF/FM sound broadcasting in the frequency range from 87,5 to 108,0 MHz

author International Electrotechnical Commission (<http://www.iec.ch>)

3.2.12.30 NRSC-4-B

title NRSC-4-B: United States RBDS Standard

author National Radio Systems Committee (<http://www.nrsstandards.org>)

3.2.12.31 ISO 12232:2006

title Photography — Digital still cameras — Determination of exposure index, ISO speed ratings, standard output sensitivity, and recommended exposure index

author International Organization for Standardization (<http://www.iso.org>)

3.2.12.32 CEA-861-E

title A DTV Profile for Uncompressed High Speed Digital Interfaces

author Consumer Electronics Association (<http://www.ce.org>)

3.2.12.33 VESA DMT

title VESA and Industry Standards and Guidelines for Computer Display Monitor Timing (DMT)

author Video Electronics Standards Association (<http://www.vesa.org>)

3.2.12.34 EDID

title VESA Enhanced Extended Display Identification Data Standard

subtitle Release A, Revision 2

author Video Electronics Standards Association (<http://www.vesa.org>)

3.2.12.35 HDCP

title High-bandwidth Digital Content Protection System

subtitle Revision 1.3

author Digital Content Protection LLC (<http://www.digital-cp.com>)

3.2.12.36 HDMI

title High-Definition Multimedia Interface

subtitle Specification Version 1.4a

author HDMI Licensing LLC (<http://www.hdmi.org>)

3.2.12.37 HDMI2

title High-Definition Multimedia Interface

subtitle Specification Version 2.0

author HDMI Licensing LLC (<http://www.hdmi.org>)

3.2.12.38 DP

title VESA DisplayPort Standard

subtitle Version 1, Revision 2

author Video Electronics Standards Association (<http://www.vesa.org>)

3.2.12.39 poynton

title Digital Video and HDTV, Algorithms and Interfaces

author Charles Poynton

3.2.12.40 colimg

title Color Imaging: Fundamentals and Applications

author Erik Reinhard et al.

3.2.12.41 VP8

title RFC 6386: “VP8 Data Format and Decoding Guide”

author

J. Bankoski et al.

3.2.12.42 VP9

title VP9 Bitstream & Decoding Process Specification

author Adrian Grange (Google), Peter de Rivaz (Argon Design), Jonathan Hunt (Argon Design)

3.2.13 Revision and Copyright

Authors, in alphabetical order:

- Ailus, Sakari <sakari.ailus@iki.fi>
 - Subdev selections API.
- Carvalho Chehab, Mauro <mchehab+samsung@kernel.org>
 - Documented libv4l, designed and added v4l2grab example, Remote Controller chapter.
- Dirks, Bill
 - Original author of the V4L2 API and documentation.
- Figa, Tomasz <tfiga@chromium.org>
 - Documented the memory-to-memory decoder interface.
 - Documented the memory-to-memory encoder interface.
- H Schimek, Michael <mschimek@gmx.at>
 - Original author of the V4L2 API and documentation.
- Karicheri, Muralidharan <m-karicheri2@ti.com>
 - Documented the Digital Video timings API.
- Osciak, Pawel <posciak@chromium.org>
 - Documented the memory-to-memory decoder interface.
 - Documented the memory-to-memory encoder interface.
- Osciak, Pawel <pawel@osciak.com>

- Designed and documented the multi-planar API.
- Palosaari, Antti <crope@iki.fi>
 - SDR API.
- Ribalda, Ricardo
 - Introduce HSV formats and other minor changes.
- Rubli, Martin
 - Designed and documented the VIDIOC_ENUM_FRAMESIZES and VIDIOC_ENUM_FRAMEINTERVALS ioctls.
- Walls, Andy <awalls@md.metrocast.net>
 - Documented the fielded V4L2_MPEG_STREAM_VBI_FMT_IVTV MPEG stream embedded, sliced VBI data format in this specification.
- Verkuil, Hans <hverkuil@xs4all.nl>
 - Designed and documented the VIDIOC_LOG_STATUS ioctl, the extended control ioctls, major parts of the sliced VBI API, the MPEG encoder and decoder APIs and the DV Timings API.

Copyright © 1999-2018: Bill Dirks, Michael H. Schimek, Hans Verkuil, Martin Rubli, Andy Walls, Muralidharan Karicheri, Mauro Carvalho Chehab, Pawel Osciak, Sakari Ailus & Antti Palosaari, Tomasz Figa

Except when explicitly stated as GPL, programming examples within this part can be used and distributed without restrictions.

3.2.14 Revision History

revision 4.10 / 2016-07-15 (*rr*)

Introduce HSV formats.

revision 4.5 / 2015-10-29 (*rr*)

Extend VIDIOC_G_EXT_CTRL;: Replace ctrl_class with a new union with ctrl_class and which. Which is used to select the current value of the control or the default value.

revision 4.4 / 2015-05-26 (*ap*)

Renamed V4L2_TUNER_ADC to V4L2_TUNER_SDR. Added V4L2_CID_RF_TUNER_RF_GAIN control. Added transmitter support for Software Defined Radio (SDR) Interface.

revision 4.1 / 2015-02-13 (*mcc*)

Fix documentation for media controller device nodes and add support for DVB device nodes. Add support for Tuner sub-device.

revision 3.19 / 2014-12-05 (*hv*)

Rewrote Colorspace chapter, added new enum *v4l2_ycbcr_encoding* and enum *v4l2_quantization* fields to struct *v4l2_pix_format*, struct *v4l2_pix_format_mplane* and struct *v4l2_mbus_framefmt*.

revision 3.17 / 2014-08-04 (*lp, hv*)

Extended struct `v4l2_pix_format`. Added format flags. Added compound control types and `VIDIOC_QUERY_EXT_CTRL`.

revision 3.15 / 2014-02-03 (*hv, ap*)

Update several sections of “Common API Elements”: “Opening and Closing Devices” “Querying Capabilities”, “Application Priority”, “Video Inputs and Outputs”, “Audio Inputs and Outputs” “Tuners and Modulators”, “Video Standards” and “Digital Video (DV) Timings”. Added SDR API.

revision 3.14 / 2013-11-25 (*rr*)

Set width and height as unsigned on `v4l2_rect`.

revision 3.11 / 2013-05-26 (*hv*)

Remove obsolete `VIDIOC_DBG_G_CHIP_IDENT` ioctl.

revision 3.10 / 2013-03-25 (*hv*)

Remove obsolete and unused `DV_PRESET` ioctls: `VIDIOC_G_DV_PRESET`, `VIDIOC_S_DV_PRESET`, `VIDIOC_QUERY_DV_PRESET` and `VIDIOC_ENUM_DV_PRESET`. Remove the related `v4l2_input/output` capability flags `V4L2_IN_CAP_PRESETS` and `V4L2_OUT_CAP_PRESETS`. Added `VIDIOC_DBG_G_CHIP_INFO`.

revision 3.9 / 2012-12-03 (*sa, sn*)

Added timestamp types to `v4l2_buffer`. Added `V4L2_EVENT_CTRL_CH_RANGE` control event changes flag.

revision 3.6 / 2012-07-02 (*hv*)

Added `VIDIOC_ENUM_FREQ_BANDS`.

revision 3.5 / 2012-05-07 (*sa, sn, hv*)

Added `V4L2_CTRL_TYPE_INTEGER_MENU` and `V4L2` subdev selections API. Improved the description of `V4L2_CID_COLORFX` control, added `V4L2_CID_COLORFX_CBCR` control. Added camera controls `V4L2_CID_AUTO_EXPOSURE_BIAS`, `V4L2_CID_AUTO_N_PRESET_WHITE_BALANCE`, `V4L2_CID_IMAGE_STABILIZATION`, `V4L2_CID_ISO_SENSITIVITY`, `V4L2_CID_ISO_SENSITIVITY_AUTO`, `V4L2_CID_EXPOSURE_METERING`, `V4L2_CID_SCENE_MODE`, `V4L2_CID_3A_LOCK`, `V4L2_CID_AUTO_FOCUS_START`, `V4L2_CID_AUTO_FOCUS_STOP`, `V4L2_CID_AUTO_FOCUS_STATUS` and `V4L2_CID_AUTO_FOCUS_RANGE`. Added `VIDIOC_ENUM_DV_TIMINGS`, `VIDIOC_QUERY_DV_TIMINGS` and `VIDIOC_DV_TIMINGS_CAP`.

revision 3.4 / 2012-01-25 (*sn*)

Added *JPEG compression control class*.

revision 3.3 / 2012-01-11 (*hv*)

Added `device_caps` field to struct `v4l2_capabilities`.

revision 3.2 / 2011-08-26 (*hv*)

Added `V4L2_CTRL_FLAG_VOLATILE`.

revision 3.1 / 2011-06-27 (*mcc, po, hv*)

Documented that `VIDIOC_QUERYCAP` now returns a per-subsystem version instead of a per-driver one. Standardize an error code for invalid ioctl. Added `V4L2_CTRL_TYPE_BITMASK`.

revision 2.6.39 / 2011-03-01 (*mcc, po*)

Removed VIDIOC_*_OLD from videodev2.h header and update it to reflect latest changes. Added the *multi-planar API*.

revision 2.6.37 / 2010-08-06 (*hv*)

Removed obsolete vtx (videotext) API.

revision 2.6.33 / 2009-12-03 (*mk*)

Added documentation for the Digital Video timings API.

revision 2.6.32 / 2009-08-31 (*mcc*)

Now, revisions will match the kernel version where the V4L2 API changes will be used by the Linux Kernel. Also added Remote Controller chapter.

revision 0.29 / 2009-08-26 (*ev*)

Added documentation for string controls and for FM Transmitter controls.

revision 0.28 / 2009-08-26 (*gl*)

Added V4L2_CID_BAND_STOP_FILTER documentation.

revision 0.27 / 2009-08-15 (*mcc*)

Added libv4l and Remote Controller documentation; added v4l2grab and keytable application examples.

revision 0.26 / 2009-07-23 (*hv*)

Finalized the RDS capture API. Added modulator and RDS encoder capabilities. Added support for string controls.

revision 0.25 / 2009-01-18 (*hv*)

Added pixel formats VYUY, NV16 and NV61, and changed the debug ioctls VIDIOC_DBG_G/S_REGISTER and VIDIOC_DBG_G_CHIP_IDENT. Added camera controls V4L2_CID_ZOOM_ABSOLUTE, V4L2_CID_ZOOM_RELATIVE, V4L2_CID_ZOOM_CONTINUOUS and V4L2_CID_PRIVACY.

revision 0.24 / 2008-03-04 (*mhs*)

Added pixel formats Y16 and SBGGR16, new controls and a camera controls class. Removed VIDIOC_G/S_MPEGCOMP.

revision 0.23 / 2007-08-30 (*mhs*)

Fixed a typo in VIDIOC_DBG_G/S_REGISTER. Clarified the byte order of packed pixel formats.

revision 0.22 / 2007-08-29 (*mhs*)

Added the Video Output Overlay interface, new MPEG controls, V4L2_FIELD_INTERLACED_TB and V4L2_FIELD_INTERLACED_BT, VIDIOC_DBG_G/S_REGISTER, VIDIOC_(TRY_)ENCODER_CMD, VIDIOC_G_CHIP_IDENT, VIDIOC_G_ENC_INDEX, new pixel formats. Clarifications in the cropping chapter, about RGB pixel formats, the mmap(), poll(), select(), read() and write() functions. Typographical fixes.

revision 0.21 / 2006-12-19 (*mhs*)

Fixed a link in the VIDIOC_G_EXT_CTRLs section.

revision 0.20 / 2006-11-24 (*mhs*)

Clarified the purpose of the audioset field in struct `v4l2_input` and `v4l2_output`.

revision 0.19 / 2006-10-19 (*mhs*)

Documented `V4L2_PIX_FMT_RGB444`.

revision 0.18 / 2006-10-18 (*mhs*)

Added the description of extended controls by Hans Verkuil. Linked `V4L2_PIX_FMT_MPEG` to `V4L2_CID_MPEG_STREAM_TYPE`.

revision 0.17 / 2006-10-12 (*mhs*)

Corrected `V4L2_PIX_FMT_HM12` description.

revision 0.16 / 2006-10-08 (*mhs*)

`VIDIOC_ENUM_FRAMESIZES` and `VIDIOC_ENUM_FRAMEINTERVALS` are now part of the API.

revision 0.15 / 2006-09-23 (*mhs*)

Cleaned up the bibliography, added BT.653 and BT.1119. `capture.c/start_capturing()` for user pointer I/O did not initialize the buffer index. Documented the V4L MPEG and MJPEG `VID_TYPE`s and `V4L2_PIX_FMT_SBGGR8`. Updated the list of reserved pixel formats. See the history chapter for API changes.

revision 0.14 / 2006-09-14 (*mr*)

Added `VIDIOC_ENUM_FRAMESIZES` and `VIDIOC_ENUM_FRAMEINTERVALS` proposal for frame format enumeration of digital devices.

revision 0.13 / 2006-04-07 (*mhs*)

Corrected the description of struct `v4l2_window` clips. New `V4L2_STD_` and `V4L2_TUNER_MODE_LANG1_LANG2` defines.

revision 0.12 / 2006-02-03 (*mhs*)

Corrected the description of struct `v4l2_captureparm` and `v4l2_outputparm`.

revision 0.11 / 2006-01-27 (*mhs*)

Improved the description of struct `v4l2_tuner`.

revision 0.10 / 2006-01-10 (*mhs*)

`VIDIOC_G_INPUT` and `VIDIOC_S_PARM` clarifications.

revision 0.9 / 2005-11-27 (*mhs*)

Improved the 525 line numbering diagram. Hans Verkuil and I rewrote the sliced VBI section. He also contributed a `VIDIOC_LOG_STATUS` page. Fixed `VIDIOC_S_STD` call in the video standard selection example. Various updates.

revision 0.8 / 2004-10-04 (*mhs*)

Somehow a piece of junk slipped into the capture example, removed.

revision 0.7 / 2004-09-19 (*mhs*)

Fixed video standard selection, control enumeration, downscaling and aspect example. Added read and user pointer i/o to video capture example.

revision 0.6 / 2004-08-01 (*mhs*)

v4l2_buffer changes, added video capture example, various corrections.

revision 0.5 / 2003-11-05 (*mhs*)

Pixel format erratum.

revision 0.4 / 2003-09-17 (*mhs*)

Corrected source and Makefile to generate a PDF. SGML fixes. Added latest API changes. Closed gaps in the history chapter.

revision 0.3 / 2003-02-05 (*mhs*)

Another draft, more corrections.

revision 0.2 / 2003-01-15 (*mhs*)

Second draft, with corrections pointed out by Gerd Knorr.

revision 0.1 / 2002-12-01 (*mhs*)

First draft, based on documentation by Bill Dirks and discussions on the V4L mailing list.

3.3 Part II - Digital TV API

Note: This API is also known as Linux **DVB API**.

It was originally written to support the European digital TV standard (DVB), and later extended to support all digital TV standards.

In order to avoid confusion, within this document, it was opted to refer to it, and to associated hardware as **Digital TV**.

The word **DVB** is reserved to be used for:

- the Digital TV API version (e. g. DVB API version 3 or DVB API version 5);
 - digital TV data types (enums, structs, defines, etc);
 - digital TV device nodes (`/dev/dvb/...`);
 - the European DVB standard.
-

Version 5.10

3.3.1 Introduction

3.3.1.1 What you need to know

The reader of this document is required to have some knowledge in the area of digital video broadcasting (Digital TV) and should be familiar with part I of the MPEG2 specification ISO/IEC 13818 (aka ITU-T H.222), i.e you should know what a program/transport stream (PS/TS) is and what is meant by a packetized elementary stream (PES) or an I-frame.

Various Digital TV standards documents are available for download at:

- European standards (DVB): <http://www.dvb.org> and/or <http://www.etsi.org>.
- American standards (ATSC): <https://www.atsc.org/standards/>
- Japanese standards (ISDB): <http://www.dibeg.org/>

It is also necessary to know how to access Linux devices and how to use ioctl calls. This also includes the knowledge of C or C++.

3.3.1.2 History

The first API for Digital TV cards we used at Convergence in late 1999 was an extension of the Video4Linux API which was primarily developed for frame grabber cards. As such it was not really well suited to be used for Digital TV cards and their new features like recording MPEG streams and filtering several section and PES data streams at the same time.

In early 2000, Convergence was approached by Nokia with a proposal for a new standard Linux Digital TV API. As a commitment to the development of terminals based on open standards, Nokia and Convergence made it available to all Linux developers and published it on <https://linuxtv.org> in September 2000. With the Linux driver for the Siemens/Hauppauge DVB PCI card, Convergence provided a first implementation of the Linux Digital TV API. Convergence was the maintainer of the Linux Digital TV API in the early days.

Now, the API is maintained by the LinuxTV community (i.e. you, the reader of this document). The Linux Digital TV API is constantly reviewed and improved together with the improvements at the subsystem's core at the Kernel.

3.3.1.3 Overview

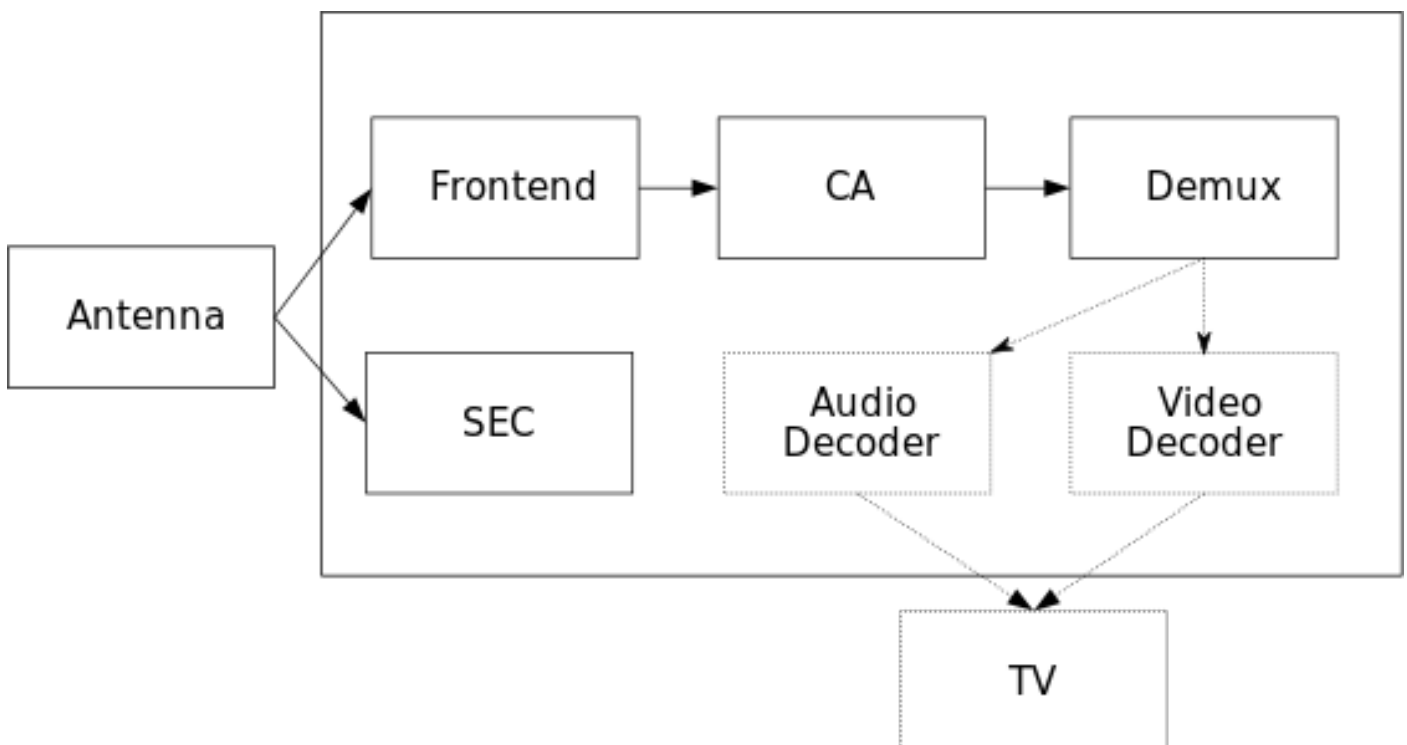


Fig. 19: Components of a Digital TV card/STB

A Digital TV card or set-top-box (STB) usually consists of the following main hardware components:

Frontend consisting of tuner and digital TV demodulator Here the raw signal reaches the digital TV hardware from a satellite dish or antenna or directly from cable. The frontend down-converts and demodulates this signal into an MPEG transport stream (TS). In case of a satellite frontend, this includes a facility for satellite equipment control (SEC), which allows control of LNB polarization, multi feed switches or dish rotors.

Conditional Access (CA) hardware like CI adapters and smartcard slots The complete TS is passed through the CA hardware. Programs to which the user has access (controlled by the smart card) are decoded in real time and re-inserted into the TS.

Note: Not every digital TV hardware provides conditional access hardware.

Demultiplexer which filters the incoming Digital TV MPEG-TS stream The demultiplexer splits the TS into its components like audio and video streams. Besides usually several of such audio and video streams it also contains data streams with information about the programs offered in this or other streams of the same provider.

Audio and video decoder The main targets of the demultiplexer are audio and video decoders. After decoding, they pass on the uncompressed audio and video to the computer screen or to a TV set.

Note: Modern hardware usually doesn't have a separate decoder hardware, as such functionality can be provided by the main CPU, by the graphics adapter of the system or by a signal processing hardware embedded on a Systems on a Chip (SoC) integrated circuit.

It may also not be needed for certain usages (e.g. for data-only uses like "internet over satellite").

Components of a Digital TV card/STB shows a crude schematic of the control and data flow between those components.

3.3.1.4 Linux Digital TV Devices

The Linux Digital TV API lets you control these hardware components through currently six Unix-style character devices for video, audio, frontend, demux, CA and IP-over-DVB networking. The video and audio devices control the MPEG2 decoder hardware, the frontend device the tuner and the Digital TV demodulator. The demux device gives you control over the PES and section filters of the hardware. If the hardware does not support filtering these filters can be implemented in software. Finally, the CA device controls all the conditional access capabilities of the hardware. It can depend on the individual security requirements of the platform, if and how many of the CA functions are made available to the application through this device.

All devices can be found in the /dev tree under /dev/dvb. The individual devices are called:

- /dev/dvb/adapterN/audioM,
- /dev/dvb/adapterN/videoM,
- /dev/dvb/adapterN/frontendM,

- /dev/dvb/adapterN/netM,
- /dev/dvb/adapterN/demuxM,
- /dev/dvb/adapterN/dvrM,
- /dev/dvb/adapterN/caM,

where N enumerates the Digital TV cards in a system starting from 0, and M enumerates the devices of each type within each adapter, starting from 0, too. We will omit the “/dev/dvb/adapterN/” in the further discussion of these devices.

More details about the data structures and function calls of all the devices are described in the following chapters.

3.3.1.5 API include files

For each of the Digital TV devices a corresponding include file exists. The Digital TV API include files should be included in application sources with a partial path like:

```
#include <linux/dvb/ca.h>
#include <linux/dvb/dmx.h>
#include <linux/dvb/frontend.h>
#include <linux/dvb/net.h>
```

To enable applications to support different API version, an additional include file `linux/dvb/version.h` exists, which defines the constant `DVB_API_VERSION`. This document describes `DVB_API_VERSION 5.10`.

3.3.2 Digital TV Frontend API

The Digital TV frontend API was designed to support three groups of delivery systems: Terrestrial, cable and Satellite. Currently, the following delivery systems are supported:

- Terrestrial systems: DVB-T, DVB-T2, ATSC, ATSC M/H, ISDB-T, DVB-H, DTMB, CMMB
- Cable systems: DVB-C Annex A/C, ClearQAM (DVB-C Annex B)
- Satellite systems: DVB-S, DVB-S2, DVB Turbo, ISDB-S, DSS

The Digital TV frontend controls several sub-devices including:

- Tuner
- Digital TV demodulator
- Low noise amplifier (LNA)
- Satellite Equipment Control (SEC)¹.

¹ On Satellite systems, the API support for the Satellite Equipment Control (SEC) allows to power control and to send/receive signals to control the antenna subsystem, selecting the polarization and choosing the Intermediate Frequency (IF) of the Low Noise Block Converter Feed Horn (LNBf). It supports the DiSEqC and V-SEC protocols. The DiSEqC (digital SEC) specification is available at [Eutelsat](#).

The frontend can be accessed through `/dev/dvb/adapter?/frontend?`. Data types and ioctl definitions can be accessed by including `linux/dvb/frontend.h` in your application.

Note: Transmission via the internet (DVB-IP) and MMT (MPEG Media Transport) is not yet handled by this API but a future extension is possible.

3.3.2.1 Querying frontend information

Usually, the first thing to do when the frontend is opened is to check the frontend capabilities. This is done using `ioctl FE_GET_INFO`. This ioctl will enumerate the Digital TV API version and other characteristics about the frontend, and can be opened either in read only or read/write mode.

3.3.2.2 Querying frontend status and statistics

Once `FE_SET_PROPERTY` is called, the frontend will run a kernel thread that will periodically check for the tuner lock status and provide statistics about the quality of the signal.

The information about the frontend tuner locking status can be queried using `ioctl FE_READ_STATUS`.

Signal statistics are provided via `ioctl FE_SET_PROPERTY, FE_GET_PROPERTY`.

Note: Most statistics require the demodulator to be fully locked (e. g. with `FE_HAS_LOCK` bit set). See *Frontend statistics indicators* for more details.

3.3.2.3 Property types

Tuning into a Digital TV physical channel and starting decoding it requires changing a set of parameters, in order to control the tuner, the demodulator, the Linear Low-noise Amplifier (LNA) and to set the antenna subsystem via Satellite Equipment Control - SEC (on satellite systems). The actual parameters are specific to each particular digital TV standards, and may change as the digital TV specs evolves.

In the past (up to DVB API version 3 - DVBv3), the strategy used was to have a union with the parameters needed to tune for DVB-S, DVB-C, DVB-T and ATSC delivery systems grouped there. The problem is that, as the second generation standards appeared, the size of such union was not big enough to group the structs that would be required for those new standards. Also, extending it would break userspace.

So, the legacy union/struct based approach was deprecated, in favor of a properties set approach. On such approach, `FE_GET_PROPERTY` and `FE_SET_PROPERTY` are used to setup the frontend and read its status.

The actual action is determined by a set of `dtv_property cmd/data` pairs. With one single ioctl, is possible to get/set up to 64 properties.

This section describes the new and recommended way to set the frontend, with supports all digital TV delivery systems.

Note:

1. On Linux DVB API version 3, setting a frontend was done via struct `dvb_frontend_parameters`.
 2. Don't use DVB API version 3 calls on hardware with supports newer standards. Such API provides no support or a very limited support to new standards and/or new hardware.
 3. Nowadays, most frontends support multiple delivery systems. Only with DVB API version 5 calls it is possible to switch between the multiple delivery systems supported by a frontend.
 4. DVB API version 5 is also called *S2API*, as the first new standard added to it was DVB-S2.
-

Example: in order to set the hardware to tune into a DVB-C channel at 651 kHz, modulated with 256-QAM, FEC 3/4 and symbol rate of 5.217 Mbauds, those properties should be sent to `FE_SET_PROPERTY` ioctl:

```
DTV_DELIVERY_SYSTEM = SYS_DVBC_ANNEX_A
DTV_FREQUENCY = 651000000
DTV_MODULATION = QAM_256
DTV_INVERSION = INVERSION_AUTO
DTV_SYMBOL_RATE = 5217000
DTV_INNER_FEC = FEC_3_4
DTV_TUNE
```

The code that would that would do the above is show in *Example: Setting digital TV frontend properties*.

Listing 1: Example: Setting digital TV frontend properties

```
#include <stdio.h>
#include <fcntl.h>
#include <sys/ioctl.h>
#include <linux/dvb/frontend.h>

static struct dtv_property props[] = {
    { .cmd = DTV_DELIVERY_SYSTEM, .u.data = SYS_DVBC_ANNEX_A },
    { .cmd = DTV_FREQUENCY,       .u.data = 651000000 },
    { .cmd = DTV_MODULATION,      .u.data = QAM_256 },
    { .cmd = DTV_INVERSION,       .u.data = INVERSION_AUTO },
    { .cmd = DTV_SYMBOL_RATE,     .u.data = 5217000 },
    { .cmd = DTV_INNER_FEC,       .u.data = FEC_3_4 },
    { .cmd = DTV_TUNE }
};

static struct dtv_properties dtv_prop = {
    .num = 6, .props = props
};

int main(void)
{
    int fd = open("/dev/dvb/adapter0/frontend0", O_RDWR);
```



```
if (!fd) {
    perror ("open");
    return -1;
}
if (ioctl(fd, FE_SET_PROPERTY, &dtv_prop) == -1) {
    perror("ioctl");
    return -1;
}
printf("Frontend set\\n");
return 0;
}
```

Attention: While it is possible to directly call the Kernel code like the above example, it is strongly recommended to use [libdvbv5](#), as it provides abstraction to work with the supported digital TV standards and provides methods for usual operations like program scanning and to read/write channel descriptor files.

Digital TV property parameters

There are several different Digital TV parameters that can be used by *FE_SET_PROPERTY* and *FE_GET_PROPERTY* *ioctls*. This section describes each of them. Please notice, however, that only a subset of them are needed to setup a frontend.

DTV_UNDEFINED

Used internally. A GET/SET operation for it won't change or return anything.

DTV_TUNE

Interpret the cache of data, build either a traditional frontend tunerequest so we can pass validation in the *FE_SET_FRONTEND* *ioctl*.

DTV_CLEAR

Reset a cache of data specific to the frontend here. This does not effect hardware.

DTV_FREQUENCY

Frequency of the digital TV transponder/channel.

Note:

1. For satellite delivery systems, the frequency is in kHz.
 2. For cable and terrestrial delivery systems, the frequency is in Hz.
 3. On most delivery systems, the frequency is the center frequency of the transponder/channel. The exception is for ISDB-T, where the main carrier has a 1/7 offset from the center.
 4. For ISDB-T, the channels are usually transmitted with an offset of about 143kHz. E.g. a valid frequency could be 474,143 kHz. The stepping is bound to the bandwidth of the channel which is typically 6MHz.
 5. In ISDB-Tsb, the channel consists of only one or three segments the frequency step is 429kHz, 3*429 respectively.
-

DTV_MODULATION

Specifies the frontend modulation type for delivery systems that supports more multiple modulations.

The modulation can be one of the types defined by enum `fe_modulation`.

Most of the digital TV standards offers more than one possible modulation type.

The table below presents a summary of the types of modulation types supported by each delivery system, as currently defined by specs.

Standard	Modulation types
ATSC (version 1)	8-VSB and 16-VSB.
DMTB	4-QAM, 16-QAM, 32-QAM, 64-QAM and 4-QAM-NR.
DVB-C Annex A/C	16-QAM, 32-QAM, 64-QAM and 256-QAM.
DVB-C Annex B	64-QAM.
DVB-T	QPSK, 16-QAM and 64-QAM.
DVB-T2	QPSK, 16-QAM, 64-QAM and 256-QAM.
DVB-S	No need to set. It supports only QPSK.
DVB-S2	QPSK, 8-PSK, 16-APSK and 32-APSK.
ISDB-T	QPSK, DQPSK, 16-QAM and 64-QAM.
ISDB-S	8-PSK, QPSK and BPSK.

Note: Please notice that some of the above modulation types may not be defined currently at the Kernel. The reason is simple: no driver needed such definition yet.

DTV_BANDWIDTH_HZ

Bandwidth for the channel, in HZ.

Should be set only for terrestrial delivery systems.

Possible values: 1712000, 5000000, 6000000, 7000000, 8000000, 10000000.

Terrestrial Standard	Possible values for bandwidth
ATSC (version 1)	No need to set. It is always 6MHz.
DMTB	No need to set. It is always 8MHz.
DVB-T	6MHz, 7MHz and 8MHz.
DVB-T2	1.172 MHz, 5MHz, 6MHz, 7MHz, 8MHz and 10MHz
ISDB-T	5MHz, 6MHz, 7MHz and 8MHz, although most places use 6MHz.

Note:

1. For ISDB-Tsb, the bandwidth can vary depending on the number of connected segments. It can be easily derived from other parameters (`DTV_ISDBT_SB_SEGMENT_IDX`, `DTV_ISDBT_SB_SEGMENT_COUNT`).
2. On Satellite and Cable delivery systems, the bandwidth depends on the symbol rate. So, the Kernel will silently ignore any setting `DTV_BANDWIDTH_HZ`. I will however fill it back with a bandwidth estimation.

Such bandwidth estimation takes into account the symbol rate set with `DTV_SYMBOL_RATE`, and the rolloff factor, with is fixed for DVB-C and DVB-S.

For DVB-S2, the rolloff should also be set via `DTV_ROLLOFF`.

DTV_INVERSION

Specifies if the frontend should do spectral inversion or not.

The acceptable values are defined by `fe_spectral_inversion`.

DTV_DISEQC_MASTER

Currently not implemented.

DTV_SYMBOL_RATE

Used on cable and satellite delivery systems.

Digital TV symbol rate, in bauds (symbols/second).

DTV_INNER_FEC

Used on cable and satellite delivery systems.

The acceptable values are defined by `fe_code_rate`.

DTV_VOLTAGE

Used on satellite delivery systems.

The voltage is usually used with non-DiSEqC capable LNBs to switch the polarization (horizontal/vertical). When using DiSEqC equipment this voltage has to be switched consistently to the DiSEqC commands as described in the DiSEqC spec.

The acceptable values are defined by `fe_sec_voltage`.

DTV_TONE

Currently not used.

DTV_PILOT

Used on DVB-S2.

Sets DVB-S2 pilot.

The acceptable values are defined by `fe_pilot`.

DTV_ROLLOFF

Used on DVB-S2.

Sets DVB-S2 rolloff.

The acceptable values are defined by `fe_rolloff`.

DTV_DISEQC_SLAVE_REPLY

Currently not implemented.

DTV_FE_CAPABILITY_COUNT

Currently not implemented.

DTV_FE_CAPABILITY

Currently not implemented.

DTV_DELIVERY_SYSTEM

Specifies the type of the delivery system.

The acceptable values are defined by `fe_delivery_system`.

DTV_ISDBT_PARTIAL_RECEPTION

Used only on ISDB.

If `DTV_ISDBT_SOUND_BROADCASTING` is '0' this bit-field represents whether the channel is in partial reception mode or not.

If '1' `DTV_ISDBT_LAYERA_*` values are assigned to the center segment and `DTV_ISDBT_LAYERA_SEGMENT_COUNT` has to be '1'.

If in addition `DTV_ISDBT_SOUND_BROADCASTING` is '1' `DTV_ISDBT_PARTIAL_RECEPTION` represents whether this ISDB-Tsb channel is consisting of one segment and layer or three segments and two layers.

Possible values: 0, 1, -1 (AUTO)

DTV_ISDBT_SOUND_BROADCASTING

Used only on ISDB.

This field represents whether the other `DTV_ISDBT_*`-parameters are referring to an ISDB-T and an ISDB-Tsb channel. (See also `DTV_ISDBT_PARTIAL_RECEPTION`).

Possible values: 0, 1, -1 (AUTO)

DTV_ISDBT_SB_SUBCHANNEL_ID

Used only on ISDB.

This field only applies if `DTV_ISDBT_SOUND_BROADCASTING` is '1'.

(Note of the author: This might not be the correct description of the `SUBCHANNEL-ID` in all details, but it is my understanding of the technical background needed to program a device)

An ISDB-Tsb channel (1 or 3 segments) can be broadcasted alone or in a set of connected ISDB-Tsb channels. In this set of channels every channel can be received independently. The number of connected ISDB-Tsb segment can vary, e.g. depending on the frequency spectrum bandwidth available.

Example: Assume 8 ISDB-Tsb connected segments are broadcasted. The broadcaster has several possibilities to put those channels in the air: Assuming a normal 13-segment ISDB-T spectrum he can align the 8 segments from position 1-8 to 5-13 or anything in between.

The underlying layer of segments are subchannels: each segment is consisting of several sub-channels with a predefined IDs. A sub-channel is used to help the demodulator to synchronize on the channel.

An ISDB-T channel is always centered over all sub-channels. As for the example above, in ISDB-Tsb it is no longer as simple as that.

The `DTV_ISDBT_SB_SUBCHANNEL_ID` parameter is used to give the sub-channel ID of the segment to be demodulated.

Possible values: 0 .. 41, -1 (AUTO)

DTV_ISDBT_SB_SEGMENT_IDX

Used only on ISDB.

This field only applies if `DTV_ISDBT_SOUND_BROADCASTING` is '1'.

`DTV_ISDBT_SB_SEGMENT_IDX` gives the index of the segment to be demodulated for an ISDB-Tsb channel where several of them are transmitted in the connected manner.

Possible values: 0 .. `DTV_ISDBT_SB_SEGMENT_COUNT` - 1

Note: This value cannot be determined by an automatic channel search.

DTV_ISDBT_SB_SEGMENT_COUNT

Used only on ISDB.

This field only applies if `DTV_ISDBT_SOUND_BROADCASTING` is '1'.

`DTV_ISDBT_SB_SEGMENT_COUNT` gives the total count of connected ISDB-Tsb channels.

Possible values: 1 .. 13

Note: This value cannot be determined by an automatic channel search.

DTV-ISDBT-LAYER[A-C] parameters

Used only on ISDB.

ISDB-T channels can be coded hierarchically. As opposed to DVB-T in ISDB-T hierarchical layers can be decoded simultaneously. For that reason a ISDB-T demodulator has 3 Viterbi and 3 Reed-Solomon decoders.

ISDB-T has 3 hierarchical layers which each can use a part of the available segments. The total number of segments over all layers has to 13 in ISDB-T.

There are 3 parameter sets, for Layers A, B and C.

DTV_ISDBT_LAYER_ENABLED

Used only on ISDB.

Hierarchical reception in ISDB-T is achieved by enabling or disabling layers in the decoding process. Setting all bits of DTV_ISDBT_LAYER_ENABLED to '1' forces all layers (if applicable) to be demodulated. This is the default.

If the channel is in the partial reception mode (DTV_ISDBT_PARTIAL_RECEPTION = 1) the central segment can be decoded independently of the other 12 segments. In that mode layer A has to have a SEGMENT_COUNT of 1.

In ISDB-Tsb only layer A is used, it can be 1 or 3 in ISDB-Tsb according to DTV_ISDBT_PARTIAL_RECEPTION. SEGMENT_COUNT must be filled accordingly.

Only the values of the first 3 bits are used. Other bits will be silently ignored:

DTV_ISDBT_LAYER_ENABLED bit 0: layer A enabled

DTV_ISDBT_LAYER_ENABLED bit 1: layer B enabled

DTV_ISDBT_LAYER_ENABLED bit 2: layer C enabled

DTV_ISDBT_LAYER_ENABLED bits 3-31: unused

DTV_ISDBT_LAYER[A-C]_FEC

Used only on ISDB.

The Forward Error Correction mechanism used by a given ISDB Layer, as defined by fe_code_rate.

Possible values are: FEC_AUTO, FEC_1_2, FEC_2_3, FEC_3_4, FEC_5_6, FEC_7_8

DTV_ISDBT_LAYER[A-C]_MODULATION

Used only on ISDB.

The modulation used by a given ISDB Layer, as defined by `fe_modulation`.

Possible values are: QAM_AUTO, QPSK, QAM_16, QAM_64, DQPSK

Note:

1. If layer C is DQPSK, then layer B has to be DQPSK.
2. If layer B is DQPSK and `DTV_ISDBT_PARTIAL_RECEPTION= 0`, then layer has to be DQPSK.

DTV_ISDBT_LAYER[A-C]_SEGMENT_COUNT

Used only on ISDB.

Possible values: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, -1 (AUTO)

Note: Truth table for `DTV_ISDBT_SOUND_BROADCASTING` and `DTV_ISDBT_PARTIAL_RECEPTION` and `LAYER[A-C]_SEGMENT_COUNT`

Table 248: Truth table for ISDB-T Sound Broadcasting

Partial Reception	Sound Broadcasting	Layer A width	Layer B width	Layer C width	total width
0	0	1 .. 13	1 .. 13	1 .. 13	13
1	0	1	1 .. 13	1 .. 13	13
0	1	1	0	0	1
1	1	1	2	0	13

DTV_ISDBT_LAYER[A-C]_TIME_INTERLEAVING

Used only on ISDB.

Valid values: 0, 1, 2, 4, -1 (AUTO)

when `DTV_ISDBT_SOUND_BROADCASTING` is active, value 8 is also valid.

Note: The real time interleaving length depends on the mode (fft-size). The values here are referring to what can be found in the TMCC-structure, as shown in the table below.

isdbt_layer_interleaving_table

Table 249: ISDB-T time interleaving modes

DTV_ISDBT_LAYER[A-C]_TIME_INTERLEAVING	Mode 1 (2K FFT)	Mode 2 (4K FFT)	Mode 3 (8K FFT)
0	0	0	0
1	4	2	1
2	8	4	2
4	16	8	4

DTV_ATSCMH_FIC_VER

Used only on ATSC-MH.

Version number of the FIC (Fast Information Channel) signaling data.

FIC is used for relaying information to allow rapid service acquisition by the receiver.

Possible values: 0, 1, 2, 3, ..., 30, 31

DTV_ATSCMH_PARADE_ID

Used only on ATSC-MH.

Parade identification number

A parade is a collection of up to eight MH groups, conveying one or two ensembles.

Possible values: 0, 1, 2, 3, ..., 126, 127

DTV_ATSCMH_NOG

Used only on ATSC-MH.

Number of MH groups per MH subframe for a designated parade.

Possible values: 1, 2, 3, 4, 5, 6, 7, 8

DTV_ATSCMH_TNOG

Used only on ATSC-MH.

Total number of MH groups including all MH groups belonging to all MH parades in one MH subframe.

Possible values: 0, 1, 2, 3, ..., 30, 31

DTV_ATSCMH_SGN

Used only on ATSC-MH.

Start group number.

Possible values: 0, 1, 2, 3, ..., 14, 15

DTV_ATSCMH_PRC

Used only on ATSC-MH.

Parade repetition cycle.

Possible values: 1, 2, 3, 4, 5, 6, 7, 8

DTV_ATSCMH_RS_FRAME_MODE

Used only on ATSC-MH.

Reed Solomon (RS) frame mode.

The acceptable values are defined by `atscmh_rs_frame_mode`.

DTV_ATSCMH_RS_FRAME_ENSEMBLE

Used only on ATSC-MH.

Reed Solomon(RS) frame ensemble.

The acceptable values are defined by `atscmh_rs_frame_ensemble`.

DTV_ATSCMH_RS_CODE_MODE_PRI

Used only on ATSC-MH.

Reed Solomon (RS) code mode (primary).

The acceptable values are defined by `atscmh_rs_code_mode`.

DTV_ATSCMH_RS_CODE_MODE_SEC

Used only on ATSC-MH.

Reed Solomon (RS) code mode (secondary).

The acceptable values are defined by `atscmh_rs_code_mode`.

DTV_ATSCMH_SCCC_BLOCK_MODE

Used only on ATSC-MH.

Series Concatenated Convolutional Code Block Mode.

The acceptable values are defined by `atscmh_sccc_block_mode`.

DTV_ATSCMH_SCCC_CODE_MODE_A

Used only on ATSC-MH.

Series Concatenated Convolutional Code Rate.

The acceptable values are defined by `atscmh_sccc_code_mode`.

DTV_ATSCMH_SCCC_CODE_MODE_B

Used only on ATSC-MH.

Series Concatenated Convolutional Code Rate.

Possible values are the same as documented on enum `atscmh_sccc_code_mode`.

DTV_ATSCMH_SCCC_CODE_MODE_C

Used only on ATSC-MH.

Series Concatenated Convolutional Code Rate.

Possible values are the same as documented on enum `atscmh_sccc_code_mode`.

DTV_ATSCMH_SCCC_CODE_MODE_D

Used only on ATSC-MH.

Series Concatenated Convolutional Code Rate.

Possible values are the same as documented on enum `atscmh_sccc_code_mode`.

DTV_API_VERSION

Returns the major/minor version of the Digital TV API

DTV_CODE_RATE_HP

Used on terrestrial transmissions.

The acceptable values are defined by `fe_transmit_mode`.

DTV_CODE_RATE_LP

Used on terrestrial transmissions.

The acceptable values are defined by `fe_transmit_mode`.

DTV_GUARD_INTERVAL

The acceptable values are defined by `fe_guard_interval`.

Note:

1. If `DTV_GUARD_INTERVAL` is set the `GUARD_INTERVAL_AUTO` the hardware will try to find the correct guard interval (if capable) and will use TMCC to fill in the missing parameters.
 2. Intervals `GUARD_INTERVAL_1_128`, `GUARD_INTERVAL_19_128` and `GUARD_INTERVAL_19_256` are used only for DVB-T2 at present.
 3. Intervals `GUARD_INTERVAL_PN420`, `GUARD_INTERVAL_PN595` and `GUARD_INTERVAL_PN945` are used only for DMTB at the present. On such standard, only those intervals and `GUARD_INTERVAL_AUTO` are valid.
-

DTV_TRANSMISSION_MODE

Used only on OFDM-based standards, e. g. DVB-T/T2, ISDB-T, DTMB.

Specifies the FFT size (with corresponds to the approximate number of carriers) used by the standard.

The acceptable values are defined by `fe_transmit_mode`.

Note:

1. ISDB-T supports three carrier/symbol-size: 8K, 4K, 2K. It is called **mode** on such standard, and are numbered from 1 to 3:

Mode	FFT size	Transmission mode
1	2K	<code>TRANSMISSION_MODE_2K</code>
2	4K	<code>TRANSMISSION_MODE_4K</code>
3	8K	<code>TRANSMISSION_MODE_8K</code>

2. If `DTV_TRANSMISSION_MODE` is set the `TRANSMISSION_MODE_AUTO` the hardware will try to find the correct FFT-size (if capable) and will use TMCC to fill in the missing parameters.
 3. DVB-T specifies 2K and 8K as valid sizes.
 4. DVB-T2 specifies 1K, 2K, 4K, 8K, 16K and 32K.
 5. DTMB specifies C1 and C3780.
-

DTV_HIERARCHY

Used only on DVB-T and DVB-T2.

Frontend hierarchy.

The acceptable values are defined by `fe_hierarchy`.

DTV_STREAM_ID

Used on DVB-S2, DVB-T2 and ISDB-S.

DVB-S2, DVB-T2 and ISDB-S support the transmission of several streams on a single transport stream. This property enables the digital TV driver to handle substream filtering, when supported by the hardware. By default, substream filtering is disabled.

For DVB-S2 and DVB-T2, the valid substream id range is from 0 to 255.

For ISDB, the valid substream id range is from 1 to 65535.

To disable it, you should use the special macro `NO_STREAM_ID_FILTER`.

Note: any value outside the id range also disables filtering.

DTV_DVBT2_PLP_ID_LEGACY

Obsolete, replaced with `DTV_STREAM_ID`.

DTV_ENUM_DELSYS

A Multi standard frontend needs to advertise the delivery systems provided. Applications need to enumerate the provided delivery systems, before using any other operation with the frontend. Prior to it's introduction, `FE_GET_INFO` was used to determine a frontend type. A frontend which provides more than a single delivery system, `FE_GET_INFO` doesn't help much. Applications which intends to use a multistandard frontend must enumerate the delivery systems associated with it, rather than trying to use `FE_GET_INFO`. In the case of a legacy frontend, the result is just the same as with `FE_GET_INFO`, but in a more structured format

The acceptable values are defined by `fe_delivery_system`.

DTV_INTERLEAVING

Time interleaving to be used.

The acceptable values are defined by `fe_interleaving`.

DTV_LNA

Low-noise amplifier.

Hardware might offer controllable LNA which can be set manually using that parameter. Usually LNA could be found only from terrestrial devices if at all.

Possible values: 0, 1, LNA_AUTO

0, LNA off

1, LNA on

use the special macro LNA_AUTO to set LNA auto

DTV_SCRAMBLING_SEQUENCE_INDEX

Used on DVB-S2.

This 18 bit field, when present, carries the index of the DVB-S2 physical layer scrambling sequence as defined in clause 5.5.4 of EN 302 307. There is no explicit signalling method to convey scrambling sequence index to the receiver. If S2 satellite delivery system descriptor is available it can be used to read the scrambling sequence index (EN 300 468 table 41).

By default, gold scrambling sequence index 0 is used.

The valid scrambling sequence index range is from 0 to 262142.

Frontend statistics indicators

The values are returned via `dtv_property.stat`. If the property is supported, `dtv_property.stat.len` is bigger than zero.

For most delivery systems, `dtv_property.stat.len` will be 1 if the stats is supported, and the properties will return a single value for each parameter.

It should be noted, however, that new OFDM delivery systems like ISDB can use different modulation types for each group of carriers. On such standards, up to 3 groups of statistics can be provided, and `dtv_property.stat.len` is updated to reflect the “global” metrics, plus one metric per each carrier group (called “layer” on ISDB).

So, in order to be consistent with other delivery systems, the first value at `dtv_property.stat.dtv_stats` array refers to the global metric. The other elements of the array represent each layer, starting from layer A(index 1), layer B (index 2) and so on.

The number of filled elements are stored at `dtv_property.stat.len`.

Each element of the `dtv_property.stat.dtv_stats` array consists on two elements:

- `svalue` or `uvalue`, where `svalue` is for signed values of the measure (dB measures) and `uvalue` is for unsigned values (counters, relative scale)
- `scale` - Scale for the value. It can be:
 - `FE_SCALE_NOT_AVAILABLE` - The parameter is supported by the frontend, but it was not possible to collect it (could be a transitory or permanent condition)
 - `FE_SCALE_DECIBEL` - parameter is a signed value, measured in 1/1000 dB

- `FE_SCALE_RELATIVE` - parameter is a unsigned value, where 0 means 0% and 65535 means 100%.
- `FE_SCALE_COUNTER` - parameter is a unsigned value that counts the occurrence of an event, like bit error, block error, or lapsed time.

DTV_STAT_SIGNAL_STRENGTH

Indicates the signal strength level at the analog part of the tuner or of the demod.

Possible scales for this metric are:

- `FE_SCALE_NOT_AVAILABLE` - it failed to measure it, or the measurement was not complete yet.
- `FE_SCALE_DECIBEL` - signal strength is in 0.001 dBm units, power measured in miliwatts. This value is generally negative.
- `FE_SCALE_RELATIVE` - The frontend provides a 0% to 100% measurement for power (actually, 0 to 65535).

DTV_STAT_CNR

Indicates the Signal to Noise ratio for the main carrier.

Possible scales for this metric are:

- `FE_SCALE_NOT_AVAILABLE` - it failed to measure it, or the measurement was not complete yet.
- `FE_SCALE_DECIBEL` - Signal/Noise ratio is in 0.001 dB units.
- `FE_SCALE_RELATIVE` - The frontend provides a 0% to 100% measurement for Signal/Noise (actually, 0 to 65535).

DTV_STAT_PRE_ERROR_BIT_COUNT

Measures the number of bit errors before the forward error correction (FEC) on the inner coding block (before Viterbi, LDPC or other inner code).

This measure is taken during the same interval as `DTV_STAT_PRE_TOTAL_BIT_COUNT`.

In order to get the BER (Bit Error Rate) measurement, it should be divided by [*DTV_STAT_PRE_TOTAL_BIT_COUNT*](#).

This measurement is monotonically increased, as the frontend gets more bit count measurements. The frontend may reset it when a channel/transponder is tuned.

Possible scales for this metric are:

- `FE_SCALE_NOT_AVAILABLE` - it failed to measure it, or the measurement was not complete yet.
- `FE_SCALE_COUNTER` - Number of error bits counted before the inner coding.

DTV_STAT_PRE_TOTAL_BIT_COUNT

Measures the amount of bits received before the inner code block, during the same period as *DTV_STAT_PRE_ERROR_BIT_COUNT* measurement was taken.

It should be noted that this measurement can be smaller than the total amount of bits on the transport stream, as the frontend may need to manually restart the measurement, losing some data between each measurement interval.

This measurement is monotonically increased, as the frontend gets more bit count measurements. The frontend may reset it when a channel/transponder is tuned.

Possible scales for this metric are:

- FE_SCALE_NOT_AVAILABLE - it failed to measure it, or the measurement was not complete yet.
- FE_SCALE_COUNTER - Number of bits counted while measuring *DTV_STAT_PRE_ERROR_BIT_COUNT*.

DTV_STAT_POST_ERROR_BIT_COUNT

Measures the number of bit errors after the forward error correction (FEC) done by inner code block (after Viterbi, LDPC or other inner code).

This measure is taken during the same interval as *DTV_STAT_POST_TOTAL_BIT_COUNT*.

In order to get the BER (Bit Error Rate) measurement, it should be divided by *DTV_STAT_POST_TOTAL_BIT_COUNT*.

This measurement is monotonically increased, as the frontend gets more bit count measurements. The frontend may reset it when a channel/transponder is tuned.

Possible scales for this metric are:

- FE_SCALE_NOT_AVAILABLE - it failed to measure it, or the measurement was not complete yet.
- FE_SCALE_COUNTER - Number of error bits counted after the inner coding.

DTV_STAT_POST_TOTAL_BIT_COUNT

Measures the amount of bits received after the inner coding, during the same period as *DTV_STAT_POST_ERROR_BIT_COUNT* measurement was taken.

It should be noted that this measurement can be smaller than the total amount of bits on the transport stream, as the frontend may need to manually restart the measurement, losing some data between each measurement interval.

This measurement is monotonically increased, as the frontend gets more bit count measurements. The frontend may reset it when a channel/transponder is tuned.

Possible scales for this metric are:

- FE_SCALE_NOT_AVAILABLE - it failed to measure it, or the measurement was not complete yet.

- `FE_SCALE_COUNTER` - Number of bits counted while measuring `DTV_STAT_POST_ERROR_BIT_COUNT`.

DTV_STAT_ERROR_BLOCK_COUNT

Measures the number of block errors after the outer forward error correction coding (after Reed-Solomon or other outer code).

This measurement is monotonically increased, as the frontend gets more bit count measurements. The frontend may reset it when a channel/transponder is tuned.

Possible scales for this metric are:

- `FE_SCALE_NOT_AVAILABLE` - it failed to measure it, or the measurement was not complete yet.
- `FE_SCALE_COUNTER` - Number of error blocks counted after the outer coding.

DTV-STAT_TOTAL_BLOCK_COUNT

Measures the total number of blocks received during the same period as `DTV_STAT_ERROR_BLOCK_COUNT` measurement was taken.

It can be used to calculate the PER indicator, by dividing `DTV_STAT_ERROR_BLOCK_COUNT` by `DTV-STAT_TOTAL_BLOCK_COUNT`.

Possible scales for this metric are:

- `FE_SCALE_NOT_AVAILABLE` - it failed to measure it, or the measurement was not complete yet.
- `FE_SCALE_COUNTER` - Number of blocks counted while measuring `DTV_STAT_ERROR_BLOCK_COUNT`.

Properties used on terrestrial delivery systems

DVB-T delivery system

The following parameters are valid for DVB-T:

- `DTV_API_VERSION`
- `DTV_DELIVERY_SYSTEM`
- `DTV_TUNE`
- `DTV_CLEAR`
- `DTV_FREQUENCY`
- `DTV_MODULATION`
- `DTV_BANDWIDTH_HZ`
- `DTV_INVERSION`
- `DTV_CODE_RATE_HP`

- *DTV_CODE_RATE_LP*
- *DTV_GUARD_INTERVAL*
- *DTV_TRANSMISSION_MODE*
- *DTV_HIERARCHY*
- *DTV_LNA*

In addition, the *DTV QoS statistics* are also valid.

DVB-T2 delivery system

DVB-T2 support is currently in the early stages of development, so expect that this section may grow and become more detailed with time.

The following parameters are valid for DVB-T2:

- *DTV_API_VERSION*
- *DTV_DELIVERY_SYSTEM*
- *DTV_TUNE*
- *DTV_CLEAR*
- *DTV_FREQUENCY*
- *DTV_MODULATION*
- *DTV_BANDWIDTH_HZ*
- *DTV_INVERSION*
- *DTV_CODE_RATE_HP*
- *DTV_CODE_RATE_LP*
- *DTV_GUARD_INTERVAL*
- *DTV_TRANSMISSION_MODE*
- *DTV_HIERARCHY*
- *DTV_STREAM_ID*
- *DTV_LNA*

In addition, the *DTV QoS statistics* are also valid.

ISDB-T delivery system

This ISDB-T/ISDB-Tsb API extension should reflect all information needed to tune any ISDB-T/ISDB-Tsb hardware. Of course it is possible that some very sophisticated devices won't need certain parameters to tune.

The information given here should help application writers to know how to handle ISDB-T and ISDB-Tsb hardware using the Linux Digital TV API.

The details given here about ISDB-T and ISDB-Tsb are just enough to basically show the dependencies between the needed parameter values, but surely some information is left out. For more detailed information see the following documents:

ARIB STD-B31 - “Transmission System for Digital Terrestrial Television Broadcasting” and
ARIB TR-B14 - “Operational Guidelines for Digital Terrestrial Television Broadcasting”.

In order to understand the ISDB specific parameters, one has to have some knowledge the channel structure in ISDB-T and ISDB-Tsb. I.e. it has to be known to the reader that an ISDB-T channel consists of 13 segments, that it can have up to 3 layer sharing those segments, and things like that.

The following parameters are valid for ISDB-T:

- *DTV_API_VERSION*
- *DTV_DELIVERY_SYSTEM*
- *DTV_TUNE*
- *DTV_CLEAR*
- *DTV_FREQUENCY*
- *DTV_BANDWIDTH_HZ*
- *DTV_INVERSION*
- *DTV_GUARD_INTERVAL*
- *DTV_TRANSMISSION_MODE*
- *DTV_ISDBT_LAYER_ENABLED*
- *DTV_ISDBT_PARTIAL_RECEPTION*
- *DTV_ISDBT_SOUND_BROADCASTING*
- *DTV_ISDBT_SB_SUBCHANNEL_ID*
- *DTV_ISDBT_SB_SEGMENT_IDX*
- *DTV_ISDBT_SB_SEGMENT_COUNT*
- *DTV_ISDBT_LAYERA_FEC*
- *DTV_ISDBT_LAYERA_MODULATION*
- *DTV_ISDBT_LAYERA_SEGMENT_COUNT*
- *DTV_ISDBT_LAYERA_TIME_INTERLEAVING*
- *DTV_ISDBT_LAYERB_FEC*
- *DTV_ISDBT_LAYERB_MODULATION*
- *DTV_ISDBT_LAYERB_SEGMENT_COUNT*
- *DTV_ISDBT_LAYERB_TIME_INTERLEAVING*
- *DTV_ISDBT_LAYERC_FEC*
- *DTV_ISDBT_LAYERC_MODULATION*
- *DTV_ISDBT_LAYERC_SEGMENT_COUNT*

- *DTV_ISDBT_LAYERC_TIME_INTERLEAVING*

In addition, the *DTV QoS statistics* are also valid.

ATSC delivery system

The following parameters are valid for ATSC:

- *DTV_API_VERSION*
- *DTV_DELIVERY_SYSTEM*
- *DTV_TUNE*
- *DTV_CLEAR*
- *DTV_FREQUENCY*
- *DTV_MODULATION*
- *DTV_BANDWIDTH_HZ*

In addition, the *DTV QoS statistics* are also valid.

ATSC-MH delivery system

The following parameters are valid for ATSC-MH:

- *DTV_API_VERSION*
- *DTV_DELIVERY_SYSTEM*
- *DTV_TUNE*
- *DTV_CLEAR*
- *DTV_FREQUENCY*
- *DTV_BANDWIDTH_HZ*
- *DTV_ATSCMH_FIC_VER*
- *DTV_ATSCMH_PARADE_ID*
- *DTV_ATSCMH_NOG*
- *DTV_ATSCMH_TNOG*
- *DTV_ATSCMH_SGN*
- *DTV_ATSCMH_PRC*
- *DTV_ATSCMH_RS_FRAME_MODE*
- *DTV_ATSCMH_RS_FRAME_ENSEMBLE*
- *DTV_ATSCMH_RS_CODE_MODE_PRI*
- *DTV_ATSCMH_RS_CODE_MODE_SEC*
- *DTV_ATSCMH_SCCC_BLOCK_MODE*
- *DTV_ATSCMH_SCCC_CODE_MODE_A*

- *DTV_ATSCMH_SCCC_CODE_MODE_B*
- *DTV_ATSCMH_SCCC_CODE_MODE_C*
- *DTV_ATSCMH_SCCC_CODE_MODE_D*

In addition, the *DTV QoS statistics* are also valid.

DTMB delivery system

The following parameters are valid for DTMB:

- *DTV_API_VERSION*
- *DTV_DELIVERY_SYSTEM*
- *DTV_TUNE*
- *DTV_CLEAR*
- *DTV_FREQUENCY*
- *DTV_MODULATION*
- *DTV_BANDWIDTH_HZ*
- *DTV_INVERSION*
- *DTV_INNER_FEC*
- *DTV_GUARD_INTERVAL*
- *DTV_TRANSMISSION_MODE*
- *DTV_INTERLEAVING*
- *DTV_LNA*

In addition, the *DTV QoS statistics* are also valid.

Properties used on cable delivery systems

DVB-C delivery system

The DVB-C Annex-A is the widely used cable standard. Transmission uses QAM modulation.

The DVB-C Annex-C is optimized for 6MHz, and is used in Japan. It supports a subset of the Annex A modulation types, and a roll-off of 0.13, instead of 0.15

The following parameters are valid for DVB-C Annex A/C:

- *DTV_API_VERSION*
- *DTV_DELIVERY_SYSTEM*
- *DTV_TUNE*
- *DTV_CLEAR*
- *DTV_FREQUENCY*
- *DTV_MODULATION*

- *DTV_INVERSION*
- *DTV_SYMBOL_RATE*
- *DTV_INNER_FEC*
- *DTV_LNA*

In addition, the *DTV QoS statistics* are also valid.

DVB-C Annex B delivery system

The DVB-C Annex-B is only used on a few Countries like the United States.

The following parameters are valid for DVB-C Annex B:

- *DTV_API_VERSION*
- *DTV_DELIVERY_SYSTEM*
- *DTV_TUNE*
- *DTV_CLEAR*
- *DTV_FREQUENCY*
- *DTV_MODULATION*
- *DTV_INVERSION*
- *DTV_LNA*

In addition, the *DTV QoS statistics* are also valid.

Properties used on satellite delivery systems

DVB-S delivery system

The following parameters are valid for DVB-S:

- *DTV_API_VERSION*
- *DTV_DELIVERY_SYSTEM*
- *DTV_TUNE*
- *DTV_CLEAR*
- *DTV_FREQUENCY*
- *DTV_INVERSION*
- *DTV_SYMBOL_RATE*
- *DTV_INNER_FEC*
- *DTV_VOLTAGE*
- *DTV_TONE*

In addition, the *DTV QoS statistics* are also valid.

Future implementations might add those two missing parameters:

- *DTV_DISEQC_MASTER*
- *DTV_DISEQC_SLAVE_REPLY*

DVB-S2 delivery system

In addition to all parameters valid for DVB-S, DVB-S2 supports the following parameters:

- *DTV_MODULATION*
- *DTV_PILOT*
- *DTV_ROLLOFF*
- *DTV_STREAM_ID*
- *DTV_SCRAMBLING_SEQUENCE_INDEX*

In addition, the *DTV QoS statistics* are also valid.

Turbo code delivery system

In addition to all parameters valid for DVB-S, turbo code supports the following parameters:

- *DTV_MODULATION*

ISDB-S delivery system

The following parameters are valid for ISDB-S:

- *DTV_API_VERSION*
- *DTV_DELIVERY_SYSTEM*
- *DTV_TUNE*
- *DTV_CLEAR*
- *DTV_FREQUENCY*
- *DTV_INVERSION*
- *DTV_SYMBOL_RATE*
- *DTV_INNER_FEC*
- *DTV_VOLTAGE*
- *DTV_STREAM_ID*

Frontend uAPI data types

enum **fe_caps**

Frontend capabilities

Constants

- FE_IS_STUPID** There's something wrong at the frontend, and it can't report its capabilities.
- FE_CAN_INVERSION_AUTO** Can auto-detect frequency spectral band inversion
- FE_CAN_FEC_1_2** Supports FEC 1/2
- FE_CAN_FEC_2_3** Supports FEC 2/3
- FE_CAN_FEC_3_4** Supports FEC 3/4
- FE_CAN_FEC_4_5** Supports FEC 4/5
- FE_CAN_FEC_5_6** Supports FEC 5/6
- FE_CAN_FEC_6_7** Supports FEC 6/7
- FE_CAN_FEC_7_8** Supports FEC 7/8
- FE_CAN_FEC_8_9** Supports FEC 8/9
- FE_CAN_FEC_AUTO** Can auto-detect FEC
- FE_CAN_QPSK** Supports QPSK modulation
- FE_CAN_QAM_16** Supports 16-QAM modulation
- FE_CAN_QAM_32** Supports 32-QAM modulation
- FE_CAN_QAM_64** Supports 64-QAM modulation
- FE_CAN_QAM_128** Supports 128-QAM modulation
- FE_CAN_QAM_256** Supports 256-QAM modulation
- FE_CAN_QAM_AUTO** Can auto-detect QAM modulation
- FE_CAN_TRANSMISSION_MODE_AUTO** Can auto-detect transmission mode
- FE_CAN_BANDWIDTH_AUTO** Can auto-detect bandwidth
- FE_CAN_GUARD_INTERVAL_AUTO** Can auto-detect guard interval
- FE_CAN_HIERARCHY_AUTO** Can auto-detect hierarchy
- FE_CAN_8VSB** Supports 8-VSB modulation
- FE_CAN_16VSB** Supporta 16-VSB modulation
- FE_HAS_EXTENDED_CAPS** Unused
- FE_CAN_MULTISTREAM** Supports multistream filtering
- FE_CAN_TURBO_FEC** Supports "turbo FEC" modulation
- FE_CAN_2G_MODULATION** Supports "2nd generation" modulation, e. g. DVB-S2, DVB-T2, DVB-C2
- FE_NEEDS_BENDING** Unused

FE_CAN_RECOVER Can recover from a cable unplug automatically

FE_CAN_MUTE_TS Can stop spurious TS data output

struct **dvb_frontend_info**
Frontend properties and capabilities

Definition

```
struct dvb_frontend_info {
    char name[128];
    enum fe_type type;
    __u32 frequency_min;
    __u32 frequency_max;
    __u32 frequency_stepsize;
    __u32 frequency_tolerance;
    __u32 symbol_rate_min;
    __u32 symbol_rate_max;
    __u32 symbol_rate_tolerance;
    __u32 notifier_delay;
    enum fe_caps caps;
};
```

Members

name Name of the frontend

type **DEPRECATED**. Should not be used on modern programs, as a frontend may have more than one type. In order to get the support types of a given frontend, use `DTV_ENUM_DELSYS` instead.

frequency_min Minimal frequency supported by the frontend.

frequency_max Maximal frequency supported by the frontend.

frequency_stepsize All frequencies are multiple of this value.

frequency_tolerance Frequency tolerance.

symbol_rate_min Minimal symbol rate, in bauds (for Cable/Satellite systems).

symbol_rate_max Maximal symbol rate, in bauds (for Cable/Satellite systems).

symbol_rate_tolerance Maximal symbol rate tolerance, in ppm (for Cable/Satellite systems).

notifier_delay **DEPRECATED**. Not used by any driver.

caps Capabilities supported by the frontend, as specified in enum `fe_caps`.

Description

struct **dvb_diseqc_master_cmd**
DiSEqC master command

Definition

```
struct dvb_diseqc_master_cmd {
    __u8 msg[6];
    __u8 msg_len;
};
```

Members

msg

DiSEqC message to be sent. It contains a 3 bytes header with: framing + address + command, and an optional argument of up to 3 bytes of data.

msg_len

Length of the DiSEqC message. Valid values are 3 to 6.

Description

Check out the DiSEqC bus spec available on <http://www.eutelsat.org/> for the possible messages that can be used.

struct **dvb_diseqc_slave_reply**

DiSEqC received data

Definition

```
struct dvb_diseqc_slave_reply {
    __u8 msg[4];
    __u8 msg_len;
    int timeout;
};
```

Members

msg

DiSEqC message buffer to store a message received via DiSEqC. It contains one byte header with: framing and an optional argument of up to 3 bytes of data.

msg_len

Length of the DiSEqC message. Valid values are 0 to 4, where 0 means no message.

timeout

Return from ioctl after timeout ms with errorcode when no message was received.

Description

Check out the DiSEqC bus spec available on <http://www.eutelsat.org/> for the possible messages that can be used.

enum **fe_sec_voltage**

DC Voltage used to feed the LNBf

Constants

SEC_VOLTAGE_13 Output 13V to the LNBf

SEC_VOLTAGE_18 Output 18V to the LNBf

SEC_VOLTAGE_OFF Don't feed the LNBf with a DC voltage

enum **fe_sec_tone_mode**

Type of tone to be send to the LNBf.

Constants

SEC_TONE_ON Sends a 22kHz tone burst to the antenna.

SEC_TONE_OFF Don't send a 22kHz tone to the antenna (except if the FE_DISEQC_* ioctls are called).

enum **fe_sec_mini_cmd**
Type of mini burst to be sent

Constants

SEC_MINI_A Sends a mini-DiSEqC 22kHz '0' Tone Burst to select satellite-A

SEC_MINI_B Sends a mini-DiSEqC 22kHz '1' Data Burst to select satellite-B

enum **fe_status**
Enumerates the possible frontend status.

Constants

FE_NONE The frontend doesn't have any kind of lock. That's the initial frontend status

FE_HAS_SIGNAL Has found something above the noise level.

FE_HAS_CARRIER Has found a signal.

FE_HAS_VITERBI FEC inner coding (Viterbi, LDPC or other inner code). is stable.

FE_HAS_SYNC Synchronization bytes was found.

FE_HAS_LOCK Digital TV were locked and everything is working.

FE_TIMEDOUT Fo lock within the last about 2 seconds.

FE_REINIT Frontend was reinitialized, application is recommended to reset DiSEqC, tone and parameters.

enum **fe_spectral_inversion**
Type of inversion band

Constants

INVERSION_OFF Don't do spectral band inversion.

INVERSION_ON Do spectral band inversion.

INVERSION_AUTO Autodetect spectral band inversion.

Description

This parameter indicates if spectral inversion should be presumed or not. In the automatic setting (**INVERSION_AUTO**) the hardware will try to figure out the correct setting by itself. If the hardware doesn't support, the `dvb_frontend` will try to lock at the carrier first with inversion off. If it fails, it will try to enable inversion.

enum **fe_code_rate**
Type of Forward Error Correction (FEC)

Constants

FEC_NONE No Forward Error Correction Code

FEC_1_2 Forward Error Correction Code 1/2

FEC_2_3 Forward Error Correction Code 2/3

FEC_3_4 Forward Error Correction Code 3/4

FEC_4_5 Forward Error Correction Code 4/5
FEC_5_6 Forward Error Correction Code 5/6
FEC_6_7 Forward Error Correction Code 6/7
FEC_7_8 Forward Error Correction Code 7/8
FEC_8_9 Forward Error Correction Code 8/9
FEC_AUTO Autodetect Error Correction Code
FEC_3_5 Forward Error Correction Code 3/5
FEC_9_10 Forward Error Correction Code 9/10
FEC_2_5 Forward Error Correction Code 2/5

Description

Please note that not all FEC types are supported by a given standard.

enum **fe_modulation**

Type of modulation/constellation

Constants

QPSK QPSK modulation
QAM_16 16-QAM modulation
QAM_32 32-QAM modulation
QAM_64 64-QAM modulation
QAM_128 128-QAM modulation
QAM_256 256-QAM modulation
QAM_AUTO Autodetect QAM modulation
VSB_8 8-VSB modulation
VSB_16 16-VSB modulation
PSK_8 8-PSK modulation
APSK_16 16-APSK modulation
APSK_32 32-APSK modulation
DQPSK DQPSK modulation
QAM_4_NR 4-QAM-NR modulation

Description

Please note that not all modulations are supported by a given standard.

enum **fe_transmit_mode**

Transmission mode

Constants

TRANSMISSION_MODE_2K
Transmission mode 2K

TRANSMISSION_MODE_8K

Transmission mode 8K

TRANSMISSION_MODE_AUTO

Autodetect transmission mode. The hardware will try to find the correct FFT-size (if capable) to fill in the missing parameters.

TRANSMISSION_MODE_4K

Transmission mode 4K

TRANSMISSION_MODE_1K

Transmission mode 1K

TRANSMISSION_MODE_16K

Transmission mode 16K

TRANSMISSION_MODE_32K

Transmission mode 32K

TRANSMISSION_MODE_C1

Single Carrier (C=1) transmission mode (DTMB only)

TRANSMISSION_MODE_C3780

Multi Carrier (C=3780) transmission mode (DTMB only)

Description

Please note that not all transmission modes are supported by a given standard.

enum **fe_guard_interval**

Guard interval

Constants

GUARD_INTERVAL_1_32 Guard interval 1/32

GUARD_INTERVAL_1_16 Guard interval 1/16

GUARD_INTERVAL_1_8 Guard interval 1/8

GUARD_INTERVAL_1_4 Guard interval 1/4

GUARD_INTERVAL_AUTO Autodetect the guard interval

GUARD_INTERVAL_1_128 Guard interval 1/128

GUARD_INTERVAL_19_128 Guard interval 19/128

GUARD_INTERVAL_19_256 Guard interval 19/256

GUARD_INTERVAL_PN420 PN length 420 (1/4)

GUARD_INTERVAL_PN595 PN length 595 (1/6)

GUARD_INTERVAL_PN945 PN length 945 (1/9)

Description

Please note that not all guard intervals are supported by a given standard.

enum **fe_hierarchy**

Hierarchy

Constants

HIERARCHY_NONE No hierarchy

HIERARCHY_1 Hierarchy 1

HIERARCHY_2 Hierarchy 2

HIERARCHY_4 Hierarchy 4

HIERARCHY_AUTO Autodetect hierarchy (if supported)

Description

Please note that not all hierarchy types are supported by a given standard.

enum **fe_interleaving**

Interleaving

Constants

INTERLEAVING_NONE No interleaving.

INTERLEAVING_AUTO Auto-detect interleaving.

INTERLEAVING_240 Interleaving of 240 symbols.

INTERLEAVING_720 Interleaving of 720 symbols.

Description

Please note that, currently, only DTMB uses it.

enum **fe_pilot**

Type of pilot tone

Constants

PILOT_ON Pilot tones enabled

PILOT_OFF Pilot tones disabled

PILOT_AUTO Autodetect pilot tones

enum **fe_rolloff**

Rolloff factor

Constants

ROLLOFF_35 Rolloff factor: $\alpha=35\%$

ROLLOFF_20 Rolloff factor: $\alpha=20\%$

ROLLOFF_25 Rolloff factor: $\alpha=25\%$

ROLLOFF_AUTO Auto-detect the rolloff factor.

Description

enum **fe_delivery_system**

Type of the delivery system

Constants**SYS_UNDEFINED**

Undefined standard. Generally, indicates an error

SYS_DVBC_ANNEX_A

Cable TV: DVB-C following ITU-T J.83 Annex A spec

SYS_DVBC_ANNEX_B

Cable TV: DVB-C following ITU-T J.83 Annex B spec (ClearQAM)

SYS_DVBT

Terrestrial TV: DVB-T

SYS_DSS

Satellite TV: DSS (not fully supported)

SYS_DVBS

Satellite TV: DVB-S

SYS_DVBS2

Satellite TV: DVB-S2

SYS_DVBH

Terrestrial TV (mobile): DVB-H (standard deprecated)

SYS_ISDBT

Terrestrial TV: ISDB-T

SYS_ISDBS

Satellite TV: ISDB-S

SYS_ISDBC

Cable TV: ISDB-C (no drivers yet)

SYS_ATSC

Terrestrial TV: ATSC

SYS_ATSCMH

Terrestrial TV (mobile): ATSC-M/H

SYS_DTMB

Terrestrial TV: DTMB

SYS_CMMA

Terrestrial TV (mobile): CMMA (not fully supported)

SYS_DAB

Digital audio: DAB (not fully supported)

SYS_DVBT2

Terrestrial TV: DVB-T2

SYS_TURBO

Satellite TV: DVB-S Turbo

SYS_DVBC_ANNEX_C

Cable TV: DVB-C following ITU-T J.83 Annex C spec

enum **atscmh_sccc_block_mode**

Type of Series Concatenated Convolutional Code Block Mode.

Constants

ATSCMH_SCCC_BLK_SEP

Separate SCCC: the SCCC outer code mode shall be set independently for each Group Region (A, B, C, D)

ATSCMH_SCCC_BLK_COMB

Combined SCCC: all four Regions shall have the same SCCC outer code mode.

ATSCMH_SCCC_BLK_RES

Reserved. Shouldn't be used.

enum **atscmh_sccc_code_mode**

Type of Series Concatenated Convolutional Code Rate.

Constants

ATSCMH_SCCC_CODE_HLF

The outer code rate of a SCCC Block is 1/2 rate.

ATSCMH_SCCC_CODE_QTR

The outer code rate of a SCCC Block is 1/4 rate.

ATSCMH_SCCC_CODE_RES

Reserved. Should not be used.

enum **atscmh_rs_frame_ensemble**

Reed Solomon(RS) frame ensemble.

Constants

ATSCMH_RSFRAME_ENS_PRI Primary Ensemble.

ATSCMH_RSFRAME_ENS_SEC Secondary Ensemble.

enum **atscmh_rs_frame_mode**

Reed Solomon (RS) frame mode.

Constants

ATSCMH_RSFRAME_PRI_ONLY

Single Frame: There is only a primary RS Frame for all Group Regions.

ATSCMH_RSFRAME_PRI_SEC

Dual Frame: There are two separate RS Frames: Primary RS Frame for Group Region A and B and Secondary RS Frame for Group Region C and D.

ATSCMH_RSFRAME_RES

Reserved. Shouldn't be used.

enum **atscmh_rs_code_mode**

Constants

ATSCMH_RSCODE_211_187 Reed Solomon code (211,187).

ATSCMH_RSCODE_223_187 Reed Solomon code (223,187).

ATSCMH_RSCODE_235_187 Reed Solomon code (235,187).

ATSCMH_RSCODE_RES Reserved. Shouldn't be used.

enum **fecap_scale_params**

scale types for the quality parameters.

Constants

FE_SCALE_NOT_AVAILABLE That QoS measure is not available. That could indicate a temporary or a permanent condition.

FE_SCALE_DECIBEL The scale is measured in 0.001 dB steps, typically used on signal measures.

FE_SCALE_RELATIVE The scale is a relative percentual measure, ranging from 0 (0%) to 0xffff (100%).

FE_SCALE_COUNTER The scale counts the occurrence of an event, like bit error, block error, lapsed time.

struct **dtv_stats**

Used for reading a DTV status property

Definition

```
struct dtv_stats {
    __u8 scale;
    union {
        __u64 uvalue;
        __s64 svalue;
    };
};
```

Members

scale

Filled with enum `fecap_scale_params` - the scale in usage for that parameter

{unnamed_union} anonymous

uvalue

unsigned integer value of the measure, used when **scale** is either `FE_SCALE_RELATIVE` or `FE_SCALE_COUNTER`.

svalue

integer value of the measure, for `FE_SCALE_DECIBEL`, used for dB measures. The unit is 0.001 dB.

Description

For most delivery systems, this will return a single value for each parameter.

It should be noticed, however, that new OFDM delivery systems like ISDB can use different modulation types for each group of carriers. On such standards, up to 8 groups of statistics can be provided, one for each carrier group (called “layer” on ISDB).

In order to be consistent with other delivery systems, the first value refers to the entire set of carriers (“global”).

scale should use the value `FE_SCALE_NOT_AVAILABLE` when the value for the entire group of carriers or from one specific layer is not provided by the hardware.

len should be filled with the latest filled status + 1.

In other words, for ISDB, those values should be filled like:

```
u.st.stat.svalue[0] = global statistics;
u.st.stat.scale[0] = FE_SCALE_DECIBEL;
u.st.stat.value[1] = layer A statistics;
u.st.stat.scale[1] = FE_SCALE_NOT_AVAILABLE (if not available);
u.st.stat.svalue[2] = layer B statistics;
u.st.stat.scale[2] = FE_SCALE_DECIBEL;
u.st.stat.svalue[3] = layer C statistics;
u.st.stat.scale[3] = FE_SCALE_DECIBEL;
u.st.len = 4;
```

```
struct dtv_fe_stats
    store Digital TV frontend statistics
```

Definition

```
struct dtv_fe_stats {
    __u8 len;
    struct dtv_stats stat[MAX_DTV_STATS];
};
```

Members

len length of the statistics - if zero, stats is disabled.

stat array with digital TV statistics.

Description

On most standards, **len** can either be 0 or 1. However, for ISDB, each layer is modulated in separate. So, each layer may have its own set of statistics. If so, `stat[0]` carries on a global value for the property. Indexes 1 to 3 means layer A to B.

```
struct dtv_property
    store one of frontend command and its value
```

Definition

```
struct dtv_property {
    __u32 cmd;
    __u32 reserved[3];
};
```

```

union {
    __u32 data;
    struct dtv_fe_stats st;
    struct {
        __u8 data[32];
        __u32 len;
        __u32 reserved1[3];
        void *reserved2;
    } buffer;
} u;
int result;
};

```

Members

cmd Digital TV command.

reserved Not used.

u Union with the values for the command.

u.data A unsigned 32 bits integer with command value.

u.st a struct `dtv_fe_stats` array of statistics.

u.buffer Struct to store bigger properties. Currently unused.

u.buffer.data an unsigned 32-bits array.

u.buffer.len number of elements of the buffer.

u.buffer.reserved1 Reserved.

u.buffer.reserved2 Reserved.

result Currently unused.

struct **dtv_properties**

a set of command/value pairs.

Definition

```

struct dtv_properties {
    __u32 num;
    struct dtv_property *props;
};

```

Members

num amount of commands stored at the struct.

props a pointer to struct `dtv_property`.

3.3.2.4 Frontend Function Calls

Digital TV frontend `open()`

Name

`fe-open` - Open a frontend device

Synopsis

```
#include <fcntl.h>
```

```
int open(const char *device_name, int flags)
```

Arguments

device_name Device to be opened.

flags Open flags. Access can either be `O_RDWR` or `O_RDONLY`.

Multiple opens are allowed with `O_RDONLY`. In this mode, only query and read ioctls are allowed.

Only one open is allowed in `O_RDWR`. In this mode, all ioctls are allowed.

When the `O_NONBLOCK` flag is given, the system calls may return `EAGAIN` error code when no data is available or when the device driver is temporarily busy.

Other flags have no effect.

Description

This system call opens a named frontend device (`/dev/dvb/adapter?/frontend?`) for subsequent use. Usually the first thing to do after a successful open is to find out the frontend type with `ioctl FE_GET_INFO`.

The device can be opened in read-only mode, which only allows monitoring of device status and statistics, or read/write mode, which allows any kind of use (e.g. performing tuning operations.)

In a system with multiple front-ends, it is usually the case that multiple devices cannot be open in read/write mode simultaneously. As long as a front-end device is opened in read/write mode, other `open()` calls in read/write mode will either fail or block, depending on whether non-blocking or blocking mode was specified. A front-end device opened in blocking mode can later be put into non-blocking mode (and vice versa) using the `F_SETFL` command of the `fcntl` system call. This is a standard system call, documented in the Linux manual page for `fcntl`. When an `open()` call has succeeded, the device will be ready for use in the specified mode. This implies that the corresponding hardware is powered up, and that other front-ends may have been powered down to make that possible.

Return Value

On success `open()` returns the new file descriptor. On error, `-1` is returned, and the `errno` variable is set appropriately.

Possible error codes are:

On success `0` is returned, and `ca_slot_info` is filled.

On error `-1` is returned, and the `errno` variable is set appropriately.

EPERM	The caller has no permission to access the device.
EBUSY	The the device driver is already in use.
EMFILE	The process already has the maximum number of files open.
ENFILE	The limit on the total number of files open on the system has been reached.

The generic error codes are described at the [Generic Error Codes](#) chapter.

Digital TV frontend close()

Name

`fe-close` - Close a frontend device

Synopsis

```
#include <unistd.h>
```

```
int close(int fd)
```

Arguments

fd File descriptor returned by `open()`.

Description

This system call closes a previously opened front-end device. After closing a front-end device, its corresponding hardware might be powered down automatically.

Return Value

On success 0 is returned.

On error -1 is returned, and the `errno` variable is set appropriately.

Generic error codes are described at the [Generic Error Codes](#) chapter.

ioctl FE_GET_INFO

Name

`FE_GET_INFO` - Query Digital TV frontend capabilities and returns information about the - frontend. This call only requires read-only access to the device.

Synopsis

FE_GET_INFO

```
int ioctl(int fd, FE_GET_INFO, struct dvb_frontend_info *argp)
```

Arguments

fd File descriptor returned by `open()`.

argp pointer to `struct dvb_frontend_info`

Description

All Digital TV frontend devices support the *ioctl FE_GET_INFO* ioctl. It is used to identify kernel devices compatible with this specification and to obtain information about driver and hardware capabilities. The ioctl takes a pointer to `dvb_frontend_info` which is filled by the driver. When the driver is not compatible with this specification the ioctl returns an error.

frontend capabilities

Capabilities describe what a frontend can do. Some capabilities are supported only on some specific frontend types.

The frontend capabilities are described at `fe_caps`.

Return Value

On success 0 is returned.

On error -1 is returned, and the `errno` variable is set appropriately.

Generic error codes are described at the [Generic Error Codes](#) chapter.

ioctl FE_READ_STATUS

Name

`FE_READ_STATUS` - Returns status information about the front-end. This call only requires -read-only access to the device

Synopsis

FE_READ_STATUS

```
int ioctl(int fd, FE_READ_STATUS, unsigned int *status)
```

Arguments

fd File descriptor returned by `open()`.

status pointer to a bitmask integer filled with the values defined by enum `fe_status`.

Description

All Digital TV frontend devices support the `FE_READ_STATUS` ioctl. It is used to check about the locking status of the frontend after being tuned. The ioctl takes a pointer to an integer where the status will be written.

Note: The size of `status` is actually `sizeof(enum fe_status)`, with varies according with the architecture. This needs to be fixed in the future.

int fe_status

The `fe_status` parameter is used to indicate the current state and/or state changes of the frontend hardware. It is produced using the enum `fe_status` values on a bitmask

Return Value

On success 0 is returned.

On error -1 is returned, and the `errno` variable is set appropriately.

Generic error codes are described at the [Generic Error Codes](#) chapter.

ioctl FE_SET_PROPERTY, FE_GET_PROPERTY

Name

`FE_SET_PROPERTY` - `FE_GET_PROPERTY` - `FE_SET_PROPERTY` sets one or more frontend properties. - `FE_GET_PROPERTY` returns one or more frontend properties.

Synopsis

FE_GET_PROPERTY

```
int ioctl(int fd, FE_GET_PROPERTY, struct dtv_properties *argp)
```

FE_SET_PROPERTY

```
int ioctl(int fd, FE_SET_PROPERTY, struct dtv_properties *argp)
```

Arguments

fd File descriptor returned by `open()`.

argp Pointer to struct `dtv_properties`.

Description

All Digital TV frontend devices support the `FE_SET_PROPERTY` and `FE_GET_PROPERTY` ioctls. The supported properties and statistics depends on the delivery system and on the device:

- `FE_SET_PROPERTY`:
 - This ioctl is used to set one or more frontend properties.
 - This is the basic command to request the frontend to tune into some frequency and to start decoding the digital TV signal.
 - This call requires read/write access to the device.

Note: At return, the values aren't updated to reflect the actual parameters used. If the actual parameters are needed, an explicit call to `FE_GET_PROPERTY` is needed.

- `FE_GET_PROPERTY`:
 - This ioctl is used to get properties and statistics from the frontend.

- No properties are changed, and statistics aren't reset.
- This call only requires read-only access to the device.

Return Value

On success 0 is returned.

On error -1 is returned, and the `errno` variable is set appropriately.

Generic error codes are described at the *Generic Error Codes* chapter.

ioctl FE_DISEQC_RESET_OVERLOAD

Name

`FE_DISEQC_RESET_OVERLOAD` - Restores the power to the antenna subsystem, if it was powered off due - to power overload.

Synopsis

`FE_DISEQC_RESET_OVERLOAD`

```
int ioctl(int fd, FE_DISEQC_RESET_OVERLOAD, NULL)
```

Arguments

fd File descriptor returned by `open()`.

Description

If the bus has been automatically powered off due to power overload, this `ioctl` call restores the power to the bus. The call requires read/write access to the device. This call has no effect if the device is manually powered off. Not all Digital TV adapters support this `ioctl`.

Return Value

On success 0 is returned.

On error -1 is returned, and the `errno` variable is set appropriately.

Generic error codes are described at the *Generic Error Codes* chapter.

ioctl FE_DISEQC_SEND_MASTER_CMD

Name

FE_DISEQC_SEND_MASTER_CMD - Sends a DiSEqC command

Synopsis

FE_DISEQC_SEND_MASTER_CMD

```
int ioctl(int fd, FE_DISEQC_SEND_MASTER_CMD, struct dvb_diseqc_master_cmd
*argp)
```

Arguments

fd File descriptor returned by `open()`.

argp pointer to `struct dvb_diseqc_master_cmd`

Description

Sends the DiSEqC command pointed by `dvb_diseqc_master_cmd` to the antenna subsystem.

Return Value

On success 0 is returned.

On error -1 is returned, and the `errno` variable is set appropriately.

Generic error codes are described at the [Generic Error Codes](#) chapter.

ioctl FE_DISEQC_RECV_SLAVE_REPLY

Name

FE_DISEQC_RECV_SLAVE_REPLY - Receives reply from a DiSEqC 2.0 command

Synopsis

FE_DISEQC_RECV_SLAVE_REPLY

```
int ioctl(int fd, FE_DISEQC_RECV_SLAVE_REPLY, struct dvb_diseqc_slave_reply
*argp)
```

Arguments

fd File descriptor returned by `open()`.

argp pointer to struct `dvb_diseqc_slave_reply`.

Description

Receives reply from a DiSEqC 2.0 command.

The received message is stored at the buffer pointed by `argp`.

Return Value

On success 0 is returned.

On error -1 is returned, and the `errno` variable is set appropriately.

Generic error codes are described at the *Generic Error Codes* chapter.

ioctl FE_DISEQC_SEND_BURST

Name

`FE_DISEQC_SEND_BURST` - Sends a 22KHz tone burst for 2x1 mini DiSEqC satellite selection.

Synopsis

`FE_DISEQC_SEND_BURST`

```
int ioctl(int fd, FE_DISEQC_SEND_BURST, enum fe_sec_mini_cmd tone)
```

Arguments

fd File descriptor returned by `open()`.

tone An integer enumerated value described at `fe_sec_mini_cmd`.

Description

This `ioctl` is used to set the generation of a 22kHz tone burst for mini DiSEqC satellite selection for 2x1 switches. This call requires read/write permissions.

It provides support for what's specified at [Digital Satellite Equipment Control \(DiSEqC\) - Simple "ToneBurst" Detection Circuit specification](#).

Return Value

On success 0 is returned.

On error -1 is returned, and the `errno` variable is set appropriately.

Generic error codes are described at the *Generic Error Codes* chapter.

ioctl FE_SET_TONE

Name

FE_SET_TONE - Sets/resets the generation of the continuous 22kHz tone.

Synopsis

FE_SET_TONE

```
int ioctl(int fd, FE_SET_TONE, enum fe_sec_tone_mode tone)
```

Arguments

fd File descriptor returned by `open()`.

tone an integer enumerated value described at `fe_sec_tone_mode`

Description

This `ioctl` is used to set the generation of the continuous 22kHz tone. This call requires read/write permissions.

Usually, satellite antenna subsystems require that the digital TV device to send a 22kHz tone in order to select between high/low band on some dual-band LNBf. It is also used to send signals to DiSEqC equipment, but this is done using the DiSEqC `ioctls`.

Attention: If more than one device is connected to the same antenna, setting a tone may interfere on other devices, as they may lose the capability of selecting the band. So, it is recommended that applications would change to `SEC_TONE_OFF` when the device is not used.

Return Value

On success 0 is returned.

On error -1 is returned, and the `errno` variable is set appropriately.

Generic error codes are described at the *Generic Error Codes* chapter.

ioctl FE_SET_VOLTAGE

Name

FE_SET_VOLTAGE - Allow setting the DC level sent to the antenna subsystem.

Synopsis

FE_SET_VOLTAGE

```
int ioctl(int fd, FE_SET_VOLTAGE, enum fe_sec_voltage voltage)
```

Arguments

fd File descriptor returned by `open()`.

voltage an integer enumerated value described at `fe_sec_voltage`

Description

This `ioctl` allows to set the DC voltage level sent through the antenna cable to 13V, 18V or off.

Usually, a satellite antenna subsystems require that the digital TV device to send a DC voltage to feed power to the LNBf. Depending on the LNBf type, the polarization or the intermediate frequency (IF) of the LNBf can controlled by the voltage level. Other devices (for example, the ones that implement DISEqC and multipoint LNBf's don't need to control the voltage level, provided that either 13V or 18V is sent to power up the LNBf.

Attention: if more than one device is connected to the same antenna, setting a voltage level may interfere on other devices, as they may lose the capability of setting polarization or IF. So, on those cases, setting the voltage to `SEC_VOLTAGE_OFF` while the device is not is used is recommended.

Return Value

On success 0 is returned.

On error -1 is returned, and the `errno` variable is set appropriately.

Generic error codes are described at the [Generic Error Codes](#) chapter.

ioctl FE_ENABLE_HIGH_LNB_VOLTAGE

Name

`FE_ENABLE_HIGH_LNB_VOLTAGE` - Select output DC level between normal LNBf voltages or higher LNBf - voltages.

Synopsis

`FE_ENABLE_HIGH_LNB_VOLTAGE`

```
int ioctl(int fd, FE_ENABLE_HIGH_LNB_VOLTAGE, unsigned int high)
```

Arguments

fd File descriptor returned by `open()`.

high Valid flags:

- 0 - normal 13V and 18V.
- >0 - enables slightly higher voltages instead of 13/18V, in order to compensate for long antenna cables.

Description

Select output DC level between normal LNBf voltages or higher LNBf voltages between 0 (normal) or a value greater than 0 for higher voltages.

Return Value

On success 0 is returned.

On error -1 is returned, and the `errno` variable is set appropriately.

Generic error codes are described at the [Generic Error Codes](#) chapter.

ioctl FE_SET_FRONTEND_TUNE_MODE

Name

FE_SET_FRONTEND_TUNE_MODE - Allow setting tuner mode flags to the frontend.

Synopsis

FE_SET_FRONTEND_TUNE_MODE

```
int ioctl(int fd, FE_SET_FRONTEND_TUNE_MODE, unsigned int flags)
```

Arguments

fd File descriptor returned by `open()`.

flags Valid flags:

- 0 - normal tune mode
- FE_TUNE_MODE_ONESHOT - When set, this flag will disable any zigzagging or other “normal” tuning behaviour. Additionally, there will be no automatic monitoring of the lock status, and hence no frontend events will be generated. If a frontend device is closed, this flag will be automatically turned off when the device is reopened read-write.

Description

Allow setting tuner mode flags to the frontend, between 0 (normal) or FE_TUNE_MODE_ONESHOT mode

Return Value

On success 0 is returned.

On error -1 is returned, and the `errno` variable is set appropriately.

Generic error codes are described at the [Generic Error Codes](#) chapter.

3.3.3 Digital TV Demux Device

The Digital TV demux device controls the MPEG-TS filters for the digital TV. If the driver and hardware supports, those filters are implemented at the hardware. Otherwise, the Kernel provides a software emulation.

It can be accessed through `/dev/adapter?/demux?`. Data types and ioctl definitions can be accessed by including `linux/dvb/dmx.h` in your application.

3.3.3.1 Demux Data Types

enum **dmx_output**

Output for the demux.

Constants

DMX_OUT_DECODER

Streaming directly to decoder.

DMX_OUT_TAP

Output going to a memory buffer (to be retrieved via the read command). Delivers the stream output to the demux device on which the ioctl is called.

DMX_OUT_TS_TAP

Output multiplexed into a new TS (to be retrieved by reading from the logical DVR device). Routes output to the logical DVR device `/dev/dvb/adapter?/dvr?`, which delivers a TS multiplexed from all filters for which **DMX_OUT_TS_TAP** was specified.

DMX_OUT_TSDEMUX_TAP

Like **DMX_OUT_TS_TAP** but retrieved from the DMX device.

enum **dmx_input**

Input from the demux.

Constants

DMX_IN_FRONTEND Input from a front-end device.

DMX_IN_DVR Input from the logical DVR device.

enum **dmx_ts_pes**

type of the PES filter.

Constants

DMX_PES_AUDI00 first audio PID. Also referred as **DMX_PES_AUDIO**.

DMX_PES_VIDE00 first video PID. Also referred as **DMX_PES_VIDEO**.

DMX_PES_TELETEXT0 first teletext PID. Also referred as **DMX_PES_TELETEXT**.

DMX_PES_SUBTITLE0 first subtitle PID. Also referred as **DMX_PES_SUBTITLE**.

DMX_PES_PCR0 first Program Clock Reference PID. Also referred as **DMX_PES_PCR**.

DMX_PES_AUDI01 second audio PID.

DMX_PES_VIDE01 second video PID.

DMX_PES_TELETEXT1 second teletext PID.

DMX_PES_SUBTITLE1 second subtitle PID.

DMX_PES_PCR1 second Program Clock Reference PID.

DMX_PES_AUDI02 third audio PID.

DMX_PES_VIDE02 third video PID.

DMX_PES_TELETEXT2 third teletext PID.

DMX_PES_SUBTITLE2 third subtitle PID.

DMX_PES_PCR2 third Program Clock Reference PID.

DMX_PES_AUDIO3 fourth audio PID.

DMX_PES_VIDEO3 fourth video PID.

DMX_PES_TELETEXT3 fourth teletext PID.

DMX_PES_SUBTITLE3 fourth subtitle PID.

DMX_PES_PCR3 fourth Program Clock Reference PID.

DMX_PES_OTHER any other PID.

struct **dmx_filter**

Specifies a section header filter.

Definition

```
struct dmx_filter {
    __u8 filter[DMX_FILTER_SIZE];
    __u8 mask[DMX_FILTER_SIZE];
    __u8 mode[DMX_FILTER_SIZE];
};
```

Members

filter bit array with bits to be matched at the section header.

mask bits that are valid at the filter bit array.

mode mode of match: if bit is zero, it will match if equal (positive match); if bit is one, it will match if the bit is negated.

Note

All arrays in this struct have a size of `DMX_FILTER_SIZE` (16 bytes).

struct **dmx_sct_filter_params**

Specifies a section filter.

Definition

```
struct dmx_sct_filter_params {
    __u16 pid;
    struct dmx_filter filter;
    __u32 timeout;
    __u32 flags;
#define DMX_CHECK_CRC          1;
#define DMX_ONESHOT           2;
#define DMX_IMMEDIATE_START 4;
};
```

Members

pid PID to be filtered.

filter section header filter, as defined by *struct dmx_filter*.

timeout maximum time to filter, in milliseconds.

flags extra flags for the section filter.

Description

Carries the configuration for a MPEG-TS section filter.

The **flags** can be:

- `DMX_CHECK_CRC` - only deliver sections where the CRC check succeeded;
- `DMX_ONESHOT` - disable the section filter after one section has been delivered;
- `DMX_IMMEDIATE_START` - Start filter immediately without requiring a `DMX_START`.

struct **dmx_pes_filter_params**

Specifies Packetized Elementary Stream (PES) filter parameters.

Definition

```
struct dmx_pes_filter_params {
    __u16 pid;
    enum dmx_input input;
    enum dmx_output output;
    enum dmx_ts_pes pes_type;
    __u32 flags;
};
```

Members

pid PID to be filtered.

input Demux input, as specified by *enum dmx_input*.

output Demux output, as specified by *enum dmx_output*.

pes_type Type of the pes filter, as specified by *enum dmx_pes_type*.

flags Demux PES flags.

struct **dmx_stc**

Stores System Time Counter (STC) information.

Definition

```
struct dmx_stc {
    unsigned int num;
    unsigned int base;
    __u64 stc;
};
```

Members

num input data: number of the STC, from 0 to N.

base output: divisor for STC to get 90 kHz clock.

stc output: stc in **base** * 90 kHz units.

enum **dmx_buffer_flags**

DMX memory-mapped buffer flags

Constants

`DMX_BUFFER_FLAG_HAD_CRC32_DISCARD`

Indicates that the Kernel discarded one or more frames due to wrong CRC32 checksum.

DMX_BUFFER_FLAG_TEI

Indicates that the Kernel has detected a Transport Error indicator (TEI) on a filtered pid.

DMX_BUFFER_PKT_COUNTER_MISMATCH

Indicates that the Kernel has detected a packet counter mismatch on a filtered pid.

DMX_BUFFER_FLAG_DISCONTINUITY_DETECTED

Indicates that the Kernel has detected one or more frame discontinuity.

DMX_BUFFER_FLAG_DISCONTINUITY_INDICATOR

Received at least one packet with a frame discontinuity indicator.

struct **dmx_buffer**
dmx buffer info

Definition

```
struct dmx_buffer {
    __u32 index;
    __u32 bytesused;
    __u32 offset;
    __u32 length;
    __u32 flags;
    __u32 count;
};
```

Members

index id number of the buffer

bytesused number of bytes occupied by data in the buffer (payload);

offset for buffers with memory == DMX_MEMORY_MMAP; offset from the start of the device memory for this plane, (or a "cookie" that should be passed to mmap() as offset)

length size in bytes of the buffer

flags bit array of buffer flags as defined by *enum dmx_buffer_flags*. Filled only at DMX_DQBUF.

count monotonic counter for filled buffers. Helps to identify data stream loses. Filled only at DMX_DQBUF.

Description

Contains data exchanged by application and driver using one of the streaming I/O methods.

Please notice that, for DMX_QBUF, only **index** should be filled. On DMX_DQBUF calls, all fields will be filled by the Kernel.

struct **dmx_requestbuffers**
request dmx buffer information

Definition

```
struct dmx_requestbuffers {
    __u32 count;
    __u32 size;
};
```

Members

count number of requested buffers,

size size in bytes of the requested buffer

Description

Contains data used for requesting a dmx buffer. All reserved fields must be set to zero.

struct **dmx_exportbuffer**
export of dmx buffer as DMABUF file descriptor

Definition

```
struct dmx_exportbuffer {
    __u32 index;
    __u32 flags;
    __s32 fd;
};
```

Members

index id number of the buffer

flags flags for newly created file, currently only O_CLOEXEC is supported, refer to manual of open syscall for more details

fd file descriptor associated with DMABUF (set by driver)

Description

Contains data used for exporting a dmx buffer as DMABUF file descriptor. The buffer is identified by a 'cookie' returned by DMX_QUERYBUF (identical to the cookie used to mmap() the buffer to userspace). All reserved fields must be set to zero. The field reserved0 is expected to become a structure 'type' allowing an alternative layout of the structure content. Therefore this field should not be used for any other extensions.

3.3.3.2 Demux Function Calls

Digital TV demux open()

Name

Digital TV demux open()

Synopsis

```
int open (const char *deviceName, int flags)
```

Arguments

name Name of specific Digital TV demux device.

flags A bit-wise OR of the following flags:

O_RDONLY	read-only access
O_RDWR	read/write access
O_NONBLOCK	open in non-blocking mode (blocking mode is the default)

Description

This system call, used with a device name of `/dev/dvb/adapter?/demux?`, allocates a new filter and returns a handle which can be used for subsequent control of that filter. This call has to be made for each filter to be used, i.e. every returned file descriptor is a reference to a single filter. `/dev/dvb/adapter?/dvr?` is a logical device to be used for retrieving Transport Streams for digital video recording. When reading from this device a transport stream containing the packets from all PES filters set in the corresponding demux device (`/dev/dvb/adapter?/demux?`) having the output set to `DMX_OUT_TS_TAP`. A recorded Transport Stream is replayed by writing to this device.

The significance of blocking or non-blocking mode is described in the documentation for functions where there is a difference. It does not affect the semantics of the `open()` call itself. A device opened in blocking mode can later be put into non-blocking mode (and vice versa) using the `F_SETFL` command of the `fcntl` system call.

Return Value

On success 0 is returned.

On error -1 is returned, and the `errno` variable is set appropriately.

EMFILE	"Too many open files", i.e. no more filters available.
--------	--

The generic error codes are described at the [Generic Error Codes](#) chapter.

Digital TV demux close()

Name

Digital TV demux close()

Synopsis

```
int close(int fd)
```

Arguments

fd File descriptor returned by a previous call to `open()`.

Description

This system call deactivates and deallocates a filter that was previously allocated via the `open()` call.

Return Value

On success 0 is returned.

On error, -1 is returned and the `errno` variable is set appropriately.

The generic error codes are described at the [Generic Error Codes](#) chapter.

Digital TV demux read()

Name

Digital TV demux read()

Synopsis

```
size_t read(int fd, void *buf, size_t count)
```

Arguments

fd

File descriptor returned by a previous call to `open()`.

buf Buffer to be filled

count Max number of bytes to read

Description

This system call returns filtered data, which might be section or Packetized Elementary Stream (PES) data. The filtered data is transferred from the driver's internal circular buffer to `buf`. The maximum amount of data to be transferred is implied by `count`.

Note: if a section filter created with `DMX_CHECK_CRC` flag set, data that fails on CRC check will be silently ignored.

Return Value

On success 0 is returned.

On error -1 is returned, and the `errno` variable is set appropriately.

<code>EWOULDBLOCK</code>	No data to return and <code>O_NONBLOCK</code> was specified.
<code>E_OVERFLOW</code>	The filtered data was not read from the buffer in due time, resulting in non-read data being lost. The buffer is flushed.
<code>ETIMEDOUT</code>	The section was not loaded within the stated timeout period. See <code>ioctl DMX_SET_FILTER</code> for how to set a timeout.
<code>EFAULT</code>	The driver failed to write to the callers buffer due to an invalid <code>*buf</code> pointer.

The generic error codes are described at the [Generic Error Codes](#) chapter.

Digital TV demux write()

Name

Digital TV demux write()

Synopsis

```
ssize_t write(int fd, const void *buf, size_t count)
```

Arguments

fd File descriptor returned by a previous call to `open()`.

buf Buffer with data to be written

count Number of bytes at the buffer

Description

This system call is only provided by the logical device `/dev/dvb/adapter?/dvr?`, associated with the physical demux device that provides the actual DVR functionality. It is used for replay of a digitally recorded Transport Stream. Matching filters have to be defined in the corresponding physical demux device, `/dev/dvb/adapter?/demux?`. The amount of data to be transferred is implied by `count`.

Return Value

On success 0 is returned.

On error -1 is returned, and the `errno` variable is set appropriately.

EWouldBlock	No data was written. This might happen if <code>O_NONBLOCK</code> was specified and there is no more buffer space available (if <code>O_NONBLOCK</code> is not specified the function will block until buffer space is available).
EBUSY	This error code indicates that there are conflicting requests. The corresponding demux device is setup to receive data from the front-end. Make sure that these filters are stopped and that the filters with input set to <code>DMX_IN_DVR</code> are started.

The generic error codes are described at the [Generic Error Codes](#) chapter.

Digital TV mmap()

Name

`dmx-mmap` - Map device memory into application address space

Warning: this API is still experimental

Synopsis

```
#include <unistd.h>
#include <sys/mman.h>
```

```
void *mmap(void *start, size_t length, int prot, int flags, int fd, off_t offset)
```

Arguments

start Map the buffer to this address in the application's address space. When the `MAP_FIXED` flag is specified, `start` must be a multiple of the `pagesize` and `mmap` will fail when the specified address cannot be used. Use of this option is discouraged; applications should just specify a `NULL` pointer here.

length Length of the memory area to map. This must be a multiple of the DVB packet length (188, on most drivers).

prot The `prot` argument describes the desired memory protection. Regardless of the device type and the direction of data exchange it should be set to `PROT_READ | PROT_WRITE`, permitting read and write access to image buffers. Drivers should support at least this combination of flags.

flags The `flags` parameter specifies the type of the mapped object, mapping options and whether modifications made to the mapped copy of the page are private to the process or are to be shared with other references.

`MAP_FIXED` requests that the driver selects no other address than the one specified. If the specified address cannot be used, `mmap()` will fail. If `MAP_FIXED` is specified, `start` must be a multiple of the `pagesize`. Use of this option is discouraged.

One of the `MAP_SHARED` or `MAP_PRIVATE` flags must be set. `MAP_SHARED` allows applications to share the mapped memory with other (e. g. child-) processes.

Note: The Linux Digital TV applications should not set the `MAP_PRIVATE`, `MAP_DENYWRITE`, `MAP_EXECUTABLE` or `MAP_ANON` flags.

fd File descriptor returned by `open()`.

offset Offset of the buffer in device memory, as returned by `ioctl DMX_QUERYBUF` `ioctl`.

Description

The `mmap()` function asks to map `length` bytes starting at `offset` in the memory of the device specified by `fd` into the application address space, preferably at address `start`. This latter address is a hint only, and is usually specified as 0.

Suitable `length` and `offset` parameters are queried with the `ioctl DMX_QUERYBUF` `ioctl`. Buffers must be allocated with the `ioctl DMX_REQBUFS` `ioctl` before they can be queried.

To unmap buffers the `munmap()` function is used.

Return Value

On success `mmap()` returns a pointer to the mapped buffer. On error `MAP_FAILED` (-1) is returned, and the `errno` variable is set appropriately. Possible error codes are:

EBADF `fd` is not a valid file descriptor.

EACCES `fd` is not open for reading and writing.

EINVAL The start or length or offset are not suitable. (E. g. they are too large, or not aligned on a `PAGESIZE` boundary.)

The flags or `prot` value is not supported.

No buffers have been allocated with the *ioctl* `DMX_REQBUFS` *ioctl*.

ENOMEM Not enough physical or virtual memory was available to complete the request.

DVB munmap()

Name

`dmx-munmap` - Unmap device memory

Warning: This API is still experimental.

Synopsis

```
#include <unistd.h>
#include <sys/mman.h>
```

int **munmap**(void **start*, size_t *length*)

Arguments

start Address of the mapped buffer as returned by the `mmap()` function.

length Length of the mapped buffer. This must be the same value as given to `mmap()`.

Description

Unmaps a previously with the `mmap()` function mapped buffer and frees it, if possible.

Return Value

On success `munmap()` returns 0, on failure -1 and the `errno` variable is set appropriately: **EINVAL**. The start or length is incorrect, or no buffers have been mapped yet.

DMX_START

Name

DMX_START

Synopsis

DMX_START

```
int ioctl(int fd, DMX_START)
```

Arguments

fd File descriptor returned by `open()`.

Description

This `ioctl` call is used to start the actual filtering operation defined via the `ioctl` calls `DMX_SET_FILTER` or `DMX_SET_PES_FILTER`.

Return Value

On success 0 is returned.

On error -1 is returned, and the `errno` variable is set appropriately.

EINVAL	Invalid argument, i.e. no filtering parameters provided via the <code>DMX_SET_FILTER</code> or <code>DMX_SET_PES_FILTER</code> <code>ioctl</code> s.
EBUSY	This error code indicates that there are conflicting requests. There are active filters filtering data from another input source. Make sure that these filters are stopped before starting this filter.

The generic error codes are described at the [Generic Error Codes](#) chapter.

DMX_STOP

Name

DMX_STOP

Synopsis

DMX_STOP

```
int ioctl(int fd, DMX_STOP)
```

Arguments

fd File descriptor returned by `open()`.

Description

This `ioctl` call is used to stop the actual filtering operation defined via the `ioctl` calls `DMX_SET_FILTER` or `DMX_SET_PES_FILTER` and started via the `DMX_START` command.

Return Value

On success 0 is returned.

On error -1 is returned, and the `errno` variable is set appropriately.

The generic error codes are described at the *Generic Error Codes* chapter.

DMX_SET_FILTER

Name

DMX_SET_FILTER

Synopsis

DMX_SET_FILTER

```
int ioctl(int fd, DMX_SET_FILTER, struct dmx_sct_filter_params *params)
```

Arguments

fd File descriptor returned by `open()`.

params

Pointer to structure containing filter parameters.

Description

This `ioctl` call sets up a filter according to the filter and mask parameters provided. A timeout may be defined stating number of seconds to wait for a section to be loaded. A value of 0 means that no timeout should be applied. Finally there is a flag field where it is possible to state whether a section should be CRC-checked, whether the filter should be a “one-shot” filter, i.e. if the filtering operation should be stopped after the first section is received, and whether the filtering operation should be started immediately (without waiting for a `DMX_START` `ioctl` call). If a filter was previously set-up, this filter will be canceled, and the receive buffer will be flushed.

Return Value

On success 0 is returned.

On error -1 is returned, and the `errno` variable is set appropriately.

The generic error codes are described at the [Generic Error Codes](#) chapter.

DMX_SET_PES_FILTER

Name

DMX_SET_PES_FILTER

Synopsis

DMX_SET_PES_FILTER

```
int ioctl(int fd, DMX_SET_PES_FILTER, struct dmx_pes_filter_params *params)
```

Arguments

fd File descriptor returned by `open()`.

params Pointer to structure containing filter parameters.

Description

This ioctl call sets up a PES filter according to the parameters provided. By a PES filter is meant a filter that is based just on the packet identifier (PID), i.e. no PES header or payload filtering capability is supported.

Return Value

On success 0 is returned.

On error -1 is returned, and the `errno` variable is set appropriately.

EBUSY	This error code indicates that there are conflicting requests. There are active filters filtering data from another input source. Make sure that these filters are stopped before starting this filter.
-------	---

The generic error codes are described at the [Generic Error Codes](#) chapter.

DMX_SET_BUFFER_SIZE

Name

DMX_SET_BUFFER_SIZE

Synopsis

DMX_SET_BUFFER_SIZE

```
int ioctl(int fd, DMX_SET_BUFFER_SIZE, unsigned long size)
```

Arguments

fd File descriptor returned by `open()`.

size Unsigned long size

Description

This ioctl call is used to set the size of the circular buffer used for filtered data. The default size is two maximum sized sections, i.e. if this function is not called a buffer size of $2 * 4096$ bytes will be used.

Return Value

On success 0 is returned.

On error -1 is returned, and the `errno` variable is set appropriately.

The generic error codes are described at the [Generic Error Codes](#) chapter.

DMX_GET_STC

Name

DMX_GET_STC

Synopsis

DMX_GET_STC

```
int ioctl(int fd, DMX_GET_STC, struct dmx_stc *stc)
```

Arguments

fd File descriptor returned by `open()`.

stc Pointer to `dmx_stc` where the stc data is to be stored.

Description

This `ioctl` call returns the current value of the system time counter (which is driven by a PES filter of type `DMX_PES_PCR`). Some hardware supports more than one STC, so you must specify which one by setting the `num` field of `stc` before the `ioctl` (range 0...n). The result is returned in form of a ratio with a 64 bit numerator and a 32 bit denominator, so the real 90kHz STC value is `stc->stc / stc->base`.

Return Value

On success 0 is returned.

On error -1 is returned, and the `errno` variable is set appropriately.

EINVAL	Invalid stc number.
--------	---------------------

The generic error codes are described at the [Generic Error Codes](#) chapter.

DMX_GET_PES_PIDS

Name

DMX_GET_PES_PIDS

Synopsis

DMX_GET_PES_PIDS

```
int ioctl(fd, DMX_GET_PES_PIDS, __u16 pids[5])
```

Arguments

fd File descriptor returned by `open()`.

pids Array used to store 5 Program IDs.

Description

This `ioctl` allows to query a DVB device to return the first PID used by audio, video, teletext, subtitle and PCR programs on a given service. They're stored as:

PID element	position	content
<code>pids[DMX_PES_AUDIO]</code>	0	first audio PID
<code>pids[DMX_PES_VIDEO]</code>	1	first video PID
<code>pids[DMX_PES_TELETEXT]</code>	2	first teletext PID
<code>pids[DMX_PES_SUBTITLE]</code>	3	first subtitle PID
<code>pids[DMX_PES_PCR]</code>	4	first Program Clock Reference PID

Note: A value equal to `0xffff` means that the PID was not filled by the Kernel.

Return Value

On success 0 is returned.

On error -1 is returned, and the `errno` variable is set appropriately.

The generic error codes are described at the [Generic Error Codes](#) chapter.

DMX_ADD_PID

Name

DMX_ADD_PID

Synopsis

DMX_ADD_PID

```
int ioctl(fd, DMX_ADD_PID, __u16 *pid)
```

Arguments

fd File descriptor returned by `open()`.

pid PID number to be filtered.

Description

This `ioctl` call allows to add multiple PIDs to a transport stream filter previously set up with `DMX_SET_PES_FILTER` and output equal to `DMX_OUT_TSDEMUX_TAP`.

Return Value

On success 0 is returned.

On error -1 is returned, and the `errno` variable is set appropriately.

Generic error codes are described at the *Generic Error Codes* chapter.

DMX_REMOVE_PID

Name

DMX_REMOVE_PID

Synopsis

DMX_REMOVE_PID

```
int ioctl(fd, DMX_REMOVE_PID, __u16 *pid)
```

Arguments

fd File descriptor returned by `open()`.

pid PID of the PES filter to be removed.

Description

This `ioctl` call allows to remove a PID when multiple PIDs are set on a transport stream filter, e. g. a filter previously set up with output equal to `DMX_OUT_TSDEMUX_TAP`, created via either `DMX_SET_PES_FILTER` or `DMX_ADD_PID`.

Return Value

On success 0 is returned.

On error -1 is returned, and the `errno` variable is set appropriately.

The generic error codes are described at the [Generic Error Codes](#) chapter.

`ioctl DMX_REQBUFS`

Name

`DMX_REQBUFS` - Initiate Memory Mapping and/or DMA buffer I/O

Warning: this API is still experimental

Synopsis

`DMX_REQBUFS`

```
int ioctl(int fd, DMX_REQBUFS, struct dmx_requestbuffers *argp)
```

Arguments

fd File descriptor returned by `open()`.

argp Pointer to struct `dmx_requestbuffers`.

Description

This ioctl is used to initiate a memory mapped or DMABUF based demux I/O.

Memory mapped buffers are located in device memory and must be allocated with this ioctl before they can be mapped into the application's address space. User buffers are allocated by applications themselves, and this ioctl is merely used to switch the driver into user pointer I/O mode and to setup some internal structures. Similarly, DMABUF buffers are allocated by applications through a device driver, and this ioctl only configures the driver into DMABUF I/O mode without performing any direct allocation.

To allocate device buffers applications initialize all fields of the struct *dmx_requestbuffers* structure. They set the count field to the desired number of buffers, and size to the size of each buffer.

When the ioctl is called with a pointer to this structure, the driver will attempt to allocate the requested number of buffers and it stores the actual number allocated in the count field. The count can be smaller than the number requested, even zero, when the driver runs out of free memory. A larger number is also possible when the driver requires more buffers to function correctly. The actual allocated buffer size can be returned at size, and can be smaller than what's requested.

When this I/O method is not supported, the ioctl returns an EOPNOTSUPP error code.

Applications can call *ioctl DMX_REQBUFS* again to change the number of buffers, however this cannot succeed when any buffers are still mapped. A count value of zero frees all buffers, after aborting or finishing any DMA in progress.

Return Value

On success 0 is returned, on error -1 and the errno variable is set appropriately. The generic error codes are described at the *Generic Error Codes* chapter.

EOPNOTSUPP The the requested I/O method is not supported.

ioctl DMX_QUERYBUF

Name

DMX_QUERYBUF - Query the status of a buffer

Warning: this API is still experimental

Synopsis

DMX_QUERYBUF

```
int ioctl(int fd, DMX_QUERYBUF, struct dvb_buffer *argp)
```

Arguments

fd File descriptor returned by `open()`.

argp Pointer to struct `dvb_buffer`.

Description

This `ioctl` is part of the `mmap` streaming I/O method. It can be used to query the status of a buffer at any time after buffers have been allocated with the `ioctl DMX_REQBUFS` `ioctl`.

Applications set the `index` field. Valid index numbers range from zero to the number of buffers allocated with `ioctl DMX_REQBUFS` (struct `dvb_requestbuffers` count) minus one.

After calling `ioctl DMX_QUERYBUF` with a pointer to this structure, drivers return an error code or fill the rest of the structure.

On success, the `offset` will contain the offset of the buffer from the start of the device memory, the `length` field its size, and the `bytesused` the number of bytes occupied by data in the buffer (payload).

Return Value

On success 0 is returned, the `offset` will contain the offset of the buffer from the start of the device memory, the `length` field its size, and the `bytesused` the number of bytes occupied by data in the buffer (payload).

On error it returns -1 and the `errno` variable is set appropriately. The generic error codes are described at the [Generic Error Codes](#) chapter.

EINVAL The index is out of bounds.

ioctl DMX_EXPBUF

Name

DMX_EXPBUF - Export a buffer as a DMABUF file descriptor.

Warning: this API is still experimental

Synopsis

DMX_EXPBUF

```
int ioctl(int fd, DMX_EXPBUF, struct dmx_exportbuffer *argp)
```

Arguments

fd File descriptor returned by `open()`.

argp Pointer to struct `dmx_exportbuffer`.

Description

This `ioctl` is an extension to the memory mapping I/O method. It can be used to export a buffer as a DMABUF file at any time after buffers have been allocated with the `ioctl DMX_REQBUFS` `ioctl`.

To export a buffer, applications fill struct `dmx_exportbuffer`. Applications must set the `index` field. Valid index numbers range from zero to the number of buffers allocated with `ioctl DMX_REQBUFS` (struct `dmx_requestbuffers` count) minus one. Additional flags may be posted in the `flags` field. Refer to a manual for `open()` for details. Currently only `O_CLOEXEC`, `O_RDONLY`, `O_WRONLY`, and `O_RDWR` are supported. All other fields must be set to zero. In the case of multi-planar API, every plane is exported separately using multiple `ioctl DMX_EXPBUF` calls.

After calling `ioctl DMX_EXPBUF` the `fd` field will be set by a driver, on success. This is a DMABUF file descriptor. The application may pass it to other DMABUF-aware devices. It is recommended to close a DMABUF file when it is no longer used to allow the associated memory to be reclaimed.

Examples

```
int buffer_export(int v4lfd, enum dmx_buf_type bt, int index, int *dmafd)
{
    struct dmx_exportbuffer expbuf;

    memset(&expbuf, 0, sizeof(expbuf));
    expbuf.type = bt;
    expbuf.index = index;
    if (ioctl(v4lfd, DMX_EXPBUF, &expbuf) == -1) {
        perror("DMX_EXPBUF");
        return -1;
    }

    *dmafd = expbuf.fd;

    return 0;
}
```

Return Value

On success 0 is returned, on error -1 and the `errno` variable is set appropriately. The generic error codes are described at the [Generic Error Codes](#) chapter.

EINVAL A queue is not in MMAP mode or DMABUF exporting is not supported or `flags` or `index` fields are invalid.

ioctl DMX_QBUF, DMX_DQBUF

Name

DMX_QBUF - DMX_DQBUF - Exchange a buffer with the driver

Warning: this API is still experimental

Synopsis

DMX_QBUF

```
int ioctl(int fd, DMX_QBUF, struct dmxf_buffer *argp)
```

DMX_DQBUF

```
int ioctl(int fd, DMX_DQBUF, struct dmxf_buffer *argp)
```

Arguments

fd File descriptor returned by `open()`.

argp Pointer to struct *dmxf_buffer*.

Description

Applications call the `DMX_QBUF` `ioctl` to enqueue an empty (capturing) or filled (output) buffer in the driver's incoming queue. The semantics depend on the selected I/O method.

To enqueue a buffer applications set the `index` field. Valid index numbers range from zero to the number of buffers allocated with `ioctl DMX_REQBUFS` (struct *dmxf_requestbuffers* `count`) minus one. The contents of the struct *dmxf_buffer* returned by a `ioctl DMX_QUERYBUF` `ioctl` will do as well.

When `DMX_QBUF` is called with a pointer to this structure, it locks the memory pages of the buffer in physical memory, so they cannot be swapped out to disk. Buffers remain locked until dequeued, until the device is closed.

Applications call the `DMX_DQBUF` `ioctl` to dequeue a filled (capturing) buffer from the driver's outgoing queue. They just set the `index` field with the buffer ID to be queued. When `DMX_DQBUF` is called with a pointer to struct *dmxf_buffer*, the driver fills the remaining fields or returns an error code.

By default `DMX_DQBUF` blocks when no buffer is in the outgoing queue. When the `O_NONBLOCK` flag was given to the `open()` function, `DMX_DQBUF` returns immediately with an `EAGAIN` error code when no buffer is available.

The struct `dmx_buffer` structure is specified in *Buffers*.

Return Value

On success 0 is returned, on error -1 and the `errno` variable is set appropriately. The generic error codes are described at the *Generic Error Codes* chapter.

EAGAIN Non-blocking I/O has been selected using `O_NONBLOCK` and no buffer was in the outgoing queue.

EINVAL The index is out of bounds, or no buffers have been allocated yet.

EIO `DMX_DQBUF` failed due to an internal error. Can also indicate temporary problems like signal loss or CRC errors.

3.3.4 Digital TV CA Device

The Digital TV CA device controls the conditional access hardware. It can be accessed through `/dev/dvb/adapter?/ca?`. Data types and ioctl definitions can be accessed by including `linux/dvb/ca.h` in your application.

Note: There are three ioctls at this API that aren't documented: `CA_GET_MSG`, `CA_SEND_MSG` and `CA_SET_DESCR`. Documentation for them are welcome.

3.3.4.1 CA Data Types

struct `ca_slot_info`

CA slot interface types and info.

Definition

```
struct ca_slot_info {
    int num;
    int type;
#define CA_CI          1;
#define CA_CI_LINK    2;
#define CA_CI_PHYS    4;
#define CA_DESCR      8;
#define CA_SC         128;
    unsigned int flags;
#define CA_CI_MODULE_PRESENT 1;
#define CA_CI_MODULE_READY  2;
};
```

Members

num slot number.

type slot type.

flags flags applicable to the slot.

Description

This struct stores the CA slot information.

type can be:

- CA_CI - CI high level interface;
- CA_CI_LINK - CI link layer level interface;
- CA_CI_PHYS - CI physical layer level interface;
- CA_DESCR - built-in descrambler;
- CA_SC -simple smart card interface.

flags can be:

- CA_CI_MODULE_PRESENT - module (or card) inserted;
- CA_CI_MODULE_READY - module is ready for usage.

struct **ca_descr_info**
descrambler types and info.

Definition

```
struct ca_descr_info {
    unsigned int num;
    unsigned int type;
#define CA_ECD          1;
#define CA_NDS          2;
#define CA_DSS          4;
};
```

Members

num number of available descramblers (keys).

type type of supported scrambling system.

Description

Identifies the number of descramblers and their type.

type can be:

- CA_ECD - European Common Descrambler (ECD) hardware;
- CA_NDS - Videoguard (NDS) hardware;
- CA_DSS - Distributed Sample Scrambling (DSS) hardware.

struct **ca_caps**
CA slot interface capabilities.

Definition

```
struct ca_caps {
    unsigned int slot_num;
    unsigned int slot_type;
    unsigned int descr_num;
```



```
unsigned int descr_type;
};
```

Members

slot_num total number of CA card and module slots.

slot_type bitmap with all supported types as defined at struct `ca_slot_info` (e. g. `CA_CI`, `CA_CI_LINK`, etc).

descr_num total number of descrambler slots (keys)

descr_type bitmap with all supported types as defined at struct `ca_descr_info` (e. g. `CA_ECD`, `CA_NDS`, etc).

struct **ca_msg**
a message to/from a CI-CAM

Definition

```
struct ca_msg {
    unsigned int index;
    unsigned int type;
    unsigned int length;
    unsigned char msg[256];
};
```

Members

index unused

type unused

length length of the message

msg message

Description

This struct carries a message to be send/received from a CI CA module.

struct **ca_descr**
CA descrambler control words info

Definition

```
struct ca_descr {
    unsigned int index;
    unsigned int parity;
    unsigned char cw[8];
};
```

Members

index CA Descrambler slot

parity control words parity, where 0 means even and 1 means odd

cw CA Descrambler control words

3.3.4.2 CA Function Calls

Digital TV CA open()

Name

Digital TV CA open()

Synopsis

```
int open(const char *name, int flags)
```

Arguments

name Name of specific Digital TV CA device.

flags A bit-wise OR of the following flags:

<code>O_RDONLY</code>	read-only access
<code>O_RDWR</code>	read/write access
<code>O_NONBLOCK</code>	open in non-blocking mode (blocking mode is the default)

Description

This system call opens a named ca device (e.g. `/dev/dvb/adapter?/ca?`) for subsequent use.

When an `open()` call has succeeded, the device will be ready for use. The significance of blocking or non-blocking mode is described in the documentation for functions where there is a difference. It does not affect the semantics of the `open()` call itself. A device opened in blocking mode can later be put into non-blocking mode (and vice versa) using the `F_SETFL` command of the `fcntl` system call. This is a standard system call, documented in the Linux manual page for `fcntl`. Only one user can open the CA Device in `O_RDWR` mode. All other attempts to open the device in this mode will fail, and an error code will be returned.

Return Value

On success 0 is returned.

On error -1 is returned, and the `errno` variable is set appropriately.

Generic error codes are described at the [Generic Error Codes](#) chapter.

Digital TV CA close()

Name

Digital TV CA close()

Synopsis

```
int close(int fd)
```

Arguments

fd File descriptor returned by a previous call to `open()`.

Description

This system call closes a previously opened CA device.

Return Value

On success 0 is returned.

On error -1 is returned, and the `errno` variable is set appropriately.

Generic error codes are described at the [Generic Error Codes](#) chapter.

CA_RESET

Name

CA_RESET

Synopsis

CA_RESET

```
int ioctl(fd, CA_RESET)
```

Arguments

fd File descriptor returned by a previous call to `open()`.

Description

Puts the Conditional Access hardware on its initial state. It should be called before start using the CA hardware.

Return Value

On success 0 is returned.

On error -1 is returned, and the `errno` variable is set appropriately.

Generic error codes are described at the *Generic Error Codes* chapter.

CA_GET_CAP

Name

CA_GET_CAP

Synopsis

CA_GET_CAP

```
int ioctl(fd, CA_GET_CAP, struct ca_caps *caps)
```

Arguments

fd File descriptor returned by a previous call to `open()`.

caps Pointer to struct `ca_caps`.

Description

Queries the Kernel for information about the available CA and descrambler slots, and their types.

Return Value

On success 0 is returned and `ca_caps` is filled.

On error, -1 is returned and the `errno` variable is set appropriately.

The generic error codes are described at the [Generic Error Codes](#) chapter.

CA_GET_SLOT_INFO

Name

CA_GET_SLOT_INFO

Synopsis

CA_GET_SLOT_INFO

```
int ioctl(fd, CA_GET_SLOT_INFO, struct ca_slot_info *info)
```

Arguments

fd File descriptor returned by a previous call to `open()`.

info Pointer to struct `ca_slot_info`.

Description

Returns information about a CA slot identified by `ca_slot_info.slot_num`.

Return Value

On success 0 is returned, and `ca_slot_info` is filled.

On error -1 is returned, and the `errno` variable is set appropriately.

ENODEV	the slot is not available.
--------	----------------------------

The generic error codes are described at the [Generic Error Codes](#) chapter.

CA_GET_DESCR_INFO

Name

CA_GET_DESCR_INFO

Synopsis

CA_GET_DESCR_INFO

```
int ioctl(fd, CA_GET_DESCR_INFO, struct ca_descr_info *desc)
```

Arguments

fd File descriptor returned by a previous call to `open()`.

desc Pointer to `struct ca_descr_info`.

Description

Returns information about all descrambler slots.

Return Value

On success 0 is returned, and `ca_descr_info` is filled.

On error -1 is returned, and the `errno` variable is set appropriately. The generic error codes are described at the *Generic Error Codes* chapter.

CA_GET_MSG

Name

CA_GET_MSG

Synopsis

CA_GET_MSG

```
int ioctl(fd, CA_GET_MSG, struct ca_msg *msg)
```

Arguments

fd File descriptor returned by a previous call to `open()`.

msg Pointer to struct `ca_msg`.

Description

Receives a message via a CI CA module.

Note: Please notice that, on most drivers, this is done by reading from the `/dev/adapter?/ca?` device node.

Return Value

On success 0 is returned.

On error -1 is returned, and the `errno` variable is set appropriately.

Generic error codes are described at the [Generic Error Codes](#) chapter.

CA_SEND_MSG

Name

CA_SEND_MSG

Synopsis

CA_SEND_MSG

```
int ioctl(fd, CA_SEND_MSG, struct ca_msg *msg)
```

Arguments

fd File descriptor returned by a previous call to `open()`.

msg Pointer to struct `ca_msg`.

Description

Sends a message via a CI CA module.

Note: Please notice that, on most drivers, this is done by writing to the `/dev/adapter?/ca?` device node.

Return Value

On success 0 is returned.

On error -1 is returned, and the `errno` variable is set appropriately.

Generic error codes are described at the [Generic Error Codes](#) chapter.

CA_SET_DESCR

Name

CA_SET_DESCR

Synopsis

CA_SET_DESCR

```
int ioctl(fd, CA_SET_DESCR, struct ca_descr *desc)
```

Arguments

fd File descriptor returned by a previous call to `open()`.

msg Pointer to struct `ca_descr`.

Description

CA_SET_DESCR is used for feeding descrambler CA slots with descrambling keys (referred as control words).

Return Value

On success 0 is returned.

On error -1 is returned, and the `errno` variable is set appropriately.

Generic error codes are described at the [Generic Error Codes](#) chapter.

3.3.4.3 The High level CI API

Note: This documentation is outdated.

This document describes the high level CI API as in accordance to the Linux DVB API.

With the High Level CI approach any new card with almost any random architecture can be implemented with this style, the definitions inside the switch statement can be easily adapted for any card, thereby eliminating the need for any additional ioctls.

The disadvantage is that the driver/hardware has to manage the rest. For the application programmer it would be as simple as sending/receiving an array to/from the CI ioctls as defined in the Linux DVB API. No changes have been made in the API to accommodate this feature.

Why the need for another CI interface?

This is one of the most commonly asked question. Well a nice question. Strictly speaking this is not a new interface.

The CI interface is defined in the DVB API in `ca.h` as:

```
typedef struct ca_slot_info {
    int num;                /* slot number */

    int type;               /* CA interface this slot supports */
#define CA_CI              1 /* CI high level interface */
#define CA_CI_LINK        2 /* CI link layer level interface */
#define CA_CI_PHYS        4 /* CI physical layer level interface */
#define CA_DESCR          8 /* built-in descrambler */
#define CA_SC             128 /* simple smart card interface */

    unsigned int flags;
#define CA_CI_MODULE_PRESENT 1 /* module (or card) inserted */
#define CA_CI_MODULE_READY  2
} ca_slot_info_t;
```

This CI interface follows the CI high level interface, which is not implemented by most applications. Hence this area is revisited.

This CI interface is quite different in the case that it tries to accommodate all other CI based devices, that fall into the other categories.

This means that this CI interface handles the EN50221 style tags in the Application layer only and no session management is taken care of by the application. The driver/hardware will take care of all that.

This interface is purely an EN50221 interface exchanging APDU's. This means that no session management, link layer or a transport layer do exist in this case in the application to driver communication. It is as simple as that. The driver/hardware has to take care of that.

With this High Level CI interface, the interface can be defined with the regular ioctls.

All these ioctls are also valid for the High level CI interface

```
#define CA_RESET_IO('o', 128) #define CA_GET_CAP_IOR('o', 129, ca_caps_t) #define CA_GET_SLOT_INFO_IOR('o', 130, ca_slot_info_t) #define CA_GET_DESCR_INFO_IOR('o', 131, ca_descr_info_t) #define CA_GET_MSG_IOR('o', 132, ca_msg_t) #define CA_SEND_MSG_IOW('o', 133, ca_msg_t) #define CA_SET_DESCR_IOW('o', 134, ca_descr_t)
```

On querying the device, the device yields information thus:

```
CA_GET_SLOT_INFO
-----
Command = [info]
APP: Number=[1]
APP: Type=[1]
APP: flags=[1]
APP: CI High level interface
APP: CA/CI Module Present

CA_GET_CAP
-----
Command = [caps]
APP: Slots=[1]
APP: Type=[1]
APP: Descrambler keys=[16]
APP: Type=[1]

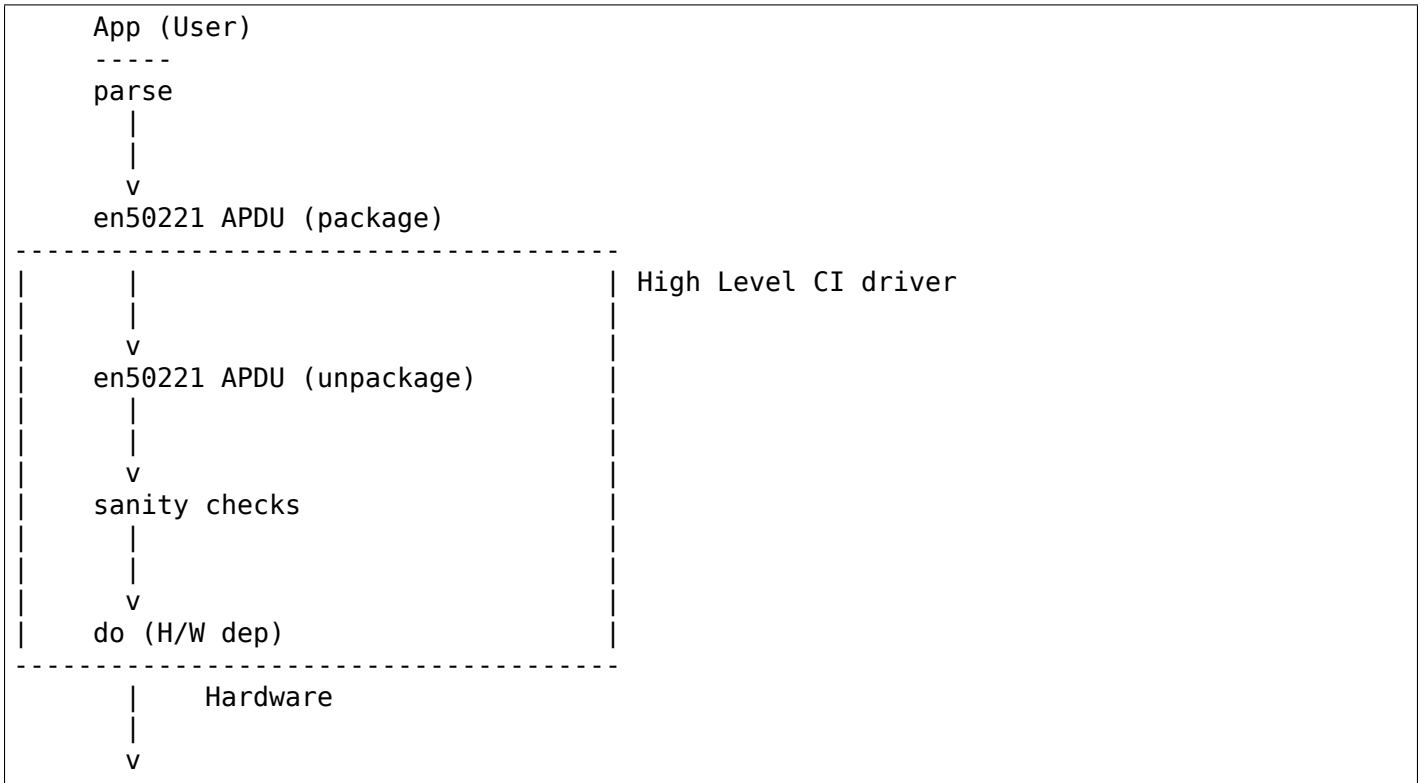
CA_SEND_MSG
-----
Descriptors(Program Level)=[ 09 06 06 04 05 50 ff f1]
Found CA descriptor @ program level

(20) ES type=[2] ES pid=[201] ES length =[0 (0x0)]
(25) ES type=[4] ES pid=[301] ES length =[0 (0x0)]
ca_message length is 25 (0x19) bytes
EN50221 CA MSG=[ 9f 80 32 19 03 01 2d d1 f0 08 01 09 06 06 04 05 50 ff f1 02 e0 c9 00_
↪00 04 e1 2d 00 00]
```

Not all ioctl's are implemented in the driver from the API, the other features of the hardware that cannot be implemented by the API are achieved using the CA_GET_MSG and CA_SEND_MSG ioctls. An EN50221 style wrapper is used to exchange the data to maintain compatibility with other hardware.

```
/* a message to/from a CI-CAM */
typedef struct ca_msg {
    unsigned int index;
    unsigned int type;
    unsigned int length;
    unsigned char msg[256];
} ca_msg_t;
```

The flow of data can be described thus,



The High Level CI interface uses the EN50221 DVB standard, following a standard ensures futureproofness.

3.3.5 Digital TV Network API

The Digital TV net device controls the mapping of data packages that are part of a transport stream to be mapped into a virtual network interface, visible through the standard Linux network protocol stack.

Currently, two encapsulations are supported:

- [Multi Protocol Encapsulation \(MPE\)](#)
- [Ultra Lightweight Encapsulation \(ULE\)](#)

In order to create the Linux virtual network interfaces, an application needs to tell to the Kernel what are the PIDs and the encapsulation types that are present on the transport stream. This is done through `/dev/dvb/adapter?/net?` device node. The data will be available via virtual `dvb?_?` network interfaces, and will be controlled/routed via the standard ip tools (like ip, route, netstat, ifconfig, etc).

Data types and ioctl definitions are defined via `linux/dvb/net.h` header.

3.3.5.1 Digital TV net Function Calls

Net Data Types

struct **dvb_net_if**
describes a DVB network interface

Definition

```
struct dvb_net_if {
    __u16 pid;
    __u16 if_num;
    __u8 feetype;
#define DVB_NET_FEEDTYPE_MPE 0 ;
#define DVB_NET_FEEDTYPE_ULE 1 ;
};
```

Members

pid Packet ID (PID) of the MPEG-TS that contains data

if_num number of the Digital TV interface.

feedtype Encapsulation type of the feed.

Description

A MPEG-TS stream may contain packet IDs with IP packages on it. This struct describes it, and the type of encoding.

feedtype can be:

- DVB_NET_FEEDTYPE_MPE for MPE encoding
- DVB_NET_FEEDTYPE_ULE for ULE encoding.

ioctl NET_ADD_IF

Name

NET_ADD_IF - Creates a new network interface for a given Packet ID.

Synopsis

NET_ADD_IF

```
int ioctl(int fd, NET_ADD_IF, struct dvb_net_if *net_if)
```

Arguments

fd File descriptor returned by `open()`.

net_if pointer to struct `dvb_net_if`

Description

The `NET_ADD_IF` ioctl system call selects the Packet ID (PID) that contains a TCP/IP traffic, the type of encapsulation to be used (MPE or ULE) and the interface number for the new interface to be created. When the system call successfully returns, a new virtual network interface is created.

The struct `dvb_net_if::ifnum` field will be filled with the number of the created interface.

Return Value

On success 0 is returned, and `ca_slot_info` is filled.

On error -1 is returned, and the `errno` variable is set appropriately.

The generic error codes are described at the [Generic Error Codes](#) chapter.

ioctl NET_REMOVE_IF

Name

`NET_REMOVE_IF` - Removes a network interface.

Synopsis

NET_REMOVE_IF

```
int ioctl(int fd, NET_REMOVE_IF, int ifnum)
```

Arguments

fd File descriptor returned by `open()`.

net_if number of the interface to be removed

Description

The `NET_REMOVE_IF` ioctl deletes an interface previously created via [NET_ADD_IF](#).

Return Value

On success 0 is returned, and `ca_slot_info` is filled.

On error -1 is returned, and the `errno` variable is set appropriately.

The generic error codes are described at the [Generic Error Codes](#) chapter.

ioctl NET_GET_IF

Name

`NET_GET_IF` - Read the configuration data of an interface created via - [NET_ADD_IF](#).

Synopsis

NET_GET_IF

```
int ioctl(int fd, NET_GET_IF, struct dvb_net_if *net_if)
```

Arguments

fd File descriptor returned by `open()`.

net_if pointer to struct `dvb_net_if`

Description

The `NET_GET_IF` ioctl uses the interface number given by the struct `dvb_net_if::ifnum` field and fills the content of struct `dvb_net_if` with the packet ID and encapsulation type used on such interface. If the interface was not created yet with [NET_ADD_IF](#), it will return -1 and fill the `errno` with `EINVAL` error code.

Return Value

On success 0 is returned, and `ca_slot_info` is filled.

On error -1 is returned, and the `errno` variable is set appropriately.

The generic error codes are described at the [Generic Error Codes](#) chapter.

3.3.6 Digital TV Deprecated APIs

The APIs described here **should not** be used on new drivers or applications.

The DVBv3 frontend API has issues with new delivery systems, including DVB-S2, DVB-T2, ISDB, etc.

Attention: The APIs described here doesn't necessarily reflect the current code implementation, as this section of the document was written for DVB version 1, while the code reflects DVB version 3 implementation.

3.3.6.1 Digital TV Frontend legacy API (a. k. a. DVBv3)

The usage of this API is deprecated, as it doesn't support all digital TV standards, doesn't provide good statistics measurements and provides incomplete information. This is kept only to support legacy applications.

Frontend Legacy Data Types

Frontend type

For historical reasons, frontend types are named by the type of modulation used in transmission. The frontend types are given by `fe_type_t` type, defined as:

`fe_type`

Table 250: Frontend types

<code>fe_type</code>	Description	<i>DTV_DELIVERY_SYSTEM</i> equivalent type
<code>FE_QPSK</code>	For DVB-S standard	<code>SYS_DVBS</code>
<code>FE_QAM</code>	For DVB-C annex A standard	<code>SYS_DVBC_ANNEX_A</code>
<code>FE_OFDM</code>	For DVB-T standard	<code>SYS_DVBT</code>
<code>FE_ATSC</code>	For ATSC standard (terrestrial) or for DVB-C Annex B (cable) used in US.	<code>SYS_ATSC</code> (terrestrial) or <code>SYS_DVBC_ANNEX_B</code> (cable)

Newer formats like DVB-S2, ISDB-T, ISDB-S and DVB-T2 are not described at the above, as they're supported via the new *FE_GET_PROPERTY/FE_GET_SET_PROPERTY* ioctl's, using the *DTV_DELIVERY_SYSTEM* parameter.

In the old days, struct `dvb_frontend_info` used to contain `fe_type_t` field to indicate the delivery systems, filled with either `FE_QPSK`, `FE_QAM`, `FE_OFDM` or `FE_ATSC`. While this is still filled to keep backward compatibility, the usage of this field is deprecated, as it can report just one delivery system, but some devices support multiple delivery systems. Please use `DTV_ENUM_DELSYS` instead.

On devices that support multiple delivery systems, struct `dvb_frontend_info::fe_type_t` is filled with the currently standard, as selected by the last call to `FE_SET_PROPERTY` using the `DTV_DELIVERY_SYSTEM` property.

Frontend bandwidth

`fe_bandwidth`

Table 251: enum `fe_bandwidth`

ID	Description
<code>BANDWIDTH_AUTO</code>	Autodetect bandwidth (if supported)
<code>BANDWIDTH_1_712_MHZ</code>	1.712 MHz
<code>BANDWIDTH_5_MHZ</code>	5 MHz
<code>BANDWIDTH_6_MHZ</code>	6 MHz
<code>BANDWIDTH_7_MHZ</code>	7 MHz
<code>BANDWIDTH_8_MHZ</code>	8 MHz
<code>BANDWIDTH_10_MHZ</code>	10 MHz

`dvb_frontend_parameters`

frontend parameters

The kind of parameters passed to the frontend device for tuning depend on the kind of hardware you are using.

The struct `dvb_frontend_parameters` uses a union with specific per-system parameters. However, as newer delivery systems required more data, the structure size weren't enough to fit, and just extending its size would break the existing applications. So, those parameters were replaced by the usage of `FE_GET_PROPERTY/FE_SET_PROPERTY` ioctl's. The new API is flexible enough to add new parameters to existing delivery systems, and to add newer delivery systems.

So, newer applications should use `FE_GET_PROPERTY/FE_SET_PROPERTY` instead, in order to be able to support the newer System Delivery like DVB-S2, DVB-T2, DVB-C2, ISDB, etc.

All kinds of parameters are combined as a union in the `dvb_frontend_parameters` structure:

```
struct dvb_frontend_parameters {
    uint32_t frequency;      /* (absolute) frequency in Hz for QAM/OFDM */
                          /* intermediate frequency in kHz for QPSK */
    fe_spectral_inversion_t inversion;
    union {
```



```

    struct dvb_qpsk_parameters qpsk;
    struct dvb_qam_parameters qam;
    struct dvb_ofdm_parameters ofdm;
    struct dvb_vsb_parameters vsb;
} u;
};

```

In the case of QPSK frontends the frequency field specifies the intermediate frequency, i.e. the offset which is effectively added to the local oscillator frequency (LOF) of the LNB. The intermediate frequency has to be specified in units of kHz. For QAM and OFDM frontends the frequency specifies the absolute frequency and is given in Hz.

dvb_qpsk_parameters

QPSK parameters

For satellite QPSK frontends you have to use the `dvb_qpsk_parameters` structure:

```

struct dvb_qpsk_parameters {
    uint32_t      symbol_rate; /* symbol rate in Symbols per second */
    fe_code_rate_t fec_inner;  /* forward error correction (see above) */
};

```

dvb_qam_parameters

QAM parameters

for cable QAM frontend you use the `dvb_qam_parameters` structure:

```

struct dvb_qam_parameters {
    uint32_t      symbol_rate; /* symbol rate in Symbols per second */
    fe_code_rate_t fec_inner;  /* forward error correction (see above) */
    fe_modulation_t modulation; /* modulation type (see above) */
};

```

dvb_vsb_parameters

VSB parameters

ATSC frontends are supported by the `dvb_vsb_parameters` structure:

```

struct dvb_vsb_parameters {
    fe_modulation_t modulation; /* modulation type (see above) */
};

```

dvb_ofdm_parameters

OFDM parameters

DVB-T frontends are supported by the `dvb_ofdm_parameters` structure:

```
struct dvb_ofdm_parameters {
    fe_bandwidth_t      bandwidth;
    fe_code_rate_t     code_rate_HP; /* high priority stream code rate */
    fe_code_rate_t     code_rate_LP; /* low priority stream code rate */
    fe_modulation_t    constellation; /* modulation type (see above) */
    fe_transmit_mode_t transmission_mode;
    fe_guard_interval_t guard_interval;
    fe_hierarchy_t     hierarchy_information;
};
```

dvb_frontend_event

frontend events

```
struct dvb_frontend_event {
    fe_status_t status;
    struct dvb_frontend_parameters parameters;
};
```

Frontend Legacy Function Calls

Those functions are defined at DVB version 3. The support is kept in the kernel due to compatibility issues only. Their usage is strongly not recommended

FE_READ_BER

Name

FE_READ_BER

Attention: This ioctl is deprecated.

Synopsis

FE_READ_BER

```
int ioctl(int fd, FE_READ_BER, uint32_t *ber)
```

Arguments

fd File descriptor returned by `open()`.

ber The bit error rate is stored into `*ber`.

Description

This `ioctl` call returns the bit error rate for the signal currently received/demodulated by the front-end. For this command, read-only access to the device is sufficient.

Return Value

On success 0 is returned.

On error -1 is returned, and the `errno` variable is set appropriately.

Generic error codes are described at the [Generic Error Codes](#) chapter.

FE_READ_SNR

Name

FE_READ_SNR

Attention: This `ioctl` is deprecated.

Synopsis

FE_READ_SNR

```
int ioctl(int fd, FE_READ_SNR, int16_t *snr)
```

Arguments

fd File descriptor returned by `open()`.

snr The signal-to-noise ratio is stored into `*snr`.

Description

This ioctl call returns the signal-to-noise ratio for the signal currently received by the front-end. For this command, read-only access to the device is sufficient.

Return Value

On success 0 is returned.

On error -1 is returned, and the `errno` variable is set appropriately.

Generic error codes are described at the *Generic Error Codes* chapter.

FE_READ_SIGNAL_STRENGTH

Name

FE_READ_SIGNAL_STRENGTH

Attention: This ioctl is deprecated.

Synopsis

FE_READ_SIGNAL_STRENGTH

```
int ioctl(int fd, FE_READ_SIGNAL_STRENGTH, uint16_t *strength)
```

Arguments

fd File descriptor returned by `open()`.

strength The signal strength value is stored into `*strength`.

Description

This ioctl call returns the signal strength value for the signal currently received by the front-end. For this command, read-only access to the device is sufficient.

Return Value

On success 0 is returned.

On error -1 is returned, and the `errno` variable is set appropriately.

Generic error codes are described at the *Generic Error Codes* chapter.

FE_READ_UNCORRECTED_BLOCKS

Name

FE_READ_UNCORRECTED_BLOCKS

Attention: This ioctl is deprecated.

Synopsis

FE_READ_UNCORRECTED_BLOCKS

```
int ioctl(int fd, FE_READ_UNCORRECTED_BLOCKS, uint32_t *ublocks)
```

Arguments

fd File descriptor returned by `open()`.

ublocks The total number of uncorrected blocks seen by the driver so far.

Description

This ioctl call returns the number of uncorrected blocks detected by the device driver during its lifetime. For meaningful measurements, the increment in block count during a specific time interval should be calculated. For this command, read-only access to the device is sufficient.

Return Value

On success 0 is returned.

On error -1 is returned, and the `errno` variable is set appropriately.

Generic error codes are described at the *Generic Error Codes* chapter.

FE_SET_FRONTEND

Attention: This ioctl is deprecated.

Name

FE_SET_FRONTEND

Synopsis

FE_SET_FRONTEND

```
int ioctl(int fd, FE_SET_FRONTEND, struct dvb_frontend_parameters *p)
```

Arguments

fd File descriptor returned by `open()`.

p Points to parameters for tuning operation.

Description

This ioctl call starts a tuning operation using specified parameters. The result of this call will be successful if the parameters were valid and the tuning could be initiated. The result of the tuning operation in itself, however, will arrive asynchronously as an event (see documentation for [FE_GET_EVENT](#) and `FrontendEvent`.) If a new [FE_SET_FRONTEND](#) operation is initiated before the previous one was completed, the previous operation will be aborted in favor of the new one. This command requires read/write access to the device.

Return Value

On success 0 is returned.

On error -1 is returned, and the `errno` variable is set appropriately.

EINVAL	Maximum supported symbol rate reached.
--------	--

Generic error codes are described at the [Generic Error Codes](#) chapter.

FE_GET_FRONTEND

Name

FE_GET_FRONTEND

Attention: This ioctl is deprecated.

Synopsis

FE_GET_FRONTEND

```
int ioctl(int fd, FE_GET_FRONTEND, struct dvb_frontend_parameters *p)
```

Arguments

fd File descriptor returned by `open()`.

p Points to parameters for tuning operation.

Description

This ioctl call queries the currently effective frontend parameters. For this command, read-only access to the device is sufficient.

Return Value

On success 0 is returned.

On error -1 is returned, and the `errno` variable is set appropriately.

EINVAL	Maximum supported symbol rate reached.
--------	--

Generic error codes are described at the [Generic Error Codes](#) chapter.

FE_GET_EVENT

Name

FE_GET_EVENT

Attention: This ioctl is deprecated.

Synopsis

FE_GET_EVENT

```
int ioctl(int fd, FE_GET_EVENT, struct dvb_frontend_event *ev)
```

Arguments

fd File descriptor returned by `open()`.

ev Points to the location where the event, if any, is to be stored.

Description

This `ioctl` call returns a frontend event if available. If an event is not available, the behavior depends on whether the device is in blocking or non-blocking mode. In the latter case, the call fails immediately with `errno` set to `EWOULDBLOCK`. In the former case, the call blocks until an event becomes available.

Return Value

On success 0 is returned.

On error -1 is returned, and the `errno` variable is set appropriately.

<code>EWOULDBLOCK</code>	There is no event pending, and the device is in non-blocking mode.
<code>E_OVERFLOW</code>	Overflow in event queue - one or more events were lost.

Generic error codes are described at the [Generic Error Codes](#) chapter.

FE_DISHNETWORK_SEND_LEGACY_CMD

Name

`FE_DISHNETWORK_SEND_LEGACY_CMD`

Synopsis

FE_DISHNETWORK_SEND_LEGACY_CMD

```
int ioctl(int fd, FE_DISHNETWORK_SEND_LEGACY_CMD, unsigned long cmd)
```


Arguments

fd File descriptor returned by `open()`.

cmd Sends the specified raw cmd to the dish via DISEqC.

Description

Warning: This is a very obscure legacy command, used only at stv0299 driver. Should not be used on newer drivers.

It provides a non-standard method for selecting Diseqc voltage on the frontend, for Dish Network legacy switches.

As support for this ioctl were added in 2004, this means that such dishes were already legacy in 2004.

Return Value

On success 0 is returned.

On error -1 is returned, and the `errno` variable is set appropriately.

Generic error codes are described at the *Generic Error Codes* chapter.

3.3.7 Examples

In the past, we used to have a set of examples here. However, those examples got out of date and doesn't even compile nowadays.

Also, nowadays, the best is to use the `libdvbv5` DVB API nowadays, with is fully documented.

Please refer to the `libdvbv5` for updated/recommended examples.

3.3.8 Digital TV uAPI header files

3.3.8.1 Digital TV uAPI headers

`frontend.h`

```

/* SPDX-License-Identifier: LGPL-2.1+ WITH Linux-syscall-note */
/*
 * frontend.h
 *
 * Copyright (C) 2000 Marcus Metzler <marcus@convergence.de>
 *           Ralph Metzler <ralph@convergence.de>
 *           Holger Waechtler <holger@convergence.de>
 *           Andre Draszik <ad@convergence.de>
 *           for convergence integrated media GmbH

```

```
*
* This program is free software; you can redistribute it and/or
* modify it under the terms of the GNU Lesser General Public License
* as published by the Free Software Foundation; either version 2.1
* of the License, or (at your option) any later version.
*
* This program is distributed in the hope that it will be useful,
* but WITHOUT ANY WARRANTY; without even the implied warranty of
* MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
* GNU General Public License for more details.
*
* You should have received a copy of the GNU Lesser General Public License
* along with this program; if not, write to the Free Software
* Foundation, Inc., 59 Temple Place - Suite 330, Boston, MA 02111-1307, USA.
*
*/

#ifndef _DVBFRONTEND_H_
#define _DVBFRONTEND_H_

#include <linux/types.h>

/**
 * enum fe_caps - Frontend capabilities
 *
 * @FE_IS_STUPID:           There's something wrong at the
 *                          frontend, and it can't report its
 *                          capabilities.
 * @FE_CAN_INVERSION_AUTO: Can auto-detect frequency spectral
 *                          band inversion
 * @FE_CAN_FEC_1_2:        Supports FEC 1/2
 * @FE_CAN_FEC_2_3:        Supports FEC 2/3
 * @FE_CAN_FEC_3_4:        Supports FEC 3/4
 * @FE_CAN_FEC_4_5:        Supports FEC 4/5
 * @FE_CAN_FEC_5_6:        Supports FEC 5/6
 * @FE_CAN_FEC_6_7:        Supports FEC 6/7
 * @FE_CAN_FEC_7_8:        Supports FEC 7/8
 * @FE_CAN_FEC_8_9:        Supports FEC 8/9
 * @FE_CAN_FEC_AUTO:       Can auto-detect FEC
 * @FE_CAN_QPSK:           Supports QPSK modulation
 * @FE_CAN_QAM_16:         Supports 16-QAM modulation
 * @FE_CAN_QAM_32:         Supports 32-QAM modulation
 * @FE_CAN_QAM_64:         Supports 64-QAM modulation
 * @FE_CAN_QAM_128:        Supports 128-QAM modulation
 * @FE_CAN_QAM_256:        Supports 256-QAM modulation
 * @FE_CAN_QAM_AUTO:       Can auto-detect QAM modulation
 * @FE_CAN_TRANSMISSION_MODE_AUTO: Can auto-detect transmission mode
 * @FE_CAN_BANDWIDTH_AUTO: Can auto-detect bandwidth
 * @FE_CAN_GUARD_INTERVAL_AUTO: Can auto-detect guard interval
 * @FE_CAN_HIERARCHY_AUTO: Can auto-detect hierarchy
 * @FE_CAN_8VSB:           Supports 8-VSB modulation

```

```

* @FE_CAN_16VSB:          Supporta 16-VSB modulation
* @FE_HAS_EXTENDED_CAPS:  Unused
* @FE_CAN_MULTISTREAM:    Supports multistream filtering
* @FE_CAN_TURBO_FEC:      Supports "turbo FEC" modulation
* @FE_CAN_2G_MODULATION:  Supports "2nd generation" modulation,
*                          e. g. DVB-S2, DVB-T2, DVB-C2
* @FE_NEEDS_BENDING:     Unused
* @FE_CAN_RECOVER:        Can recover from a cable unplug
*                          automatically
* @FE_CAN_MUTE_TS:        Can stop spurious TS data output
*/
enum fe_caps {
    FE_IS_STUPID                = 0,
    FE_CAN_INVERSION_AUTO      = 0x1,
    FE_CAN_FEC_1_2             = 0x2,
    FE_CAN_FEC_2_3             = 0x4,
    FE_CAN_FEC_3_4             = 0x8,
    FE_CAN_FEC_4_5             = 0x10,
    FE_CAN_FEC_5_6             = 0x20,
    FE_CAN_FEC_6_7             = 0x40,
    FE_CAN_FEC_7_8             = 0x80,
    FE_CAN_FEC_8_9             = 0x100,
    FE_CAN_FEC_AUTO            = 0x200,
    FE_CAN_QPSK                 = 0x400,
    FE_CAN_QAM_16               = 0x800,
    FE_CAN_QAM_32               = 0x1000,
    FE_CAN_QAM_64               = 0x2000,
    FE_CAN_QAM_128              = 0x4000,
    FE_CAN_QAM_256              = 0x8000,
    FE_CAN_QAM_AUTO             = 0x10000,
    FE_CAN_TRANSMISSION_MODE_AUTO = 0x20000,
    FE_CAN_BANDWIDTH_AUTO       = 0x40000,
    FE_CAN_GUARD_INTERVAL_AUTO  = 0x80000,
    FE_CAN_HIERARCHY_AUTO       = 0x100000,
    FE_CAN_8VSB                 = 0x200000,
    FE_CAN_16VSB                = 0x400000,
    FE_HAS_EXTENDED_CAPS        = 0x800000,
    FE_CAN_MULTISTREAM           = 0x4000000,
    FE_CAN_TURBO_FEC            = 0x8000000,
    FE_CAN_2G_MODULATION        = 0x10000000,
    FE_NEEDS_BENDING            = 0x20000000,
    FE_CAN_RECOVER               = 0x40000000,
    FE_CAN_MUTE_TS              = 0x80000000
};

/*
* DEPRECATED: Should be kept just due to backward compatibility.
*/
enum fe_type {
    FE_QPSK,
    FE_QAM,

```

```
    FE_OFDM,  
    FE_ATSC  
};  
  
/**  
 * struct dvb_frontend_info - Frontend properties and capabilities  
 *  
 * @name:           Name of the frontend  
 * @type:           ****DEPRECATED****.  
 *                 Should not be used on modern programs,  
 *                 as a frontend may have more than one type.  
 *                 In order to get the support types of a given  
 *                 frontend, use :c:type:`DTV_ENUM_DELSYS`  
 *                 instead.  
 * @frequency_min: Minimal frequency supported by the frontend.  
 * @frequency_max: Minimal frequency supported by the frontend.  
 * @frequency_stepsize: All frequencies are multiple of this value.  
 * @frequency_tolerance: Frequency tolerance.  
 * @symbol_rate_min: Minimal symbol rate, in bauds  
 *                   (for Cable/Satellite systems).  
 * @symbol_rate_max: Maximal symbol rate, in bauds  
 *                   (for Cable/Satellite systems).  
 * @symbol_rate_tolerance: Maximal symbol rate tolerance, in ppm  
 *                   (for Cable/Satellite systems).  
 * @notifier_delay: ****DEPRECATED****. Not used by any driver.  
 * @caps:           Capabilities supported by the frontend,  
 *                 as specified in &enum fe_caps.  
 *  
 * .. note:  
 *  
 *   #. The frequencies are specified in Hz for Terrestrial and Cable  
 *       systems.  
 *   #. The frequencies are specified in kHz for Satellite systems.  
 */  
struct dvb_frontend_info {  
    char        name[128];  
    enum fe_type type;      /* DEPRECATED. Use DTV_ENUM_DELSYS instead */  
    __u32       frequency_min;  
    __u32       frequency_max;  
    __u32       frequency_stepsize;  
    __u32       frequency_tolerance;  
    __u32       symbol_rate_min;  
    __u32       symbol_rate_max;  
    __u32       symbol_rate_tolerance;  
    __u32       notifier_delay;      /* DEPRECATED */  
    enum fe_caps caps;  
};  
  
/**  
 * struct dvb_diseqc_master_cmd - DiSEqC master command  
 *
```

```
* @msg:
*     DiSEqC message to be sent. It contains a 3 bytes header with:
*     framing + address + command, and an optional argument
*     of up to 3 bytes of data.
* @msg_len:
*     Length of the DiSEqC message. Valid values are 3 to 6.
*
* Check out the DiSEqC bus spec available on http://www.eutelsat.org/ for
* the possible messages that can be used.
*/
struct dvb_diseqc_master_cmd {
    __u8 msg[6];
    __u8 msg_len;
};

/**
 * struct dvb_diseqc_slave_reply - DiSEqC received data
 *
 * @msg:
 *     DiSEqC message buffer to store a message received via DiSEqC.
 *     It contains one byte header with: framing and
 *     an optional argument of up to 3 bytes of data.
 * @msg_len:
 *     Length of the DiSEqC message. Valid values are 0 to 4,
 *     where 0 means no message.
 * @timeout:
 *     Return from ioctl after timeout ms with errorcode when
 *     no message was received.
 *
 * Check out the DiSEqC bus spec available on http://www.eutelsat.org/ for
 * the possible messages that can be used.
 */
struct dvb_diseqc_slave_reply {
    __u8 msg[4];
    __u8 msg_len;
    int timeout;
};

/**
 * enum fe_sec_voltage - DC Voltage used to feed the LNBf
 *
 * @SEC_VOLTAGE_13:    Output 13V to the LNBf
 * @SEC_VOLTAGE_18:    Output 18V to the LNBf
 * @SEC_VOLTAGE_OFF:   Don't feed the LNBf with a DC voltage
 */
enum fe_sec_voltage {
    SEC_VOLTAGE_13,
    SEC_VOLTAGE_18,
    SEC_VOLTAGE_OFF
};
```

```
/**
 * enum fe_sec_tone_mode - Type of tone to be send to the LNBf.
 * @SEC_TONE_ON:          Sends a 22kHz tone burst to the antenna.
 * @SEC_TONE_OFF:        Don't send a 22kHz tone to the antenna (except
 *                        if the ``FE_DISEQC_*`` ioctls are called).
 */
enum fe_sec_tone_mode {
    SEC_TONE_ON,
    SEC_TONE_OFF
};

/**
 * enum fe_sec_mini_cmd - Type of mini burst to be sent
 *
 * @SEC_MINI_A:          Sends a mini-DiSEqC 22kHz '0' Tone Burst to select
 *                        satellite-A
 * @SEC_MINI_B:          Sends a mini-DiSEqC 22kHz '1' Data Burst to select
 *                        satellite-B
 */
enum fe_sec_mini_cmd {
    SEC_MINI_A,
    SEC_MINI_B
};

/**
 * enum fe_status - Enumerates the possible frontend status.
 * @FE_NONE:            The frontend doesn't have any kind of lock.
 *                       That's the initial frontend status
 * @FE_HAS_SIGNAL:      Has found something above the noise level.
 * @FE_HAS_CARRIER:    Has found a signal.
 * @FE_HAS_VITERBI:     FEC inner coding (Viterbi, LDPC or other inner code).
 *                       is stable.
 * @FE_HAS_SYNC:        Synchronization bytes was found.
 * @FE_HAS_LOCK:        Digital TV were locked and everything is working.
 * @FE_TIMEDOUT:        Fo lock within the last about 2 seconds.
 * @FE_REINIT:          Frontend was reinitialized, application is recommended
 *                       to reset DiSEqC, tone and parameters.
 */
enum fe_status {
    FE_NONE                = 0x00,
    FE_HAS_SIGNAL          = 0x01,
    FE_HAS_CARRIER        = 0x02,
    FE_HAS_VITERBI         = 0x04,
    FE_HAS_SYNC            = 0x08,
    FE_HAS_LOCK            = 0x10,
    FE_TIMEDOUT            = 0x20,
    FE_REINIT              = 0x40,
};

/**
 * enum fe_spectral_inversion - Type of inversion band
```

```

*
* @INVERSION_OFF:      Don't do spectral band inversion.
* @INVERSION_ON:      Do spectral band inversion.
* @INVERSION_AUTO:    Autodetect spectral band inversion.
*
* This parameter indicates if spectral inversion should be presumed or
* not. In the automatic setting (`INVERSION_AUTO`) the hardware will try
* to figure out the correct setting by itself. If the hardware doesn't
* support, the %dvb_frontend will try to lock at the carrier first with
* inversion off. If it fails, it will try to enable inversion.
*/
enum fe_spectral_inversion {
    INVERSION_OFF,
    INVERSION_ON,
    INVERSION_AUTO
};

/**
 * enum fe_code_rate - Type of Forward Error Correction (FEC)
 *
 *
 * @FEC_NONE: No Forward Error Correction Code
 * @FEC_1_2:  Forward Error Correction Code 1/2
 * @FEC_2_3:  Forward Error Correction Code 2/3
 * @FEC_3_4:  Forward Error Correction Code 3/4
 * @FEC_4_5:  Forward Error Correction Code 4/5
 * @FEC_5_6:  Forward Error Correction Code 5/6
 * @FEC_6_7:  Forward Error Correction Code 6/7
 * @FEC_7_8:  Forward Error Correction Code 7/8
 * @FEC_8_9:  Forward Error Correction Code 8/9
 * @FEC_AUTO: Autodetect Error Correction Code
 * @FEC_3_5:  Forward Error Correction Code 3/5
 * @FEC_9_10: Forward Error Correction Code 9/10
 * @FEC_2_5:  Forward Error Correction Code 2/5
 *
 * Please note that not all FEC types are supported by a given standard.
 */
enum fe_code_rate {
    FEC_NONE = 0,
    FEC_1_2,
    FEC_2_3,
    FEC_3_4,
    FEC_4_5,
    FEC_5_6,
    FEC_6_7,
    FEC_7_8,
    FEC_8_9,
    FEC_AUTO,
    FEC_3_5,
    FEC_9_10,
    FEC_2_5,

```

```
};

/**
 * enum fe_modulation - Type of modulation/constellation
 * @QPSK:          QPSK modulation
 * @QAM_16:        16-QAM modulation
 * @QAM_32:        32-QAM modulation
 * @QAM_64:        64-QAM modulation
 * @QAM_128:       128-QAM modulation
 * @QAM_256:       256-QAM modulation
 * @QAM_AUTO:      Autodetect QAM modulation
 * @VSB_8:         8-VSB modulation
 * @VSB_16:        16-VSB modulation
 * @PSK_8:         8-PSK modulation
 * @APSK_16:       16-APSK modulation
 * @APSK_32:       32-APSK modulation
 * @DQPSK:         DQPSK modulation
 * @QAM_4_NR:      4-QAM-NR modulation
 *
 * Please note that not all modulations are supported by a given standard.
 */
enum fe_modulation {
    QPSK,
    QAM_16,
    QAM_32,
    QAM_64,
    QAM_128,
    QAM_256,
    QAM_AUTO,
    VSB_8,
    VSB_16,
    PSK_8,
    APSK_16,
    APSK_32,
    DQPSK,
    QAM_4_NR,
};

/**
 * enum fe_transmit_mode - Transmission mode
 *
 * @TRANSMISSION_MODE_AUTO:
 *     Autodetect transmission mode. The hardware will try to find the
 *     correct FFT-size (if capable) to fill in the missing parameters.
 * @TRANSMISSION_MODE_1K:
 *     Transmission mode 1K
 * @TRANSMISSION_MODE_2K:
 *     Transmission mode 2K
 * @TRANSMISSION_MODE_8K:
 *     Transmission mode 8K
 */
```



```

* @TRANSMISSION_MODE_4K:
*     Transmission mode 4K
* @TRANSMISSION_MODE_16K:
*     Transmission mode 16K
* @TRANSMISSION_MODE_32K:
*     Transmission mode 32K
* @TRANSMISSION_MODE_C1:
*     Single Carrier (C=1) transmission mode (DTMB only)
* @TRANSMISSION_MODE_C3780:
*     Multi Carrier (C=3780) transmission mode (DTMB only)
*
* Please note that not all transmission modes are supported by a given
* standard.
*/
enum fe_transmit_mode {
    TRANSMISSION_MODE_2K,
    TRANSMISSION_MODE_8K,
    TRANSMISSION_MODE_AUTO,
    TRANSMISSION_MODE_4K,
    TRANSMISSION_MODE_1K,
    TRANSMISSION_MODE_16K,
    TRANSMISSION_MODE_32K,
    TRANSMISSION_MODE_C1,
    TRANSMISSION_MODE_C3780,
};

/**
 * enum fe_guard_interval - Guard interval
 *
 * @GUARD_INTERVAL_AUTO:      Autodetect the guard interval
 * @GUARD_INTERVAL_1_128:    Guard interval 1/128
 * @GUARD_INTERVAL_1_32:     Guard interval 1/32
 * @GUARD_INTERVAL_1_16:     Guard interval 1/16
 * @GUARD_INTERVAL_1_8:      Guard interval 1/8
 * @GUARD_INTERVAL_1_4:      Guard interval 1/4
 * @GUARD_INTERVAL_19_128:   Guard interval 19/128
 * @GUARD_INTERVAL_19_256:   Guard interval 19/256
 * @GUARD_INTERVAL_PN420:    PN length 420 (1/4)
 * @GUARD_INTERVAL_PN595:    PN length 595 (1/6)
 * @GUARD_INTERVAL_PN945:    PN length 945 (1/9)
 *
 * Please note that not all guard intervals are supported by a given standard.
 */
enum fe_guard_interval {
    GUARD_INTERVAL_1_32,
    GUARD_INTERVAL_1_16,
    GUARD_INTERVAL_1_8,
    GUARD_INTERVAL_1_4,
    GUARD_INTERVAL_AUTO,
    GUARD_INTERVAL_1_128,
    GUARD_INTERVAL_19_128,
};

```

```
        GUARD_INTERVAL_19_256,  
        GUARD_INTERVAL_PN420,  
        GUARD_INTERVAL_PN595,  
        GUARD_INTERVAL_PN945,  
};  
  
/**  
 * enum fe_hierarchy - Hierarchy  
 * @HIERARCHY_NONE:      No hierarchy  
 * @HIERARCHY_AUTO:     Autodetect hierarchy (if supported)  
 * @HIERARCHY_1:       Hierarchy 1  
 * @HIERARCHY_2:       Hierarchy 2  
 * @HIERARCHY_4:       Hierarchy 4  
 *  
 * Please note that not all hierarchy types are supported by a given standard.  
 */  
enum fe_hierarchy {  
    HIERARCHY_NONE,  
    HIERARCHY_1,  
    HIERARCHY_2,  
    HIERARCHY_4,  
    HIERARCHY_AUTO  
};  
  
/**  
 * enum fe_interleaving - Interleaving  
 * @INTERLEAVING_NONE:  No interleaving.  
 * @INTERLEAVING_AUTO:  Auto-detect interleaving.  
 * @INTERLEAVING_240:   Interleaving of 240 symbols.  
 * @INTERLEAVING_720:   Interleaving of 720 symbols.  
 *  
 * Please note that, currently, only DTMB uses it.  
 */  
enum fe_interleaving {  
    INTERLEAVING_NONE,  
    INTERLEAVING_AUTO,  
    INTERLEAVING_240,  
    INTERLEAVING_720,  
};  
  
/* DVBv5 property Commands */  
  
#define DTV_UNDEFINED          0  
#define DTV_TUNE                1  
#define DTV_CLEAR              2  
#define DTV_FREQUENCY          3  
#define DTV_MODULATION         4  
#define DTV_BANDWIDTH_HZ       5  
#define DTV_INVERSION          6  
#define DTV_DISEQC_MASTER      7  
#define DTV_SYMBOL_RATE        8
```

```

#define DTV_INNER_FEC          9
#define DTV_VOLTAGE           10
#define DTV_TONE               11
#define DTV_PILOT              12
#define DTV_ROLLOFF            13
#define DTV_DISEQC_SLAVE_REPLY 14

/* Basic enumeration set for querying unlimited capabilities */
#define DTV_FE_CAPABILITY_COUNT 15
#define DTV_FE_CAPABILITY       16
#define DTV_DELIVERY_SYSTEM     17

/* ISDB-T and ISDB-Tsb */
#define DTV_ISDBT_PARTIAL_RECEPTION 18
#define DTV_ISDBT_SOUND_BROADCASTING 19

#define DTV_ISDBT_SB_SUBCHANNEL_ID 20
#define DTV_ISDBT_SB_SEGMENT_IDX   21
#define DTV_ISDBT_SB_SEGMENT_COUNT 22

#define DTV_ISDBT_LAYERA_FEC       23
#define DTV_ISDBT_LAYERA_MODULATION 24
#define DTV_ISDBT_LAYERA_SEGMENT_COUNT 25
#define DTV_ISDBT_LAYERA_TIME_INTERLEAVING 26

#define DTV_ISDBT_LAYERB_FEC       27
#define DTV_ISDBT_LAYERB_MODULATION 28
#define DTV_ISDBT_LAYERB_SEGMENT_COUNT 29
#define DTV_ISDBT_LAYERB_TIME_INTERLEAVING 30

#define DTV_ISDBT_LAYERC_FEC       31
#define DTV_ISDBT_LAYERC_MODULATION 32
#define DTV_ISDBT_LAYERC_SEGMENT_COUNT 33
#define DTV_ISDBT_LAYERC_TIME_INTERLEAVING 34

#define DTV_API_VERSION           35

#define DTV_CODE_RATE_HP         36
#define DTV_CODE_RATE_LP         37
#define DTV_GUARD_INTERVAL       38
#define DTV_TRANSMISSION_MODE     39
#define DTV_HIERARCHY            40

#define DTV_ISDBT_LAYER_ENABLED 41

#define DTV_STREAM_ID             42
#define DTV_ISDBS_TS_ID_LEGACY   DTV_STREAM_ID
#define DTV_DVBT2_PLP_ID_LEGACY 43

#define DTV_ENUM_DELSYS          44

```

```
/* ATSC-MH */
#define DTV_ATSCMH_FIC_VER          45
#define DTV_ATSCMH_PARADE_ID       46
#define DTV_ATSCMH_NOG             47
#define DTV_ATSCMH_TNOG           48
#define DTV_ATSCMH_SGN             49
#define DTV_ATSCMH_PRC             50
#define DTV_ATSCMH_RS_FRAME_MODE   51
#define DTV_ATSCMH_RS_FRAME_ENSEMBLE 52
#define DTV_ATSCMH_RS_CODE_MODE_PRI 53
#define DTV_ATSCMH_RS_CODE_MODE_SEC 54
#define DTV_ATSCMH_SCCC_BLOCK_MODE 55
#define DTV_ATSCMH_SCCC_CODE_MODE_A 56
#define DTV_ATSCMH_SCCC_CODE_MODE_B 57
#define DTV_ATSCMH_SCCC_CODE_MODE_C 58
#define DTV_ATSCMH_SCCC_CODE_MODE_D 59

#define DTV_INTERLEAVING           60
#define DTV_LNA                    61

/* Quality parameters */
#define DTV_STAT_SIGNAL_STRENGTH   62
#define DTV_STAT_CNR               63
#define DTV_STAT_PRE_ERROR_BIT_COUNT 64
#define DTV_STAT_PRE_TOTAL_BIT_COUNT 65
#define DTV_STAT_POST_ERROR_BIT_COUNT 66
#define DTV_STAT_POST_TOTAL_BIT_COUNT 67
#define DTV_STAT_ERROR_BLOCK_COUNT 68
#define DTV_STAT_TOTAL_BLOCK_COUNT 69

/* Physical layer scrambling */
#define DTV_SCRAMBLING_SEQUENCE_INDEX 70

#define DTV_MAX_COMMAND            DTV_SCRAMBLING_SEQUENCE_INDEX

/**
 * enum fe_pilot - Type of pilot tone
 *
 * @PILOT_ON: Pilot tones enabled
 * @PILOT_OFF: Pilot tones disabled
 * @PILOT_AUTO: Autodetect pilot tones
 */
enum fe_pilot {
    PILOT_ON,
    PILOT_OFF,
    PILOT_AUTO,
};

/**
 * enum fe_rolloff - Rolloff factor
 * @ROLLOFF_35: Rolloff factor:  $\alpha=35\%$ 
 */
```

```

* @ROLLOFF_20:           Roloff factor:  $\alpha=20\%$ 
* @ROLLOFF_25:           Roloff factor:  $\alpha=25\%$ 
* @ROLLOFF_AUTO:         Auto-detect the rolloff factor.
*
* .. note:
*
*     Roloff factor of 35% is implied on DVB-S. On DVB-S2, it is default.
*/
enum fe_rolloff {
    ROLLOFF_35,
    ROLLOFF_20,
    ROLLOFF_25,
    ROLLOFF_AUTO,
};

/**
 * enum fe_delivery_system - Type of the delivery system
 *
 * @SYS_UNDEFINED:
 *     Undefined standard. Generally, indicates an error
 * @SYS_DVBC_ANNEX_A:
 *     Cable TV: DVB-C following ITU-T J.83 Annex A spec
 * @SYS_DVBC_ANNEX_B:
 *     Cable TV: DVB-C following ITU-T J.83 Annex B spec (ClearQAM)
 * @SYS_DVBC_ANNEX_C:
 *     Cable TV: DVB-C following ITU-T J.83 Annex C spec
 * @SYS_ISDBC:
 *     Cable TV: ISDB-C (no drivers yet)
 * @SYS_DVBT:
 *     Terrestrial TV: DVB-T
 * @SYS_DVBT2:
 *     Terrestrial TV: DVB-T2
 * @SYS_ISDBT:
 *     Terrestrial TV: ISDB-T
 * @SYS_ATSC:
 *     Terrestrial TV: ATSC
 * @SYS_ATSCMH:
 *     Terrestrial TV (mobile): ATSC-M/H
 * @SYS_DTMB:
 *     Terrestrial TV: DTMB
 * @SYS_DVBS:
 *     Satellite TV: DVB-S
 * @SYS_DVBS2:
 *     Satellite TV: DVB-S2
 * @SYS_TURBO:
 *     Satellite TV: DVB-S Turbo
 * @SYS_ISDBS:
 *     Satellite TV: ISDB-S
 * @SYS_DAB:
 *     Digital audio: DAB (not fully supported)
 * @SYS_DSS:

```

```
*      Satellite TV: DSS (not fully supported)
* @SYS_CMMB:
*      Terrestrial TV (mobile): CMMB (not fully supported)
* @SYS_DVBH:
*      Terrestrial TV (mobile): DVB-H (standard deprecated)
*/
enum fe_delivery_system {
    SYS_UNDEFINED,
    SYS_DVBC_ANNEX_A,
    SYS_DVBC_ANNEX_B,
    SYS_DVBT,
    SYS_DSS,
    SYS_DVBS,
    SYS_DVBS2,
    SYS_DVBH,
    SYS_ISDBT,
    SYS_ISDBS,
    SYS_ISDBC,
    SYS_ATSC,
    SYS_ATSCMH,
    SYS_DTMB,
    SYS_CMMB,
    SYS_DAB,
    SYS_DVBT2,
    SYS_TURBO,
    SYS_DVBC_ANNEX_C,
};

/* backward compatibility definitions for delivery systems */
#define SYS_DVBC_ANNEX_AC      SYS_DVBC_ANNEX_A
#define SYS_DMBTH              SYS_DTMB /* DMB-TH is legacy name, use DTMB */

/* ATSC-MH specific parameters */

/**
 * enum atscmh_sccc_block_mode - Type of Series Concatenated Convolutional
 *                               Code Block Mode.
 *
 * @ATSCMH_SCCC_BLK_SEP:
 *     Separate SCCC: the SCCC outer code mode shall be set independently
 *     for each Group Region (A, B, C, D)
 * @ATSCMH_SCCC_BLK_COMB:
 *     Combined SCCC: all four Regions shall have the same SCCC outer
 *     code mode.
 * @ATSCMH_SCCC_BLK_RES:
 *     Reserved. Shouldn't be used.
 */
enum atscmh_sccc_block_mode {
    ATSCMH_SCCC_BLK_SEP      = 0,
    ATSCMH_SCCC_BLK_COMB    = 1,
    ATSCMH_SCCC_BLK_RES     = 2,
```

```

};

/**
 * enum atscmh_sccc_code_mode - Type of Series Concatenated Convolutional
 *                               Code Rate.
 *
 * @ATSCMH_SCCC_CODE_HLF:
 *     The outer code rate of a SCCC Block is 1/2 rate.
 * @ATSCMH_SCCC_CODE_QTR:
 *     The outer code rate of a SCCC Block is 1/4 rate.
 * @ATSCMH_SCCC_CODE_RES:
 *     Reserved. Should not be used.
 */
enum atscmh_sccc_code_mode {
    ATSCMH_SCCC_CODE_HLF    = 0,
    ATSCMH_SCCC_CODE_QTR    = 1,
    ATSCMH_SCCC_CODE_RES    = 2,
};

/**
 * enum atscmh_rs_frame_ensemble - Reed Solomon(RS) frame ensemble.
 *
 * @ATSCMH_RSFRAME_ENS_PRI:    Primary Ensemble.
 * @ATSCMH_RSFRAME_ENS_SEC:    Secondary Ensemble.
 */
enum atscmh_rs_frame_ensemble {
    ATSCMH_RSFRAME_ENS_PRI    = 0,
    ATSCMH_RSFRAME_ENS_SEC    = 1,
};

/**
 * enum atscmh_rs_frame_mode - Reed Solomon (RS) frame mode.
 *
 * @ATSCMH_RSFRAME_PRI_ONLY:
 *     Single Frame: There is only a primary RS Frame for all Group
 *     Regions.
 * @ATSCMH_RSFRAME_PRI_SEC:
 *     Dual Frame: There are two separate RS Frames: Primary RS Frame for
 *     Group Region A and B and Secondary RS Frame for Group Region C and
 *     D.
 * @ATSCMH_RSFRAME_RES:
 *     Reserved. Shouldn't be used.
 */
enum atscmh_rs_frame_mode {
    ATSCMH_RSFRAME_PRI_ONLY    = 0,
    ATSCMH_RSFRAME_PRI_SEC    = 1,
    ATSCMH_RSFRAME_RES        = 2,
};

/**
 * enum atscmh_rs_code_mode

```

```
* @ATSCMH_RSCODE_211_187:      Reed Solomon code (211,187).
* @ATSCMH_RSCODE_223_187:      Reed Solomon code (223,187).
* @ATSCMH_RSCODE_235_187:      Reed Solomon code (235,187).
* @ATSCMH_RSCODE_RES:          Reserved. Shouldn't be used.
*/
enum atscmh_rs_code_mode {
    ATSCMH_RSCODE_211_187      = 0,
    ATSCMH_RSCODE_223_187      = 1,
    ATSCMH_RSCODE_235_187      = 2,
    ATSCMH_RSCODE_RES          = 3,
};

#define NO_STREAM_ID_FILTER      (~0U)
#define LNA_AUTO                  (~0U)

/**
 * enum fecap_scale_params - scale types for the quality parameters.
 *
 * @FE_SCALE_NOT_AVAILABLE: That QoS measure is not available. That
 * could indicate a temporary or a permanent
 * condition.
 * @FE_SCALE_DECIBEL: The scale is measured in 0.001 dB steps, typically
 * used on signal measures.
 * @FE_SCALE_RELATIVE: The scale is a relative percentual measure,
 * ranging from 0 (0%) to 0xffff (100%).
 * @FE_SCALE_COUNTER: The scale counts the occurrence of an event, like
 * bit error, block error, lapsed time.
 */
enum fecap_scale_params {
    FE_SCALE_NOT_AVAILABLE = 0,
    FE_SCALE_DECIBEL,
    FE_SCALE_RELATIVE,
    FE_SCALE_COUNTER
};

/**
 * struct dtv_stats - Used for reading a DTV status property
 *
 * @scale:
 * Filled with enum fecap_scale_params - the scale in usage
 * for that parameter
 *
 * @svalue:
 * integer value of the measure, for %FE_SCALE_DECIBEL,
 * used for dB measures. The unit is 0.001 dB.
 *
 * @uvalue:
 * unsigned integer value of the measure, used when @scale is
 * either %FE_SCALE_RELATIVE or %FE_SCALE_COUNTER.
 *
 * For most delivery systems, this will return a single value for each
```



```

* parameter.
*
* It should be noticed, however, that new OFDM delivery systems like
* ISDB can use different modulation types for each group of carriers.
* On such standards, up to 8 groups of statistics can be provided, one
* for each carrier group (called "layer" on ISDB).
*
* In order to be consistent with other delivery systems, the first
* value refers to the entire set of carriers ("global").
*
* @scale should use the value %FE_SCALE_NOT_AVAILABLE when
* the value for the entire group of carriers or from one specific layer
* is not provided by the hardware.
*
* @len should be filled with the latest filled status + 1.
*
* In other words, for ISDB, those values should be filled like::
*
*     u.st.stat.svalue[0] = global statistics;
*     u.st.stat.scale[0] = FE_SCALE_DECIBEL;
*     u.st.stat.value[1] = layer A statistics;
*     u.st.stat.scale[1] = FE_SCALE_NOT_AVAILABLE (if not available);
*     u.st.stat.svalue[2] = layer B statistics;
*     u.st.stat.scale[2] = FE_SCALE_DECIBEL;
*     u.st.stat.svalue[3] = layer C statistics;
*     u.st.stat.scale[3] = FE_SCALE_DECIBEL;
*     u.st.len = 4;
*/
struct dtv_stats {
    __u8 scale;        /* enum fecap_scale_params type */
    union {
        __u64 uvalue; /* for counters and relative scales */
        __s64 svalue; /* for 0.001 dB measures */
    };
} __attribute__((packed));

#define MAX_DTV_STATS    4

/**
 * struct dtv_fe_stats - store Digital TV frontend statistics
 *
 * @len:        length of the statistics - if zero, stats is disabled.
 * @stat:       array with digital TV statistics.
 *
 * On most standards, @len can either be 0 or 1. However, for ISDB, each
 * layer is modulated in separate. So, each layer may have its own set
 * of statistics. If so, stat[0] carries on a global value for the property.
 * Indexes 1 to 3 means layer A to B.
 */
struct dtv_fe_stats {
    __u8 len;

```

```
        struct dtv_stats stat[MAX_DTV_STATS];
} __attribute__((packed));

/**
 * struct dtv_property - store one of frontend command and its value
 *
 * @cmd:           Digital TV command.
 * @reserved:     Not used.
 * @u:           Union with the values for the command.
 * @u.data:       A unsigned 32 bits integer with command value.
 * @u.buffer:     Struct to store bigger properties.
 *               Currently unused.
 * @u.buffer.data: an unsigned 32-bits array.
 * @u.buffer.len: number of elements of the buffer.
 * @u.buffer.reserved1: Reserved.
 * @u.buffer.reserved2: Reserved.
 * @u.st:         a &struct dtv_fe_stats array of statistics.
 * @result:       Currently unused.
 */
struct dtv_property {
    __u32 cmd;
    __u32 reserved[3];
    union {
        __u32 data;
        struct dtv_fe_stats st;
        struct {
            __u8 data[32];
            __u32 len;
            __u32 reserved1[3];
            void *reserved2;
        } buffer;
    } u;
    int result;
} __attribute__((packed));

/* num of properties cannot exceed DTV_IOCTL_MAX_MSGS per ioctl */
#define DTV_IOCTL_MAX_MSGS 64

/**
 * struct dtv_properties - a set of command/value pairs.
 *
 * @num:           amount of commands stored at the struct.
 * @props:        a pointer to &struct dtv_property.
 */
struct dtv_properties {
    __u32 num;
    struct dtv_property *props;
};

/*
```

```

* When set, this flag will disable any zigzagging or other "normal" tuning
* behavior. Additionally, there will be no automatic monitoring of the lock
* status, and hence no frontend events will be generated. If a frontend device
* is closed, this flag will be automatically turned off when the device is
* reopened read-write.
*/
#define FE_TUNE_MODE_ONESHOT 0x01

/* Digital TV Frontend API calls */

#define FE_GET_INFO                _IOR('o', 61, struct dvb_frontend_info)

#define FE_DISEQC_RESET_OVERLOAD   _IO('o', 62)
#define FE_DISEQC_SEND_MASTER_CMD  _IOW('o', 63, struct dvb_diseqc_master_cmd)
#define FE_DISEQC_RECV_SLAVE_REPLY _IOR('o', 64, struct dvb_diseqc_slave_reply)
#define FE_DISEQC_SEND_BURST       _IO('o', 65) /* fe_sec_mini_cmd_t */

#define FE_SET_TONE                 _IO('o', 66) /* fe_sec_tone_mode_t */
#define FE_SET_VOLTAGE              _IO('o', 67) /* fe_sec_voltage_t */
#define FE_ENABLE_HIGH_LNB_VOLTAGE _IO('o', 68) /* int */

#define FE_READ_STATUS              _IOR('o', 69, fe_status_t)
#define FE_READ_BER                 _IOR('o', 70, __u32)
#define FE_READ_SIGNAL_STRENGTH    _IOR('o', 71, __u16)
#define FE_READ_SNR                 _IOR('o', 72, __u16)
#define FE_READ_UNCORRECTED_BLOCKS _IOR('o', 73, __u32)

#define FE_SET_FRONTEND_TUNE_MODE  _IO('o', 81) /* unsigned int */
#define FE_GET_EVENT               _IOR('o', 78, struct dvb_frontend_event)

#define FE_DISHNETWORK_SEND_LEGACY_CMD _IO('o', 80) /* unsigned int */

#define FE_SET_PROPERTY             _IOW('o', 82, struct dtv_properties)
#define FE_GET_PROPERTY            _IOR('o', 83, struct dtv_properties)

#if defined(__DVB_CORE__) || !defined(__KERNEL__)

/*
 * DEPRECATED: Everything below is deprecated in favor of DVBv5 API
 *
 * The DVBv3 only ioctls, structs and enums should not be used on
 * newer programs, as it doesn't support the second generation of
 * digital TV standards, nor supports newer delivery systems.
 * They also don't support modern frontends with usually support multiple
 * delivery systems.
 *
 * Drivers shouldn't use them.
 *
 * New applications should use DVBv5 delivery system instead
 */

```

```
/*
 */

enum fe_bandwidth {
    BANDWIDTH_8_MHZ,
    BANDWIDTH_7_MHZ,
    BANDWIDTH_6_MHZ,
    BANDWIDTH_AUTO,
    BANDWIDTH_5_MHZ,
    BANDWIDTH_10_MHZ,
    BANDWIDTH_1_712_MHZ,
};

/* This is kept for legacy userspace support */
typedef enum fe_sec_voltage fe_sec_voltage_t;
typedef enum fe_caps fe_caps_t;
typedef enum fe_type fe_type_t;
typedef enum fe_sec_tone_mode fe_sec_tone_mode_t;
typedef enum fe_sec_mini_cmd fe_sec_mini_cmd_t;
typedef enum fe_status fe_status_t;
typedef enum fe_spectral_inversion fe_spectral_inversion_t;
typedef enum fe_code_rate fe_code_rate_t;
typedef enum fe_modulation fe_modulation_t;
typedef enum fe_transmit_mode fe_transmit_mode_t;
typedef enum fe_bandwidth fe_bandwidth_t;
typedef enum fe_guard_interval fe_guard_interval_t;
typedef enum fe_hierarchy fe_hierarchy_t;
typedef enum fe_pilot fe_pilot_t;
typedef enum fe_rolloff fe_rolloff_t;
typedef enum fe_delivery_system fe_delivery_system_t;

/* DVBv3 structs */

struct dvb_qpsk_parameters {
    __u32          symbol_rate; /* symbol rate in Symbols per second */
    fe_code_rate_t fec_inner;   /* forward error correction (see above) */
};

struct dvb_qam_parameters {
    __u32          symbol_rate; /* symbol rate in Symbols per second */
    fe_code_rate_t fec_inner;   /* forward error correction (see above) */
    fe_modulation_t modulation; /* modulation type (see above) */
};

struct dvb_vsb_parameters {
    fe_modulation_t modulation; /* modulation type (see above) */
};

struct dvb_ofdm_parameters {
    fe_bandwidth_t bandwidth;
```

```

    fe_code_rate_t      code_rate_HP; /* high priority stream code rate */
    fe_code_rate_t      code_rate_LP; /* low priority stream code rate */
    fe_modulation_t     constellation; /* modulation type (see above) */
    fe_transmit_mode_t  transmission_mode;
    fe_guard_interval_t guard_interval;
    fe_hierarchy_t      hierarchy_information;
};

struct dvb_frontend_parameters {
    __u32 frequency; /* (absolute) frequency in Hz for DVB-C/DVB-T/ATSC */
                    /* intermediate frequency in kHz for DVB-S */
    fe_spectral_inversion_t inversion;
    union {
        struct dvb_qpsk_parameters qpsk; /* DVB-S */
        struct dvb_qam_parameters qam; /* DVB-C */
        struct dvb_ofdm_parameters ofdm; /* DVB-T */
        struct dvb_vsb_parameters vsb; /* ATSC */
    } u;
};

struct dvb_frontend_event {
    fe_status_t status;
    struct dvb_frontend_parameters parameters;
};

/* DVBv3 API calls */

#define FE_SET_FRONTEND      _IOW('o', 76, struct dvb_frontend_
    ↪parameters)
#define FE_GET_FRONTEND     _IOR('o', 77, struct dvb_frontend_
    ↪parameters)

#endif

#endif /* _DVBFRONTEND_H_ */

```

dmx.h

```

/* SPDX-License-Identifier: LGPL-2.1+ WITH Linux-syscall-note */
/*
 * dmx.h
 *
 * Copyright (C) 2000 Marcus Metzler <marcus@convergence.de>
 *           & Ralph Metzler <ralph@convergence.de>
 *           for convergence integrated media GmbH
 *
 * This program is free software; you can redistribute it and/or
 * modify it under the terms of the GNU Lesser General Public License
 * as published by the Free Software Foundation; either version 2.1
 * of the License, or (at your option) any later version.

```

```
*
* This program is distributed in the hope that it will be useful,
* but WITHOUT ANY WARRANTY; without even the implied warranty of
* MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
* GNU General Public License for more details.
*
* You should have received a copy of the GNU Lesser General Public License
* along with this program; if not, write to the Free Software
* Foundation, Inc., 59 Temple Place - Suite 330, Boston, MA 02111-1307, USA.
*
*/

#ifndef _UAPI_DVBDMX_H_
#define _UAPI_DVBDMX_H_

#include <linux/types.h>
#ifndef __KERNEL__
#include <time.h>
#endif

#define DMX_FILTER_SIZE 16

/**
 * enum dmx_output - Output for the demux.
 *
 * @:c:type:DMX_OUT_DECODER <dmx_output>:
 *     Streaming directly to decoder.
 * @:c:type:DMX_OUT_TAP <dmx_output>:
 *     Output going to a memory buffer (to be retrieved via the read command).
 *     Delivers the stream output to the demux device on which the ioctl
 *     is called.
 * @:c:type:DMX_OUT_TS_TAP <dmx_output>:
 *     Output multiplexed into a new TS (to be retrieved by reading from the
 *     logical DVR device). Routes output to the logical DVR device
 *     ``/dev/dvb/adapter?/dvr?``, which delivers a TS multiplexed from all
 *     filters for which @:c:type:DMX_OUT_TS_TAP <dmx_output> was specified.
 * @:c:type:DMX_OUT_TSDEMUX_TAP <dmx_output>:
 *     Like @:c:type:DMX_OUT_TS_TAP <dmx_output> but retrieved from the DMX_
 *     ↪device.
 */
enum dmx_output {
    DMX_OUT_DECODER,
    DMX_OUT_TAP,
    DMX_OUT_TS_TAP,
    DMX_OUT_TSDEMUX_TAP
};

/**
 * dmx_input - Input from the demux.
 *
 * @:c:type:DMX_IN_FRONTEND <dmx_input>:    Input from a front-end device.
 */
```

```

* @:c:type:DMX_IN_DVR <dmx_input>:          Input from the logical DVR device.
*/
dmx_input {
    DMX_IN_FRONTEND,
    DMX_IN_DVR
};

/**
 * dmx_ts_pes - type of the PES filter.
 *
 * @:c:type:DMX_PES_AUDIO0 <dmx_pes_type>:    first audio PID. Also referred
→as @DMX_PES_AUDIO.
 * @:c:type:DMX_PES_VIDEO0 <dmx_pes_type>:    first video PID. Also referred
→as @DMX_PES_VIDEO.
 * @:c:type:DMX_PES_TELETEXT0 <dmx_pes_type>: first teletext PID. Also
→referred as @DMX_PES_TELETEXT.
 * @:c:type:DMX_PES_SUBTITLE0 <dmx_pes_type>: first subtitle PID. Also
→referred as @DMX_PES_SUBTITLE.
 * @:c:type:DMX_PES_PCR0 <dmx_pes_type>:      first Program Clock Reference
→PID.
 *                                     Also referred as @DMX_PES_PCR.
 *
 * @:c:type:DMX_PES_AUDIO1 <dmx_pes_type>:    second audio PID.
 * @:c:type:DMX_PES_VIDEO1 <dmx_pes_type>:    second video PID.
 * @:c:type:DMX_PES_TELETEXT1 <dmx_pes_type>: second teletext PID.
 * @:c:type:DMX_PES_SUBTITLE1 <dmx_pes_type>: second subtitle PID.
 * @:c:type:DMX_PES_PCR1 <dmx_pes_type>:      second Program Clock Reference
→PID.
 *
 * @:c:type:DMX_PES_AUDIO2 <dmx_pes_type>:    third audio PID.
 * @:c:type:DMX_PES_VIDEO2 <dmx_pes_type>:    third video PID.
 * @:c:type:DMX_PES_TELETEXT2 <dmx_pes_type>: third teletext PID.
 * @:c:type:DMX_PES_SUBTITLE2 <dmx_pes_type>: third subtitle PID.
 * @:c:type:DMX_PES_PCR2 <dmx_pes_type>:      third Program Clock Reference
→PID.
 *
 * @:c:type:DMX_PES_AUDIO3 <dmx_pes_type>:    fourth audio PID.
 * @:c:type:DMX_PES_VIDEO3 <dmx_pes_type>:    fourth video PID.
 * @:c:type:DMX_PES_TELETEXT3 <dmx_pes_type>: fourth teletext PID.
 * @:c:type:DMX_PES_SUBTITLE3 <dmx_pes_type>: fourth subtitle PID.
 * @:c:type:DMX_PES_PCR3 <dmx_pes_type>:      fourth Program Clock Reference
→PID.
 *
 * @:c:type:DMX_PES_OTHER <dmx_pes_type>:    any other PID.
*/

dmx_ts_pes {
    DMX_PES_AUDIO0,
    DMX_PES_VIDEO0,
    DMX_PES_TELETEXT0,
    DMX_PES_SUBTITLE0,

```

```
    DMX_PES_PCR0,

    DMX_PES_AUDIO1,
    DMX_PES_VIDEO1,
    DMX_PES_TELETEXT1,
    DMX_PES_SUBTITLE1,
    DMX_PES_PCR1,

    DMX_PES_AUDIO2,
    DMX_PES_VIDEO2,
    DMX_PES_TELETEXT2,
    DMX_PES_SUBTITLE2,
    DMX_PES_PCR2,

    DMX_PES_AUDIO3,
    DMX_PES_VIDEO3,
    DMX_PES_TELETEXT3,
    DMX_PES_SUBTITLE3,
    DMX_PES_PCR3,

    DMX_PES_OTHER
};

#define DMX_PES_AUDIO    DMX_PES_AUDIO0
#define DMX_PES_VIDEO    DMX_PES_VIDEO0
#define DMX_PES_TELETEXT DMX_PES_TELETEXT0
#define DMX_PES_SUBTITLE DMX_PES_SUBTITLE0
#define DMX_PES_PCR      DMX_PES_PCR0

/**
 * struct dmxfilter - Specifies a section header filter.
 *
 * @filter: bit array with bits to be matched at the section header.
 * @mask: bits that are valid at the filter bit array.
 * @mode: mode of match: if bit is zero, it will match if equal (positive
 *        match); if bit is one, it will match if the bit is negated.
 *
 * Note: All arrays in this struct have a size of DMX_FILTER_SIZE (16 bytes).
 */
struct dmxfilter {
    __u8 filter[DMX_FILTER_SIZE];
    __u8 mask[DMX_FILTER_SIZE];
    __u8 mode[DMX_FILTER_SIZE];
};

/**
 * struct dmxfilter_params - Specifies a section filter.
 *
 * @pid: PID to be filtered.
 * @filter: section header filter, as defined by &struct dmxfilter.
 * @timeout: maximum time to filter, in milliseconds.
 */
```



```

* @flags: extra flags for the section filter.
*
* Carries the configuration for a MPEG-TS section filter.
*
* The @flags can be:
*
*   - %DMX_CHECK_CRC - only deliver sections where the CRC check succeeded;
*   - %DMX_ONESHOT - disable the section filter after one section
*     has been delivered;
*   - %DMX_IMMEDIATE_START - Start filter immediately without requiring a
*     :ref:`DMX_START`.
*/
struct dmx_sct_filter_params {
    __u16          pid;
    struct dmx_filter filter;
    __u32          timeout;
    __u32          flags;
#define DMX_CHECK_CRC          1
#define DMX_ONESHOT           2
#define DMX_IMMEDIATE_START   4
};

/**
 * struct dmx_pes_filter_params - Specifies Packetized Elementary Stream (PES)
 * filter parameters.
 *
 * @pid:          PID to be filtered.
 * @input:        Demux input, as specified by &enum dmx_input.
 * @output:       Demux output, as specified by &enum dmx_output.
 * @pes_type:     Type of the pes filter, as specified by &enum dmx_pes_type.
 * @flags:        Demux PES flags.
 */
struct dmx_pes_filter_params {
    __u16          pid;
    dmx_input      input;
    enum dmx_output output;
    dmx_ts_pes     pes_type;
    __u32          flags;
};

/**
 * struct dmx_stc - Stores System Time Counter (STC) information.
 *
 * @num: input data: number of the STC, from 0 to N.
 * @base: output: divisor for STC to get 90 kHz clock.
 * @stc: output: stc in @base * 90 kHz units.
 */
struct dmx_stc {
    unsigned int num;
    unsigned int base;
    __u64 stc;
};

```

```
};

/**
 * enum dmx_buffer_flags - DMX memory-mapped buffer flags
 *
 * @:c:type:DMX_BUFFER_FLAG_HAD_CRC32_DISCARD <dmx_buffer_flags>:
 *     Indicates that the Kernel discarded one or more frames due to wrong
 *     CRC32 checksum.
 * @:c:type:DMX_BUFFER_FLAG_TEI <dmx_buffer_flags>:
 *     Indicates that the Kernel has detected a Transport Error indicator
 *     (TEI) on a filtered pid.
 * @:c:type:DMX_BUFFER_PKT_COUNTER_MISMATCH <dmx_buffer_flags>:
 *     Indicates that the Kernel has detected a packet counter mismatch
 *     on a filtered pid.
 * @:c:type:DMX_BUFFER_FLAG_DISCONTINUITY_DETECTED <dmx_buffer_flags>:
 *     Indicates that the Kernel has detected one or more frame discontinuity.
 * @:c:type:DMX_BUFFER_FLAG_DISCONTINUITY_INDICATOR <dmx_buffer_flags>:
 *     Received at least one packet with a frame discontinuity indicator.
 */

enum dmx_buffer_flags {
    DMX_BUFFER_FLAG_HAD_CRC32_DISCARD           = 1 << 0,
    DMX_BUFFER_FLAG_TEI                         = 1 << 1,
    DMX_BUFFER_PKT_COUNTER_MISMATCH             = 1 << 2,
    DMX_BUFFER_FLAG_DISCONTINUITY_DETECTED      = 1 << 3,
    DMX_BUFFER_FLAG_DISCONTINUITY_INDICATOR     = 1 << 4,
};

/**
 * struct dmx_buffer - dmx buffer info
 *
 * @index:      id number of the buffer
 * @bytesused:  number of bytes occupied by data in the buffer (payload);
 * @offset:     for buffers with memory == DMX_MEMORY_MMAP;
 *              offset from the start of the device memory for this plane,
 *              (or a "cookie" that should be passed to mmap() as offset)
 * @length:     size in bytes of the buffer
 * @flags:      bit array of buffer flags as defined by &enum dmx_buffer_flags.
 *              Filled only at &DMX_DQBUF.
 * @count:     monotonic counter for filled buffers. Helps to identify
 *              data stream loses. Filled only at &DMX_DQBUF.
 *
 * Contains data exchanged by application and driver using one of the streaming
 * I/O methods.
 *
 * Please notice that, for &DMX_QBUF, only @index should be filled.
 * On &DMX_DQBUF calls, all fields will be filled by the Kernel.
 */
struct dmx_buffer {
    __u32                index;
    __u32                bytesused;
};
```

```

        __u32          offset;
        __u32          length;
        __u32          flags;
        __u32          count;
};

/**
 * struct dmxf_requestbuffers - request dmxf buffer information
 *
 * @count:      number of requested buffers,
 * @size:       size in bytes of the requested buffer
 *
 * Contains data used for requesting a dmxf buffer.
 * All reserved fields must be set to zero.
 */
struct dmxf_requestbuffers {
        __u32          count;
        __u32          size;
};

/**
 * struct dmxf_exportbuffer - export of dmxf buffer as DMABUF file descriptor
 *
 * @index:      id number of the buffer
 * @flags:      flags for newly created file, currently only O_CLOEXEC is
 *             supported, refer to manual of open syscall for more details
 * @fd:        file descriptor associated with DMABUF (set by driver)
 *
 * Contains data used for exporting a dmxf buffer as DMABUF file descriptor.
 * The buffer is identified by a 'cookie' returned by DMXF\_QUERYBUF
 * (identical to the cookie used to mmap() the buffer to userspace). All
 * reserved fields must be set to zero. The field reserved0 is expected to
 * become a structure 'type' allowing an alternative layout of the structure
 * content. Therefore this field should not be used for any other extensions.
 */
struct dmxf_exportbuffer {
        __u32          index;
        __u32          flags;
        __s32          fd;
};

#define DMXF_START          _IO('o', 41)
#define DMXF_STOP          _IO('o', 42)
#define DMXF_SET_FILTER    _IOW('o', 43, struct dmxf_sct_filter_params)
#define DMXF_SET_PES_FILTER _IOW('o', 44, struct dmxf_pes_filter_params)
#define DMXF_SET_BUFFER_SIZE _IO('o', 45)
#define DMXF_GET_PES_PIDS  _IOR('o', 47, __u16[5])
#define DMXF_GET_STC      _IOWR('o', 50, struct dmxf_stc)
#define DMXF_ADD_PID       _IOW('o', 51, __u16)
#define DMXF_REMOVE_PID    _IOW('o', 52, __u16)

```

```
#if !defined(__KERNEL__)

/* This is needed for legacy userspace support */
typedef enum dmx_output dmx_output_t;
typedef dmx_input dmx_input_t;
typedef dmx_ts_pes dmx_pes_type_t;
typedef struct dmx_filter dmx_filter_t;

#endif

#define DMX_REQBUFS                _IOWR('o', 60, struct dmx_requestbuffers)
#define DMX_QUERYBUF              _IOWR('o', 61, struct dmx_buffer)
#define DMX_EXPBUF                _IOWR('o', 62, struct dmx_exportbuffer)
#define DMX_QBUF                  _IOWR('o', 63, struct dmx_buffer)
#define DMX_DQBUF                 _IOWR('o', 64, struct dmx_buffer)

#endif /* _DVBDMX_H_ */
```

ca.h

```
/* SPDX-License-Identifier: LGPL-2.1+ WITH Linux-syscall-note */
/*
 * ca.h
 *
 * Copyright (C) 2000 Ralph Metzler <ralph@convergence.de>
 *                & Marcus Metzler <marcus@convergence.de>
 *                for convergence integrated media GmbH
 *
 * This program is free software; you can redistribute it and/or
 * modify it under the terms of the GNU General Lesser Public License
 * as published by the Free Software Foundation; either version 2.1
 * of the License, or (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU Lesser General Public License
 * along with this program; if not, write to the Free Software
 * Foundation, Inc., 59 Temple Place - Suite 330, Boston, MA 02111-1307, USA.
 */

#ifndef _DVBCA_H_
#define _DVBCA_H_

/**
 * struct ca_slot_info - CA slot interface types and info.
 */
```

```

* @num:          slot number.
* @type:         slot type.
* @flags:        flags applicable to the slot.
*
* This struct stores the CA slot information.
*
* @type can be:
*
*   - %CA_CI - CI high level interface;
*   - %CA_CI_LINK - CI link layer level interface;
*   - %CA_CI_PHYS - CI physical layer level interface;
*   - %CA_DESCR - built-in descrambler;
*   - %CA_SC - simple smart card interface.
*
* @flags can be:
*
*   - %CA_CI_MODULE_PRESENT - module (or card) inserted;
*   - %CA_CI_MODULE_READY - module is ready for usage.
*/

struct ca_slot_info {
    int num;
    int type;
#define CA_CI          1
#define CA_CI_LINK    2
#define CA_CI_PHYS    4
#define CA_DESCR      8
#define CA_SC         128

    unsigned int flags;
#define CA_CI_MODULE_PRESENT 1
#define CA_CI_MODULE_READY  2
};

/**
 * struct ca_descr_info - descrambler types and info.
 *
 * @num:          number of available descramblers (keys).
 * @type:         type of supported scrambling system.
 *
 * Identifies the number of descramblers and their type.
 *
 * @type can be:
 *
 *   - %CA_ECD - European Common Descrambler (ECD) hardware;
 *   - %CA_NDS - Videoguard (NDS) hardware;
 *   - %CA_DSS - Distributed Sample Scrambling (DSS) hardware.
 */
struct ca_descr_info {
    unsigned int num;
    unsigned int type;

```

```
#define CA_ECD          1
#define CA_NDS          2
#define CA_DSS          4
};

/**
 * struct ca_caps - CA slot interface capabilities.
 *
 * @slot_num:    total number of CA card and module slots.
 * @slot_type:   bitmap with all supported types as defined at
 *              &struct ca_slot_info (e. g. %CA_CI, %CA_CI_LINK, etc).
 * @descr_num:   total number of descrambler slots (keys)
 * @descr_type:  bitmap with all supported types as defined at
 *              &struct ca_descr_info (e. g. %CA_ECD, %CA_NDS, etc).
 */
struct ca_caps {
    unsigned int slot_num;
    unsigned int slot_type;
    unsigned int descr_num;
    unsigned int descr_type;
};

/**
 * struct ca_msg - a message to/from a CI-CAM
 *
 * @index:       unused
 * @type:        unused
 * @length:      length of the message
 * @msg:         message
 *
 * This struct carries a message to be send/received from a CI CA module.
 */
struct ca_msg {
    unsigned int index;
    unsigned int type;
    unsigned int length;
    unsigned char msg[256];
};

/**
 * struct ca_descr - CA descrambler control words info
 *
 * @index: CA Descrambler slot
 * @parity: control words parity, where 0 means even and 1 means odd
 * @cw: CA Descrambler control words
 */
struct ca_descr {
    unsigned int index;
    unsigned int parity;
    unsigned char cw[8];
};
```

```

#define CA_RESET                _IO('o', 128)
#define CA_GET_CAP              _IOR('o', 129, struct ca_caps)
#define CA_GET_SLOT_INFO       _IOR('o', 130, struct ca_slot_info)
#define CA_GET_DESCR_INFO      _IOR('o', 131, struct ca_descr_info)
#define CA_GET_MSG             _IOR('o', 132, struct ca_msg)
#define CA_SEND_MSG            _IOW('o', 133, struct ca_msg)
#define CA_SET_DESCR           _IOW('o', 134, struct ca_descr)

```

```

#if !defined(__KERNEL__)

```

```

/* This is needed for legacy userspace support */

```

```

typedef struct ca_slot_info ca_slot_info_t;
typedef struct ca_descr_info ca_descr_info_t;
typedef struct ca_caps ca_caps_t;
typedef struct ca_msg ca_msg_t;
typedef struct ca_descr ca_descr_t;

```

```

#endif

```

```

#endif

```

net.h

```

/* SPDX-License-Identifier: LGPL-2.1+ WITH Linux-syscall-note */
/*
 * net.h
 *
 * Copyright (C) 2000 Marcus Metzler <marcus@convergence.de>
 *           & Ralph Metzler <ralph@convergence.de>
 *           for convergence integrated media GmbH
 *
 * This program is free software; you can redistribute it and/or
 * modify it under the terms of the GNU Lesser General Public License
 * as published by the Free Software Foundation; either version 2.1
 * of the License, or (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU Lesser General Public License
 * along with this program; if not, write to the Free Software
 * Foundation, Inc., 59 Temple Place - Suite 330, Boston, MA 02111-1307, USA.
 */

```

```

#ifndef __DVBNET_H__
#define __DVBNET_H__

```

```
#include <linux/types.h>

/**
 * struct dvb_net_if - describes a DVB network interface
 *
 * @pid: Packet ID (PID) of the MPEG-TS that contains data
 * @if_num: number of the Digital TV interface.
 * @feedtype: Encapsulation type of the feed.
 *
 * A MPEG-TS stream may contain packet IDs with IP packages on it.
 * This struct describes it, and the type of encoding.
 *
 * @feedtype can be:
 *
 * - %DVB_NET_FEEDTYPE_MPE for MPE encoding
 * - %DVB_NET_FEEDTYPE_ULE for ULE encoding.
 */
struct dvb_net_if {
    __u16 pid;
    __u16 if_num;
    __u8 feedtype;
#define DVB_NET_FEEDTYPE_MPE 0 /* multi protocol encapsulation */
#define DVB_NET_FEEDTYPE_ULE 1 /* ultra lightweight encapsulation */
};

#define NET_ADD_IF    _IOWR('o', 52, struct dvb_net_if)
#define NET_REMOVE_IF _IO('o', 53)
#define NET_GET_IF    _IOWR('o', 54, struct dvb_net_if)

/* binary compatibility cruft: */
struct __dvb_net_if_old {
    __u16 pid;
    __u16 if_num;
};
#define __NET_ADD_IF_OLD _IOWR('o', 52, struct __dvb_net_if_old)
#define __NET_GET_IF_OLD _IOWR('o', 54, struct __dvb_net_if_old)

#endif /* _DVBNET_H_ */
```

3.3.9 Revision and Copyright

Authors:

- J. K. Metzler, Ralph <rjkm@metzlerbros.de>
- Original author of the Digital TV API documentation.
- O. C. Metzler, Marcus <rjkm@metzlerbros.de>
- Original author of the Digital TV API documentation.
- Carvalho Chehab, Mauro <mchehab+samsung@kernel.org>

- Ported document to Docbook XML, addition of DVBv5 API, documentation gaps fix.

Copyright © 2002-2003 : Convergence GmbH

Copyright © 2009-2017 : Mauro Carvalho Chehab

3.3.10 Revision History

revision 2.2.0 / 2017-09-01 (*mcc*)

Most gaps between the uAPI document and the Kernel implementation got fixed for the non-legacy API.

revision 2.1.0 / 2015-05-29 (*mcc*)

DocBook improvements and cleanups, in order to document the system calls on a more standard way and provide more description about the current Digital TV API.

revision 2.0.4 / 2011-05-06 (*mcc*)

Add more information about DVBv5 API, better describing the frontend GET/SET props ioctl's.

revision 2.0.3 / 2010-07-03 (*mcc*)

Add some frontend capabilities flags, present on kernel, but missing at the specs.

revision 2.0.2 / 2009-10-25 (*mcc*)

documents FE_SET_FRONTEND_TUNE_MODE and FE_DISHETWORK_SEND_LEGACY_CMD ioctls.

revision 2.0.1 / 2009-09-16 (*mcc*)

Added ISDB-T test originally written by Patrick Boettcher

revision 2.0.0 / 2009-09-06 (*mcc*)

Conversion from LaTeX to DocBook XML. The contents is the same as the original LaTeX version.

revision 1.0.0 / 2003-07-24 (*rjkm*)

Initial revision on LaTeX.

3.4 Part III - Remote Controller API

3.4.1 Introduction

Currently, most analog and digital devices have a Infrared input for remote controllers. Each manufacturer has their own type of control. It is not rare for the same manufacturer to ship different types of controls, depending on the device.

A Remote Controller interface is mapped as a normal evdev/input interface, just like a keyboard or a mouse. So, it uses all ioctls already defined for any other input devices.

However, remote controllers are more flexible than a normal input device, as the IR receiver (and/or transmitter) can be used in conjunction with a wide variety of different IR remotes.

In order to allow flexibility, the Remote Controller subsystem allows controlling the RC-specific attributes via *the sysfs class nodes*.

3.4.2 Remote Controller's sysfs nodes

As defined at Documentation/ABI/testing/sysfs-class-rc, those are the sysfs nodes that control the Remote Controllers:

3.4.2.1 /sys/class/rc/

The /sys/class/rc/ class sub-directory belongs to the Remote Controller core and provides a sysfs interface for configuring infrared remote controller receivers.

3.4.2.2 /sys/class/rc/rcN/

A /sys/class/rc/rcN directory is created for each remote control receiver device where N is the number of the receiver.

3.4.2.3 /sys/class/rc/rcN/protocols

Reading this file returns a list of available protocols, something like:

```
rc5 [rc6] nec jvc [sony]
```

Enabled protocols are shown in [] brackets.

Writing "+proto" will add a protocol to the list of enabled protocols.

Writing "-proto" will remove a protocol from the list of enabled protocols.

Writing "proto" will enable only "proto".

Writing "none" will disable all protocols.

Write fails with EINVAL if an invalid protocol combination or unknown protocol name is used.

3.4.2.4 /sys/class/rc/rcN/filter

Sets the scancode filter expected value.

Use in combination with /sys/class/rc/rcN/filter_mask to set the expected value of the bits set in the filter mask. If the hardware supports it then scancodes which do not match the filter will be ignored. Otherwise the write will fail with an error.

This value may be reset to 0 if the current protocol is altered.

3.4.2.5 /sys/class/rc/rcN/filter_mask

Sets the scancode filter mask of bits to compare. Use in combination with `/sys/class/rc/rcN/filter` to set the bits of the scancode which should be compared against the expected value. A value of 0 disables the filter to allow all valid scancodes to be processed.

If the hardware supports it then scancodes which do not match the filter will be ignored. Otherwise the write will fail with an error.

This value may be reset to 0 if the current protocol is altered.

3.4.2.6 /sys/class/rc/rcN/wakeup_protocols

Reading this file returns a list of available protocols to use for the wakeup filter, something like:

```
rc-5 nec nec-x rc-6-0 rc-6-6a-24 [rc-6-6a-32] rc-6-mce
```

Note that protocol variants are listed, so `nec`, `sony`, `rc-5`, `rc-6` have their different bit length encodings listed if available.

Note that all protocol variants are listed.

The enabled wakeup protocol is shown in `[]` brackets.

Only one protocol can be selected at a time.

Writing `"proto"` will use `"proto"` for wakeup events.

Writing `"none"` will disable wakeup.

Write fails with `EINVAL` if an invalid protocol combination or unknown protocol name is used, or if wakeup is not supported by the hardware.

3.4.2.7 /sys/class/rc/rcN/wakeup_filter

Sets the scancode wakeup filter expected value. Use in combination with `/sys/class/rc/rcN/wakeup_filter_mask` to set the expected value of the bits set in the wakeup filter mask to trigger a system wake event.

If the hardware supports it and `wakeup_filter_mask` is not 0 then scancodes which match the filter will wake the system from e.g. suspend to RAM or power off. Otherwise the write will fail with an error.

This value may be reset to 0 if the wakeup protocol is altered.

3.4.2.8 /sys/class/rc/rcN/wakeup_filter_mask

Sets the scancode wakeup filter mask of bits to compare. Use in combination with `/sys/class/rc/rcN/wakeup_filter` to set the bits of the scancode which should be compared against the expected value to trigger a system wake event.

If the hardware supports it and `wakeup_filter_mask` is not 0 then scancodes which match the filter will wake the system from e.g. suspend to RAM or power off. Otherwise the write will fail with an error.

This value may be reset to 0 if the wakeup protocol is altered.

3.4.3 Remote Controller Protocols and Scancodes

IR is encoded as a series of pulses and spaces, using a protocol. These protocols can encode e.g. an address (which device should respond) and a command: what it should do. The values for these are not always consistent across different devices for a given protocol.

Therefore the output of the IR decoder is a scancode; a single u32 value. Using keymap tables this can be mapped to linux key codes.

Other things can be encoded too. Some IR protocols encode a toggle bit; this is to distinguish whether the same button is being held down, or has been released and pressed again. If has been released and pressed again, the toggle bit will invert from one IR message to the next.

Some remotes have a pointer-type device which can be used to control the mouse; some air conditioning systems can have their target temperature target set in IR.

The following are the protocols the kernel knows about and also lists how scancodes are encoded for each protocol.

3.4.3.1 rc-5 (RC_PROTO_RC5)

This IR protocol uses manchester encoding to encode 14 bits. There is a detailed description here <https://www.sbprojects.net/knowledge/ir/rc5.php>.

The scancode encoding is *not* consistent with the lirc daemon (lircd) rc5 protocol, or the manchester BPF decoder.

Table 252: rc5 bits scancode mapping

rc-5 bit	scancode bit	description
1	none	Start bit, always set
1	6 (inverted)	2nd start bit in rc5, re-used as 6th command bit
1	none	Toggle bit
5	8 to 13	Address
6	0 to 5	Command

There is a variant of rc5 called either rc5x or extended rc5 where the second start bit is the 6th command bit, but inverted. This is done so the scancodes and encoding is compatible with existing schemes. This bit is stored in bit 6 of the scancode, inverted. This is done to keep it compatible with plain rc-5 where there are two start bits.

3.4.3.2 rc-5-sz (RC_PROTO_RC5_SZ)

This is much like rc-5 but one bit longer. The scancode is encoded differently.

Table 253: rc-5-sz bits scancode mapping

rc-5-sz bits	scancode bit	description
1	none	Start bit, always set
1	13	Address bit
1	none	Toggle bit
6	6 to 11	Address
6	0 to 5	Command

3.4.3.3 rc-5x-20 (RC_PROTO_RC5X_20)

This rc-5 extended to encoded 20 bits. There is a 3555 microseconds space after the 8th bit.

Table 254: rc-5x-20 bits scancode mapping

rc-5-sz bits	scancode bit	description
1	none	Start bit, always set
1	14	Address bit
1	none	Toggle bit
5	16 to 20	Address
6	8 to 13	Address
6	0 to 5	Command

3.4.3.4 jvc (RC_PROTO_JVC)

The jvc protocol is much like nec, without the inverted values. It is described here <https://www.sbprojects.net/knowledge/ir/jvc.php>.

The scancode is a 16 bits value, where the address is the lower 8 bits and the command the higher 8 bits; this is reversed from IR order.

3.4.3.5 sony-12 (RC_PROTO_SONY12)

The sony protocol is a pulse-width encoding. There are three variants, which just differ in number of bits and scancode encoding.

Table 255: sony-12 bits scancode mapping

sony-12 bits	scancode bit	description
5	16 to 20	device
7	0 to 6	function

3.4.3.6 sony-15 (RC_PROTO_SONY15)

The sony protocol is a pulse-width encoding. There are three variants, which just differ in number of bits and scancode encoding.

Table 256: sony-12 bits scancode mapping

sony-12 bits	scancode bit	description
8	16 to 23	device
7	0 to 6	function

3.4.3.7 sony-20 (RC_PROTO_SONY20)

The sony protocol is a pulse-width encoding. There are three variants, which just differ in number of bits and scancode encoding.

Table 257: sony-20 bits scancode mapping

sony-20 bits	scancode bit	description
5	16 to 20	device
7	0 to 7	device
8	8 to 15	extended bits

3.4.3.8 nec (RC_PROTO_NEC)

The nec protocol encodes an 8 bit address and an 8 bit command. It is described here <https://www.sbprojects.net/knowledge/ir/nec.php>. Note that the protocol sends least significant bit first.

As a check, the nec protocol sends the address and command twice; the second time it is inverted. This is done for verification.

A plain nec IR message has 16 bits; the high 8 bits are the address and the low 8 bits are the command.

3.4.3.9 nec-x (RC_PROTO_NECX)

Extended nec has a 16 bit address and a 8 bit command. This is encoded as a 24 bit value as you would expect, with the lower 8 bits the command and the upper 16 bits the address.

3.4.3.10 nec-32 (RC_PROTO_NEC32)

nec-32 does not send an inverted address or an inverted command; the entire message, all 32 bits, are used.

For this to be decoded correctly, the second 8 bits must not be the inverted value of the first, and also the last 8 bits must not be the inverted value of the third 8 bit value.

The scancode has a somewhat unusual encoding.

Table 258: nec-32 bits scancode mapping

nec-32 bits	scancode bit
First 8 bits	16 to 23
Second 8 bits	24 to 31
Third 8 bits	0 to 7
Fourth 8 bits	8 to 15

3.4.3.11 sanyo (RC_PROTO_SANYO)

The sanyo protocol is like the nec protocol, but with 13 bits address rather than 8 bits. Both the address and the command are followed by their inverted versions, but these are not present in the scancodes.

Bits 8 to 20 of the scancode is the 13 bits address, and the lower 8 bits are the command.

3.4.3.12 mcir2-kbd (RC_PROTO_MCIR2_KBD)

This protocol is generated by the Microsoft MCE keyboard for keyboard events. Refer to the `ir-mce_kbd-decoder.c` to see how it is encoded.

3.4.3.13 mcir2-mse (RC_PROTO_MCIR2_MSE)

This protocol is generated by the Microsoft MCE keyboard for pointer events. Refer to the `ir-mce_kbd-decoder.c` to see how it is encoded.

3.4.3.14 rc-6-0 (RC_PROTO_RC6_0)

This is the rc-6 in mode 0. rc-6 is described here <https://www.sbprojects.net/knowledge/ir/rc6.php>. The scancode is the exact 16 bits as in the protocol. There is also a toggle bit.

3.4.3.15 rc-6-6a-20 (RC_PROTO_RC6_6A_20)

This is the rc-6 in mode 6a, 20 bits. rc-6 is described here <https://www.sbprojects.net/knowledge/ir/rc6.php>. The scancode is the exact 20 bits as in the protocol. There is also a toggle bit.

3.4.3.16 rc-6-6a-24 (RC_PROTO_RC6_6A_24)

This is the rc-6 in mode 6a, 24 bits. rc-6 is described here <https://www.sbprojects.net/knowledge/ir/rc6.php>. The scancode is the exact 24 bits as in the protocol. There is also a toggle bit.

3.4.3.17 rc-6-6a-32 (RC_PROTO_RC6_6A_32)

This is the rc-6 in mode 6a, 32 bits. rc-6 is described here <https://www.sbprojects.net/knowledge/ir/rc6.php>. The upper 16 bits are the vendor, and the lower 16 bits are the vendor-specific bits. This protocol is for the non-Microsoft MCE variant (vendor != 0x800f).

3.4.3.18 rc-6-mce (RC_PROTO_RC6_MCE)

This is the rc-6 in mode 6a, 32 bits. The upper 16 bits are the vendor, and the lower 16 bits are the vendor-specific bits. This protocol is for the Microsoft MCE variant (vendor = 0x800f). The toggle bit in the protocol itself is ignored, and the 16th bit should be taken as the toggle bit.

3.4.3.19 sharp (RC_PROTO_SHARP)

This is a protocol used by Sharp VCRs, is described here <https://www.sbprojects.net/knowledge/ir/sharp.php>. There is a very long (40ms) space between the normal and inverted values, and some IR receivers cannot decode this.

There is a 5 bit address and a 8 bit command. In the scancode the address is in bits 8 to 12, and the command in bits 0 to 7.

3.4.3.20 xmp (RC_PROTO_XMP)

This protocol has several versions and only version 1 is supported. Refer to the decoder (ir-xmp-decoder.c) to see how it is encoded.

3.4.3.21 cec (RC_PROTO_CEC)

This is not an IR protocol, this is a protocol over CEC. The CEC infrastructure uses rc-core for handling CEC commands, so that they can easily be remapped.

3.4.3.22 imon (RC_PROTO_IMON)

This protocol is used by Antec Veris/SoundGraph iMON remotes.

The protocol describes both button presses and pointer movements. The protocol encodes 31 bits, and the scancode is simply the 31 bits with the top bit always 0.

3.4.3.23 rc-mm-12 (RC_PROTO_RCMM12)

The rc-mm protocol is described here <https://www.sbprojects.net/knowledge/ir/rcmm.php>. The scancode is simply the 12 bits.

3.4.3.24 rc-mm-24 (RC_PROTO_RCMM24)

The rc-mm protocol is described here <https://www.sbprojects.net/knowledge/ir/rcmm.php>. The scancode is simply the 24 bits.

3.4.3.25 rc-mm-32 (RC_PROTO_RCMM32)

The rc-mm protocol is described here <https://www.sbprojects.net/knowledge/ir/rcmm.php>. The scancode is simply the 32 bits.

3.4.3.26 xbox-dvd (RC_PROTO_XBOX_DVD)

This protocol is used by XBox DVD Remote, which was made for the original XBox. There is no in-kernel decoder or encoder for this protocol. The usb device decodes the protocol. There is a BPF decoder available in v4l-utils.

3.4.4 Remote controller tables

Unfortunately, for several years, there was no effort to create uniform IR keycodes for different devices. This caused the same IR keyname to be mapped completely differently on different IR devices. This resulted that the same IR keyname to be mapped completely different on different IR's. Due to that, V4L2 API now specifies a standard for mapping Media keys on IR.

This standard should be used by both V4L/DVB drivers and userspace applications

The modules register the remote as keyboard within the linux input layer. This means that the IR key strokes will look like normal keyboard key strokes (if CONFIG_INPUT_KEYBOARD is enabled). Using the event devices (CONFIG_INPUT_EVDEV) it is possible for applications to access the remote via /dev/input/event devices.

Table 259: IR default keymapping

Key code	Meaning	Key examples on IR
Numeric keys		
KEY_NUMERIC_0	Keyboard digit 0	0
KEY_NUMERIC_1	Keyboard digit 1	1
KEY_NUMERIC_2	Keyboard digit 2	2
KEY_NUMERIC_3	Keyboard digit 3	3
KEY_NUMERIC_4	Keyboard digit 4	4
KEY_NUMERIC_5	Keyboard digit 5	5
KEY_NUMERIC_6	Keyboard digit 6	6
KEY_NUMERIC_7	Keyboard digit 7	7
KEY_NUMERIC_8	Keyboard digit 8	8
KEY_NUMERIC_9	Keyboard digit 9	9
Movie play control		
KEY_FORWARD	Instantly advance in time	>> / FORWARD
KEY_BACK	Instantly go back in time	<<< / BACK
KEY_FASTFORWARD	Play movie faster	>>> / FORWARD
KEY_REWIND	Play movie back	REWIND / BACKWARD
KEY_NEXT	Select next chapter / sub-chapter / interval	NEXT / SKIP

Continued on next page

Table 259 - continued from previous page

KEY_PREVIOUS	Select previous chapter / sub-chapter / interval	<< / PREV / PREVIOUS
KEY_AGAIN	Repeat the video or a video interval	REPEAT / LOOP / RECALL
KEY_PAUSE	Pause stream	PAUSE / FREEZE
KEY_PLAY	Play movie at the normal timeshift	NORMAL TIMESHIFT / LIVE / >
KEY_PLAYPAUSE	Alternate between play and pause	PLAY / PAUSE
KEY_STOP	Stop stream	STOP
KEY_RECORD	Start/stop recording stream	CAPTURE / REC / RECORD/PAUSE
KEY_CAMERA	Take a picture of the image	CAMERA ICON / CAPTURE / SNAPSHOT
KEY_SHUFFLE	Enable shuffle mode	SHUFFLE
KEY_TIME	Activate time shift mode	TIME SHIFT
KEY_TITLE	Allow changing the chapter	CHAPTER
KEY_SUBTITLE	Allow changing the subtitle	SUBTITLE
Image control		
KEY_BRIGHTNESSDOWN	Decrease Brightness	BRIGHTNESS DECREASE
KEY_BRIGHTNESSUP	Increase Brightness	BRIGHTNESS INCREASE
KEY_ANGLE	Switch video camera angle (on videos with more than one angle stored)	ANGLE / SWAP
KEY_EPG	Open the Elecrownic Play Guide (EPG)	EPG / GUIDE
KEY_TEXT	Activate/change closed caption mode	CLOSED CAPTION/TELETEXT / DVD TEXT / TELETEXT / TTX
Audio control		
KEY_AUDIO	Change audio source	AUDIO SOURCE / AUDIO / MUSIC
KEY_MUTE	Mute/unmute audio	MUTE / DEMUTE / UNMUTE
KEY_VOLUMEDOWN	Decrease volume	VOLUME- / VOLUME DOWN
KEY_VOLUMEUP	Increase volume	VOLUME+ / VOLUME UP
KEY_MODE	Change sound mode	MONO/STEREO
KEY_LANGUAGE	Select Language	1ST / 2ND LANGUAGE / DVD LANG / MTS/SAP / MTS SEL
Channel control		
KEY_CHANNEL	Go to the next favorite channel	ALT / CHANNEL / CH SURFING / SURF / FAV
KEY_CHANNELDOWN	Decrease channel sequentially	CHANNEL - / CHANNEL DOWN / DOWN
KEY_CHANNELUP	Increase channel sequentially	CHANNEL + / CHANNEL UP / UP

Continued on next page

Table 259 - continued from previous page

KEY_DIGITS	Use more than one digit for channel	PLUS / 100/ 1xx / xxx / -/- / Single Double Triple Digit
KEY_SEARCH	Start channel autoscanner	SCAN / AUTOSCAN
Colored keys		
KEY_BLUE	IR Blue key	BLUE
KEY_GREEN	IR Green Key	GREEN
KEY_RED	IR Red key	RED
KEY_YELLOW	IR Yellow key	YELLOW
Media selection		
KEY_CD	Change input source to Compact Disc	CD
KEY_DVD	Change input to DVD	DVD / DVD MENU
KEY_EJECTCLOSECD	Open/close the CD/DVD player	->) / CLOSE / OPEN
KEY_MEDIA	Turn on/off Media application	PC/TV / TURN ON/OFF APP
KEY_PC	Selects from TV to PC	PC
KEY_RADIO	Put into AM/FM radio mode	RADIO / TV/FM / TV/RADIO / FM / FM/RADIO
KEY_TV	Select tv mode	TV / LIVE TV
KEY_TV2	Select Cable mode	AIR/CBL
KEY_VCR	Select VCR mode	VCR MODE / DTR
KEY_VIDEO	Alternate between input modes	SOURCE / SELECT / DISPLAY / SWITCH INPUTS / VIDEO
Power control		
KEY_POWER	Turn on/off computer	SYSTEM POWER / COMPUTER POWER
KEY_POWER2	Turn on/off application	TV ON/OFF / POWER
KEY_SLEEP	Activate sleep timer	SLEEP / SLEEP TIMER
KEY_SUSPEND	Put computer into suspend mode	STANDBY / SUSPEND
Window control		
KEY_CLEAR	Stop stream and return to default input video/audio	CLEAR / RESET / BOSS KEY
KEY_CYCLEWINDOWS	Minimize windows and move to the next one	ALT-TAB / MINIMIZE / DESKTOP
KEY_FAVORITES	Open the favorites stream window	TV WALL / Favorites
KEY_MENU	Call application menu	2ND CONTROLS (USA: MENU) / DVD/MENU / SHOW/HIDE CTRL
KEY_NEW	Open/Close Picture in Picture	PIP
KEY_OK	Send a confirmation code to application	OK / ENTER / RETURN
KEY_ASPECT_RATIO	Select screen aspect ratio	4:3 16:9 SELECT

Continued on next page

Table 259 - continued from previous page

KEY_FULL_SCREEN	Put device into zoom/full screen mode	ZOOM / FULL SCREEN / ZOOM+ / HIDE PANNEL / SWITCH
Navigation keys		
KEY_ESC	Cancel current operation	CANCEL / BACK
KEY_HELP	Open a Help window	HELP
KEY_HOMEPAGE	Navigate to Homepage	HOME
KEY_INFO	Open On Screen Display	DISPLAY INFORMATION / OSD
KEY_WWW	Open the default browser	WEB
KEY_UP	Up key	UP
KEY_DOWN	Down key	DOWN
KEY_LEFT	Left key	LEFT
KEY_RIGHT	Right key	RIGHT
Miscellaneous keys		
KEY_DOT	Return a dot	.
KEY_FN	Select a function	FUNCTION

It should be noted that, sometimes, there some fundamental missing keys at some cheaper IR's. Due to that, it is recommended to:

Table 260: Notes

On simpler IR's, without separate channel keys, you need to map UP as KEY_CHANNELUP
On simpler IR's, without separate channel keys, you need to map DOWN as KEY_CHANNELDOWN
On simpler IR's, without separate volume keys, you need to map LEFT as KEY_VOLUMEDOWN
On simpler IR's, without separate volume keys, you need to map RIGHT as KEY_VOLUMEUP

3.4.5 Changing default Remote Controller mappings

The event interface provides two ioctls to be used against the /dev/input/event device, to allow changing the default keymapping.

This program demonstrates how to replace the keymap tables.

3.4.5.1 file: uapi/v4l/keytable.c

```
/* keytable.c - This program allows checking/replacing keys at IR

Copyright (C) 2006-2009 Mauro Carvalho Chehab <mchehab@kernel.org>

This program is free software; you can redistribute it and/or modify
it under the terms of the GNU General Public License as published by
the Free Software Foundation, version 2 of the License.

This program is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
```

```

    GNU General Public License for more details.
*/

#include <ctype.h>
#include <errno.h>
#include <fcntl.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <linux/input.h>
#include <sys/ioctl.h>

#include "parse.h"

void prtcode (int *codes)
{
    struct parse_key *p;

    for (p=keynames;p->name!=NULL;p++) {
        if (p->value == (unsigned)codes[1]) {
            printf("scancode 0x%04x = %s (0x%02x)\n", codes[0], p->name,
↳ codes[1]);
            return;
        }
    }

    if (isprint (codes[1]))
        printf("scancode %d = '%c' (0x%02x)\n", codes[0], codes[1], codes[1]);
    else
        printf("scancode %d = 0x%02x\n", codes[0], codes[1]);
}

int parse_code(char *string)
{
    struct parse_key *p;

    for (p=keynames;p->name!=NULL;p++) {
        if (!strcasecmp(p->name, string)) {
            return p->value;
        }
    }
    return -1;
}

int main (int argc, char *argv[])
{
    int fd;
    unsigned int i, j;
    int codes[2];

    if (argc<2 || argc>4) {
        printf ("usage: %s <device> to get table; or\n"
               "      %s <device> <scancode> <keycode>\n"
               "      %s <device> <keycode_file>n", *argv, *argv, *argv);
        return -1;
    }
}

```

```
if ((fd = open(argv[1], O_RDONLY)) < 0) {
    perror("Couldn't open input device");
    return(-1);
}

if (argc==4) {
    int value;

    value=parse_code(argv[3]);

    if (value==-1) {
        value = strtol(argv[3], NULL, 0);
        if (errno)
            perror("value");
    }

    codes [0] = (unsigned) strtol(argv[2], NULL, 0);
    codes [1] = (unsigned) value;

    if(ioctl(fd, EVIOCSKEYCODE, codes))
        perror ("EVIOCSKEYCODE");

    if(ioctl(fd, EVIOCGKEYCODE, codes)==0)
        prtcode(codes);
    return 0;
}

if (argc==3) {
    FILE *fin;
    int value;
    char *scancode, *keycode, s[2048];

    fin=fopen(argv[2],"r");
    if (fin==NULL) {
        perror ("opening keycode file");
        return -1;
    }

    /* Clears old table */
    for (j = 0; j < 256; j++) {
        for (i = 0; i < 256; i++) {
            codes[0] = (j << 8) | i;
            codes[1] = KEY_RESERVED;
            ioctl(fd, EVIOCSKEYCODE, codes);
        }
    }

    while (fgets(s, sizeof(s), fin)) {
        scancode=strtok(s, "\\n\\t =:");
        if (!scancode) {
            perror ("parsing input file scancode");
            return -1;
        }
        if (!strcasecmp(scancode, "scancode")) {
            scancode = strtok(NULL, "\\n\\t =:");
            if (!scancode) {
                perror ("parsing input file scancode");
            }
        }
    }
}
```

```

        return -1;
    }
}

keycode=strtok(NULL, "\\n\\t =:");
if (!keycode) {
    perror ("parsing input file keycode");
    return -1;
}

// printf ("parsing %s=%s:", scancode, keycode);
value=parse_code(keycode);
// printf ("\\tvalue=%d\\n", value);

if (value==-1) {
    value = strtol(keycode, NULL, 0);
    if (errno)
        perror("value");
}

codes [0] = (unsigned) strtol(scancode, NULL, 0);
codes [1] = (unsigned) value;

// printf("\\t%04x=%04x\\n", codes[0], codes[1]);
if(ioctl(fd, EVIOCSKEYCODE, codes)) {
    fprintf(stderr, "Setting scancode 0x%04x with 0x%04x
→via ", codes[0], codes[1]);
    perror ("EVIOCSKEYCODE");
}

if(ioctl(fd, EVIOCGKEYCODE, codes)==0)
    prtcode(codes);
}
return 0;
}

/* Get scancode table */
for (j = 0; j < 256; j++) {
    for (i = 0; i < 256; i++) {
        codes[0] = (j << 8) | i;
        if (!ioctl(fd, EVIOCGKEYCODE, codes) && codes[1] != KEY_
→RESERVED)
            prtcode(codes);
    }
}
return 0;
}
}

```

3.4.6 LIRC Device Interface

3.4.6.1 Introduction

LIRC stands for Linux Infrared Remote Control. The LIRC device interface is a bi-directional interface for transporting raw IR and decoded scancodes data between userspace and kernelspace. Fundamentally, it is just a chardev (`/dev/lircX`, for $X = 0, 1, 2, \dots$), with a number of standard struct file_operations defined on it. With respect to transporting raw IR and decoded scancodes to and fro, the essential fops are read, write and ioctl.

It is also possible to attach a BPF program to a LIRC device for decoding raw IR into scancodes.

Example dmesg output upon a driver registering w/LIRC:

```
$ dmesg |grep lirc_dev
rc rc0: lirc_dev: driver mceusb registered at minor = 0, raw IR receiver, raw IR_
↳transmitter
```

What you should see for a chardev:

```
$ ls -l /dev/lirc*
crw-rw---- 1 root root 248, 0 Jul 2 22:20 /dev/lirc0
```

Note that the package `v4l-utils` contains tools for working with LIRC devices:

- `ir-ctl`: can receive raw IR and transmit IR, as well as query LIRC device features.
- `ir-keytable`: can load keymaps; allows you to set IR kernel protocols; load BPF IR decoders and test IR decoding. Some BPF IR decoders are also provided.

3.4.6.2 LIRC modes

LIRC supports some modes of receiving and sending IR codes, as shown on the following table.

LIRC_MODE_SCANCODE

This mode is for both sending and receiving IR.

For transmitting (aka sending), create a struct `lirc_scancode` with the desired scancode set in the `scancode` member, `rc_proto` set to the *IR protocol*, and all other members set to 0. Write this struct to the lirc device.

For receiving, you read struct `lirc_scancode` from the LIRC device. The `scancode` field is set to the received scancode and the *IR protocol* is set in `rc_proto`. If the scancode maps to a valid key code, this is set in the `keycode` field, else it is set to `KEY_RESERVED`.

The `flags` can have `LIRC_SCANCODE_FLAG_TOGGLE` set if the toggle bit is set in protocols that support it (e.g. `rc-5` and `rc-6`), or `LIRC_SCANCODE_FLAG_REPEAT` for when a repeat is received for protocols that support it (e.g. `nec`).

In the Sanyo and NEC protocol, if you hold a button on remote, rather than repeating the entire scancode, the remote sends a shorter message with no scancode, which just means button is held, a “repeat”. When this is received, the `LIRC_SCANCODE_FLAG_REPEAT` is set and the scancode and keycode is repeated.

With `nec`, there is no way to distinguish “button hold” from “repeatedly pressing the same button”. The `rc-5` and `rc-6` protocols have a toggle bit. When a button

is released and pressed again, the toggle bit is inverted. If the toggle bit is set, the `LIRC_SCANCODE_FLAG_TOGGLE` is set.

The timestamp field is filled with the time nanoseconds (in `CLOCK_MONOTONIC`) when the scancode was decoded.

LIRC_MODE_MODE2

The driver returns a sequence of pulse and space codes to userspace, as a series of u32 values.

This mode is used only for IR receive.

The upper 8 bits determine the packet type, and the lower 24 bits the payload. Use `LIRC_VALUE()` macro to get the payload, and the macro `LIRC_MODE2()` will give you the type, which is one of:

LIRC_MODE2_PULSE

Signifies the presence of IR in microseconds.

LIRC_MODE2_SPACE

Signifies absence of IR in microseconds.

LIRC_MODE2_FREQUENCY

If measurement of the carrier frequency was enabled with *ioctl* `LIRC_SET_MEASURE_CARRIER_MODE` then this packet gives you the carrier frequency in Hertz.

LIRC_MODE2_TIMEOUT

If timeout reports are enabled with *ioctl* `LIRC_SET_REC_TIMEOUT_REPORTS`, when the timeout set with *ioctl* `LIRC_GET_REC_TIMEOUT` and `LIRC_SET_REC_TIMEOUT` expires due to no IR being detected, this packet will be sent, with the number of microseconds with no IR.

LIRC_MODE_PULSE

In pulse mode, a sequence of pulse/space integer values are written to the lirc device using *LIRC write()*.

The values are alternating pulse and space lengths, in microseconds. The first and last entry must be a pulse, so there must be an odd number of entries.

This mode is used only for IR send.

3.4.6.3 Data types used by LIRC_MODE_SCANCODE

struct `lirc_scancode`

decoded scancode with protocol for use with `LIRC_MODE_SCANCODE`

Definition

```
struct lirc_scancode {
    __u64 timestamp;
    __u16 flags;
    __u16 rc_proto;
    __u32 keycode;
```

```
    __u64 scancode;
};
```

Members

timestamp Timestamp in nanoseconds using CLOCK_MONOTONIC when IR was decoded.

flags should be 0 for transmit. When receiving scancodes, LIRC_SCANCODE_FLAG_TOGGLE or LIRC_SCANCODE_FLAG_REPEAT can be set depending on the protocol

rc_proto see enum rc_proto

keycode the translated keycode. Set to 0 for transmit.

scancode the scancode received or to be sent

enum **rc_proto**

the Remote Controller protocol

Constants

RC_PROTO_UNKNOWN Protocol not known

RC_PROTO_OTHER Protocol known but proprietary

RC_PROTO_RC5 Philips RC5 protocol

RC_PROTO_RC5X_20 Philips RC5x 20 bit protocol

RC_PROTO_RC5_SZ StreamZap variant of RC5

RC_PROTO_JVC JVC protocol

RC_PROTO_SONY12 Sony 12 bit protocol

RC_PROTO_SONY15 Sony 15 bit protocol

RC_PROTO_SONY20 Sony 20 bit protocol

RC_PROTO_NEC NEC protocol

RC_PROTO_NECX Extended NEC protocol

RC_PROTO_NEC32 NEC 32 bit protocol

RC_PROTO_SANYO Sanyo protocol

RC_PROTO_MCIR2_KBD RC6-ish MCE keyboard

RC_PROTO_MCIR2_MSE RC6-ish MCE mouse

RC_PROTO_RC6_0 Philips RC6-0-16 protocol

RC_PROTO_RC6_6A_20 Philips RC6-6A-20 protocol

RC_PROTO_RC6_6A_24 Philips RC6-6A-24 protocol

RC_PROTO_RC6_6A_32 Philips RC6-6A-32 protocol

RC_PROTO_RC6_MCE MCE (Philips RC6-6A-32 subtype) protocol

RC_PROTO_SHARP Sharp protocol

RC_PROTO_XMP XMP protocol

RC_PROTO_CEC CEC protocol

RC_PROTO_IMON iMon Pad protocol
RC_PROTO_RCMM12 RC-MM protocol 12 bits
RC_PROTO_RCMM24 RC-MM protocol 24 bits
RC_PROTO_RCMM32 RC-MM protocol 32 bits
RC_PROTO_XBOX_DVD Xbox DVD Movie Playback Kit protocol
RC_PROTO_MAX Maximum value of enum rc_proto

3.4.6.4 BPF based IR decoder

The kernel has support for decoding the most common *IR protocols*, but there are many protocols which are not supported. To support these, it is possible to load an BPF program which does the decoding. This can only be done on LIRC devices which support reading raw IR.

First, using the `bpf(2)` syscall with the `BPF_LOAD_PROG` argument, program must be loaded of type `BPF_PROG_TYPE_LIRC_MODE2`. Once attached to the LIRC device, this program will be called for each pulse, space or timeout event on the LIRC device. The context for the BPF program is a pointer to a unsigned int, which is a `LIRC_MODE_MODE2` value. When the program has decoded the scancode, it can be submitted using the BPF functions `bpf_rc_keydown()` or `bpf_rc_repeat()`. Mouse or pointer movements can be reported using `bpf_rc_pointer_rel()`.

Once you have the file descriptor for the `BPF_PROG_TYPE_LIRC_MODE2` BPF program, it can be attached to the LIRC device using the `bpf(2)` syscall. The target must be the file descriptor for the LIRC device, and the attach type must be `BPF_LIRC_MODE2`. No more than 64 BPF programs can be attached to a single LIRC device at a time.

3.4.6.5 LIRC Function Reference

LIRC read()

Name

lirc-read - Read from a LIRC device

Synopsis

```
#include <unistd.h>
```

```
ssize_t read(int fd, void *buf, size_t count)
```

Arguments

fd File descriptor returned by `open()`.

buf Buffer to be filled

count Max number of bytes to read

Description

`read()` attempts to read up to `count` bytes from file descriptor `fd` into the buffer starting at `buf`. If `count` is zero, `read()` returns zero and has no other results. If `count` is greater than `SSIZE_MAX`, the result is unspecified.

The exact format of the data depends on what *LIRC modes* a driver uses. Use *ioctl* `LIRC_GET_FEATURES` to get the supported mode, and use *ioctl*s `LIRC_GET_REC_MODE` and `LIRC_SET_REC_MODE` set the current active mode.

The mode `LIRC_MODE_MODE2` is for raw IR, in which packets containing an unsigned int value describing an IR signal are read from the chardev.

Alternatively, `LIRC_MODE_SCANCODE` can be available, in this mode scancodes which are either decoded by software decoders, or by hardware decoders. The `rc_proto` member is set to the *IR protocol* used for transmission, and `scancode` to the decoded scancode, and the `keycode` set to the keycode or `KEY_RESERVED`.

Return Value

On success, the number of bytes read is returned. It is not an error if this number is smaller than the number of bytes requested, or the amount of data required for one frame. On error, `-1` is returned, and the `errno` variable is set appropriately.

LIRC write()

Name

`lirc-write` - Write to a LIRC device

Synopsis

```
#include <unistd.h>
```

```
ssize_t write(int fd, void *buf, size_t count)
```

Arguments

fd File descriptor returned by `open()`.

buf Buffer with data to be written

count Number of bytes at the buffer

Description

`write()` writes up to `count` bytes to the device referenced by the file descriptor `fd` from the buffer starting at `buf`.

The exact format of the data depends on what mode a driver is in, use *ioctl* `LIRC_GET_FEATURES` to get the supported modes and use *ioctls* `LIRC_GET_SEND_MODE` and `LIRC_SET_SEND_MODE` set the mode.

When in `LIRC_MODE_PULSE` mode, the data written to the chardev is a pulse/space sequence of integer values. Pulses and spaces are only marked implicitly by their position. The data must start and end with a pulse, therefore, the data must always include an uneven number of samples. The write function blocks until the data has been transmitted by the hardware. If more data is provided than the hardware can send, the driver returns `EINVAL`.

When in `LIRC_MODE_SCANCODE` mode, one `struct lirc_scancode` must be written to the chardev at a time, else `EINVAL` is returned. Set the desired scancode in the `scancode` member, and the *IR protocol* in the `rc_proto:` member. All other members must be set to 0, else `EINVAL` is returned. If there is no protocol encoder for the protocol or the scancode is not valid for the specified protocol, `EINVAL` is returned. The write function blocks until the scancode is transmitted by the hardware.

Return Value

On success, the number of bytes written is returned. It is not an error if this number is smaller than the number of bytes requested, or the amount of data required for one frame. On error, -1 is returned, and the `errno` variable is set appropriately. The generic error codes are described at the *Generic Error Codes* chapter.

ioctl LIRC_GET_FEATURES

Name

`LIRC_GET_FEATURES` - Get the underlying hardware device's features

Synopsis

LIRC_GET_FEATURES

```
int ioctl(int fd, LIRC_GET_FEATURES, __u32 *features)
```

Arguments

fd File descriptor returned by `open()`.

features Bitmask with the LIRC features.

Description

Get the underlying hardware device's features. If a driver does not announce support of certain features, calling of the corresponding `ioctls` is undefined.

LIRC features

LIRC_CAN_REC_RAW

Unused. Kept just to avoid breaking uAPI.

LIRC_CAN_REC_PULSE

Unused. Kept just to avoid breaking uAPI. *LIRC_MODE_PULSE* can only be used for transmitting.

LIRC_CAN_REC_MODE2

This is raw IR driver for receiving. This means that *LIRC_MODE_MODE2* is used. This also implies that *LIRC_MODE_SCANCODE* is also supported, as long as the kernel is recent enough. Use the *ioctls* *LIRC_GET_REC_MODE* and *LIRC_SET_REC_MODE* to switch modes.

LIRC_CAN_REC_LIRCCODE

Unused. Kept just to avoid breaking uAPI.

LIRC_CAN_REC_SCANCODE

This is a scancode driver for receiving. This means that *LIRC_MODE_SCANCODE* is used.

LIRC_CAN_SET_SEND_CARRIER

The driver supports changing the modulation frequency via *ioctl* *LIRC_SET_SEND_CARRIER*.

LIRC_CAN_SET_SEND_DUTY_CYCLE

The driver supports changing the duty cycle using *ioctl* *LIRC_SET_SEND_DUTY_CYCLE*.

LIRC_CAN_SET_TRANSMITTER_MASK

The driver supports changing the active transmitter(s) using *ioctl* [LIRC_SET_TRANSMITTER_MASK](#).

LIRC_CAN_SET_REC_CARRIER

The driver supports setting the receive carrier frequency using *ioctl* [LIRC_SET_REC_CARRIER](#).

LIRC_CAN_SET_REC_DUTY_CYCLE_RANGE

Unused. Kept just to avoid breaking uAPI.

LIRC_CAN_SET_REC_CARRIER_RANGE

The driver supports *ioctl* [LIRC_SET_REC_CARRIER_RANGE](#).

LIRC_CAN_GET_REC_RESOLUTION

The driver supports *ioctl* [LIRC_GET_REC_RESOLUTION](#).

LIRC_CAN_SET_REC_TIMEOUT

The driver supports *ioctl* [LIRC_SET_REC_TIMEOUT](#).

LIRC_CAN_SET_REC_FILTER

Unused. Kept just to avoid breaking uAPI.

LIRC_CAN_MEASURE_CARRIER

The driver supports measuring of the modulation frequency using *ioctl* [LIRC_SET_MEASURE_CARRIER_MODE](#).

LIRC_CAN_USE_WIDEBAND_RECEIVER

The driver supports learning mode using *ioctl* [LIRC_SET_WIDEBAND_RECEIVER](#).

LIRC_CAN_NOTIFY_DECODE

Unused. Kept just to avoid breaking uAPI.

LIRC_CAN_SEND_RAW

Unused. Kept just to avoid breaking uAPI.

LIRC_CAN_SEND_PULSE

The driver supports sending (also called as IR blasting or IR TX) using [LIRC_MODE_PULSE](#). This implies that [LIRC_MODE_SCANCODE](#) is also supported for transmit, as long as the kernel is recent enough. Use the *ioctls* [LIRC_GET_SEND_MODE](#) and [LIRC_SET_SEND_MODE](#) to switch modes.

LIRC_CAN_SEND_MODE2

Unused. Kept just to avoid breaking uAPI. [LIRC_MODE_MODE2](#) can only be used for receiving.

LIRC_CAN_SEND_LIRCCODE

Unused. Kept just to avoid breaking uAPI.

Return Value

On success 0 is returned, on error -1 and the `errno` variable is set appropriately. The generic error codes are described at the [Generic Error Codes](#) chapter.

ioctl's LIRC_GET_SEND_MODE and LIRC_SET_SEND_MODE

Name

LIRC_GET_SEND_MODE/LIRC_SET_SEND_MODE - Get/set current transmit mode.

Synopsis

LIRC_GET_SEND_MODE

```
int ioctl(int fd, LIRC_GET_SEND_MODE, __u32 *mode)
```

LIRC_SET_SEND_MODE

```
int ioctl(int fd, LIRC_SET_SEND_MODE, __u32 *mode)
```

Arguments

fd File descriptor returned by `open()`.

mode The mode used for transmitting.

Description

Get/set current transmit mode.

Only [LIRC_MODE_PULSE](#) and [LIRC_MODE_SCANCODE](#) are supported by for IR send, depending on the driver. Use [ioctl LIRC_GET_FEATURES](#) to find out which modes the driver supports.

Return Value

ENODEV	Device not available.
ENOTTY	Device does not support transmitting.
EINVAL	Invalid mode or invalid mode for this device.

ioctl LIRC_GET_REC_MODE and LIRC_SET_REC_MODE

Name

LIRC_GET_REC_MODE/LIRC_SET_REC_MODE - Get/set current receive mode.

Synopsis

LIRC_GET_REC_MODE

```
int ioctl(int fd, LIRC_GET_REC_MODE, __u32 *mode)
```

LIRC_SET_REC_MODE

```
int ioctl(int fd, LIRC_SET_REC_MODE, __u32 *mode)
```

Arguments

fd File descriptor returned by open().

mode Mode used for receive.

Description

Get and set the current receive mode. Only *LIRC_MODE_MODE2* and *LIRC_MODE_SCANCODE* are supported. Use *ioctl LIRC_GET_FEATURES* to find out which modes the driver supports.

Return Value

ENODEV	Device not available.
ENOTTY	Device does not support receiving.
EINVAL	Invalid mode or invalid mode for this device.

ioctl LIRC_GET_REC_RESOLUTION

Name

LIRC_GET_REC_RESOLUTION - Obtain the value of receive resolution, in microseconds.

Synopsis

LIRC_GET_REC_RESOLUTION

```
int ioctl(int fd, LIRC_GET_REC_RESOLUTION, __u32 *microseconds)
```

Arguments

fd File descriptor returned by `open()`.

microseconds Resolution, in microseconds.

Description

Some receivers have maximum resolution which is defined by internal sample rate or data format limitations. E.g. it's common that signals can only be reported in 50 microsecond steps.

This `ioctl` returns the integer value with such resolution, with can be used by userspace applications like `lircd` to automatically adjust the tolerance value.

Return Value

On success 0 is returned, on error -1 and the `errno` variable is set appropriately. The generic error codes are described at the [Generic Error Codes](#) chapter.

ioctl LIRC_SET_SEND_DUTY_CYCLE

Name

LIRC_SET_SEND_DUTY_CYCLE - Set the duty cycle of the carrier signal for IR transmit.

Synopsis

LIRC_SET_SEND_DUTY_CYCLE

```
int ioctl(int fd, LIRC_SET_SEND_DUTY_CYCLE, __u32 *duty_cycle)
```

Arguments

fd File descriptor returned by `open()`.

duty_cycle Duty cycle, describing the pulse width in percent (from 1 to 99) of the total cycle. Values 0 and 100 are reserved.

Description

Get/set the duty cycle of the carrier signal for IR transmit.

Currently, no special meaning is defined for 0 or 100, but this could be used to switch off carrier generation in the future, so these values should be reserved.

Return Value

On success 0 is returned, on error -1 and the `errno` variable is set appropriately. The generic error codes are described at the *Generic Error Codes* chapter.

ioctls LIRC_GET_MIN_TIMEOUT and LIRC_GET_MAX_TIMEOUT

Name

LIRC_GET_MIN_TIMEOUT / LIRC_GET_MAX_TIMEOUT - Obtain the possible timeout range for IR receive.

Synopsis

LIRC_GET_MIN_TIMEOUT

```
int ioctl(int fd, LIRC_GET_MIN_TIMEOUT, __u32 *timeout)
```

LIRC_GET_MAX_TIMEOUT

```
int ioctl(int fd, LIRC_GET_MAX_TIMEOUT, __u32 *timeout)
```

Arguments

fd File descriptor returned by `open()`.

timeout Timeout, in microseconds.

Description

Some devices have internal timers that can be used to detect when there's no IR activity for a long time. This can help `lircd` in detecting that a IR signal is finished and can speed up the decoding process. Returns an integer value with the minimum/maximum timeout that can be set.

Note: Some devices have a fixed timeout, in that case both `ioctl`s will return the same value even though the timeout cannot be changed via `ioctl LIRC_GET_REC_TIMEOUT and LIRC_SET_REC_TIMEOUT`.

Return Value

On success 0 is returned, on error -1 and the `errno` variable is set appropriately. The generic error codes are described at the [Generic Error Codes](#) chapter.

ioctl LIRC_GET_REC_TIMEOUT and LIRC_SET_REC_TIMEOUT

Name

LIRC_GET_REC_TIMEOUT/LIRC_SET_REC_TIMEOUT - Get/set the integer value for IR inactivity timeout.

Synopsis

LIRC_GET_REC_TIMEOUT

```
int ioctl(int fd, LIRC_GET_REC_TIMEOUT, __u32 *timeout)
```

LIRC_SET_REC_TIMEOUT

```
int ioctl(int fd, LIRC_SET_REC_TIMEOUT, __u32 *timeout)
```

Arguments

fd File descriptor returned by `open()`.

timeout Timeout, in microseconds.

Description

Get and set the integer value for IR inactivity timeout.

If supported by the hardware, setting it to 0 disables all hardware timeouts and data should be reported as soon as possible. If the exact value cannot be set, then the next possible value `_greater_` than the given value should be set.

Note: The range of supported timeout is given by *ioctls* `LIRC_GET_MIN_TIMEOUT` and `LIRC_GET_MAX_TIMEOUT`.

Return Value

On success 0 is returned, on error -1 and the `errno` variable is set appropriately. The generic error codes are described at the [Generic Error Codes](#) chapter.

ioctl LIRC_SET_REC_CARRIER

Name

LIRC_SET_REC_CARRIER - Set carrier used to modulate IR receive.

Synopsis

LIRC_SET_REC_CARRIER

```
int ioctl(int fd, LIRC_SET_REC_CARRIER, __u32 *frequency)
```

Arguments

fd File descriptor returned by `open()`.

frequency Frequency of the carrier that modulates PWM data, in Hz.

Description

Set receive carrier used to modulate IR PWM pulses and spaces.

Note: If called together with [ioctl LIRC_SET_REC_CARRIER_RANGE](#), this `ioctl` sets the upper bound frequency that will be recognized by the device.

Return Value

On success 0 is returned, on error -1 and the `errno` variable is set appropriately. The generic error codes are described at the [Generic Error Codes](#) chapter.

ioctl LIRC_SET_REC_CARRIER_RANGE

Name

LIRC_SET_REC_CARRIER_RANGE - Set lower bound of the carrier used to modulate IR receive.

Synopsis

LIRC_SET_REC_CARRIER_RANGE

```
int ioctl(int fd, LIRC_SET_REC_CARRIER_RANGE, __u32 *frequency)
```

Arguments

fd File descriptor returned by open().

frequency Frequency of the carrier that modulates PWM data, in Hz.

Description

This ioctl sets the upper range of carrier frequency that will be recognized by the IR receiver.

Note: To set a range use *LIRC_SET_REC_CARRIER_RANGE* with the lower bound first and later call *LIRC_SET_REC_CARRIER* with the upper bound.

Return Value

On success 0 is returned, on error -1 and the `errno` variable is set appropriately. The generic error codes are described at the *Generic Error Codes* chapter.

ioctl LIRC_SET_SEND_CARRIER

Name

LIRC_SET_SEND_CARRIER - Set send carrier used to modulate IR TX.

Synopsis

LIRC_SET_SEND_CARRIER

```
int ioctl(int fd, LIRC_SET_SEND_CARRIER, __u32 *frequency)
```

Arguments

fd File descriptor returned by open().

frequency Frequency of the carrier to be modulated, in Hz.

Description

Set send carrier used to modulate IR PWM pulses and spaces.

Return Value

On success 0 is returned, on error -1 and the `errno` variable is set appropriately. The generic error codes are described at the *Generic Error Codes* chapter.

ioctl LIRC_SET_TRANSMITTER_MASK

Name

LIRC_SET_TRANSMITTER_MASK - Enables send codes on a given set of transmitters

Synopsis

LIRC_SET_TRANSMITTER_MASK

```
int ioctl(int fd, LIRC_SET_TRANSMITTER_MASK, __u32 *mask)
```

Arguments

fd File descriptor returned by `open()`.

mask Mask with channels to enable tx. Channel 0 is the least significant bit.

Description

Some IR TX devices have multiple output channels, in such case, *LIRC_CAN_SET_TRANSMITTER_MASK* is returned via *ioctl LIRC_GET_FEATURES* and this *ioctl* sets what channels will send IR codes.

This *ioctl* enables the given set of transmitters. The first transmitter is encoded by the least significant bit and so on.

When an invalid bit mask is given, i.e. a bit is set, even though the device does not have so many transmitters, then this *ioctl* returns the number of available transmitters and does nothing otherwise.

Return Value

On success 0 is returned, on error -1 and the `errno` variable is set appropriately. The generic error codes are described at the [Generic Error Codes](#) chapter.

ioctl LIRC_SET_REC_TIMEOUT_REPORTS

Name

LIRC_SET_REC_TIMEOUT_REPORTS - enable or disable timeout reports for IR receive

Synopsis

LIRC_SET_REC_TIMEOUT_REPORTS

```
int ioctl(int fd, LIRC_SET_REC_TIMEOUT_REPORTS, __u32 *enable)
```

Arguments

fd File descriptor returned by `open()`.

enable `enable = 1` means enable timeout report, `enable = 0` means disable timeout reports.

Description

Enable or disable timeout reports for IR receive. By default, timeout reports should be turned off.

Note: This `ioctl` is only valid for [LIRC_MODE_MODE2](#).

Return Value

On success 0 is returned, on error -1 and the `errno` variable is set appropriately. The generic error codes are described at the [Generic Error Codes](#) chapter.

ioctl LIRC_SET_MEASURE_CARRIER_MODE

Name

LIRC_SET_MEASURE_CARRIER_MODE - enable or disable measure mode

Synopsis

LIRC_SET_MEASURE_CARRIER_MODE

```
int ioctl(int fd, LIRC_SET_MEASURE_CARRIER_MODE, __u32 *enable)
```

Arguments

fd File descriptor returned by open().

enable enable = 1 means enable measure mode, enable = 0 means disable measure mode.

Description

Enable or disable measure mode. If enabled, from the next key press on, the driver will send LIRC_MODE2_FREQUENCY packets. By default this should be turned off.

Return Value

On success 0 is returned, on error -1 and the `errno` variable is set appropriately. The generic error codes are described at the [Generic Error Codes](#) chapter.

ioctl LIRC_SET_WIDEBAND_RECEIVER

Name

LIRC_SET_WIDEBAND_RECEIVER - enable wide band receiver.

Synopsis

LIRC_SET_WIDEBAND_RECEIVER

```
int ioctl(int fd, LIRC_SET_WIDEBAND_RECEIVER, __u32 *enable)
```

Arguments

fd File descriptor returned by open().

enable enable = 1 means enable wideband receiver, enable = 0 means disable wideband receiver.

Description

Some receivers are equipped with special wide band receiver which is intended to be used to learn output of existing remote. This ioctl allows enabling or disabling it.

This might be useful of receivers that have otherwise narrow band receiver that prevents them to be used with some remotes. Wide band receiver might also be more precise. On the other hand its disadvantage it usually reduced range of reception.

Note: Wide band receiver might be implicitly enabled if you enable carrier reports. In that case it will be disabled as soon as you disable carrier reports. Trying to disable wide band receiver while carrier reports are active will do nothing.

Return Value

On success 0 is returned, on error -1 and the `errno` variable is set appropriately. The generic error codes are described at the [Generic Error Codes](#) chapter.

3.4.6.6 LIRC Header File

`lirc.h`

```
/* SPDX-License-Identifier: GPL-2.0 WITH Linux-syscall-note */
/*
 * lirc.h - linux infrared remote control header file
 */

#ifndef _LINUX_LIRC_H
#define _LINUX_LIRC_H

#include <linux/types.h>
#include <linux/ioctl.h>

#define PULSE_BIT          0x01000000
#define PULSE_MASK        0x00FFFFFF

#define LIRC_MODE2_SPACE   0x00000000
#define LIRC_MODE2_PULSE  0x01000000
#define LIRC_MODE2_FREQUENCY 0x02000000
#define LIRC_MODE2_TIMEOUT 0x03000000

#define LIRC_VALUE_MASK   0x00FFFFFF
#define LIRC_MODE2_MASK   0xFF000000

#define LIRC_SPACE(val) (((val)&LIRC_VALUE_MASK) | LIRC_MODE2_SPACE)
#define LIRC_PULSE(val) (((val)&LIRC_VALUE_MASK) | LIRC_MODE2_PULSE)
#define LIRC_FREQUENCY(val) (((val)&LIRC_VALUE_MASK) | LIRC_MODE2_FREQUENCY)
#define LIRC_TIMEOUT(val) (((val)&LIRC_VALUE_MASK) | LIRC_MODE2_TIMEOUT)
```

```

#define LIRC_VALUE(val) ((val)&LIRC_VALUE_MASK)
#define LIRC_MODE2(val) ((val)&LIRC_MODE2_MASK)

#define LIRC_IS_SPACE(val) (LIRC_MODE2(val) == LIRC_MODE2_SPACE)
#define LIRC_IS_PULSE(val) (LIRC_MODE2(val) == LIRC_MODE2_PULSE)
#define LIRC_IS_FREQUENCY(val) (LIRC_MODE2(val) == LIRC_MODE2_FREQUENCY)
#define LIRC_IS_TIMEOUT(val) (LIRC_MODE2(val) == LIRC_MODE2_TIMEOUT)

/* used heavily by lirc userspace */
#define lirc_t int

/** lirc compatible hardware features */

#define LIRC_MODE2SEND(x) (x)
#define LIRC_SEND2MODE(x) (x)
#define LIRC_MODE2REC(x) ((x) << 16)
#define LIRC_REC2MODE(x) ((x) >> 16)

#define LIRC_MODE_RAW                0x00000001
#define LIRC_MODE_PULSE              0x00000002
#define LIRC_MODE_MODE2            0x00000004
#define LIRC_MODE_SCANCODE        0x00000008
#define LIRC_MODE_LIRCCODE           0x00000010

#define LIRC_CAN_SEND_RAW          LIRC_MODE2SEND(LIRC_MODE_RAW)
#define LIRC_CAN_SEND_PULSE       LIRC_MODE2SEND(LIRC_MODE_PULSE)
#define LIRC_CAN_SEND_MODE2      LIRC_MODE2SEND(LIRC_MODE_MODE2)
#define LIRC_CAN_SEND_LIRCCODE    LIRC_MODE2SEND(LIRC_MODE_LIRCCODE)

#define LIRC_CAN_SEND_MASK           0x0000003f

#define LIRC_CAN_SET_SEND_CARRIER 0x00000100
#define LIRC_CAN_SET_SEND_DUTY_CYCLE 0x00000200
#define LIRC_CAN_SET_TRANSMITTER_MASK 0x00000400

#define LIRC_CAN_REC_RAW          LIRC_MODE2REC(LIRC_MODE_RAW)
#define LIRC_CAN_REC_PULSE       LIRC_MODE2REC(LIRC_MODE_PULSE)
#define LIRC_CAN_REC_MODE2      LIRC_MODE2REC(LIRC_MODE_MODE2)
#define LIRC_CAN_REC_SCANCODE    LIRC_MODE2REC(LIRC_MODE_SCANCODE)
#define LIRC_CAN_REC_LIRCCODE    LIRC_MODE2REC(LIRC_MODE_LIRCCODE)

#define LIRC_CAN_REC_MASK            LIRC_MODE2REC(LIRC_CAN_SEND_MASK)

#define LIRC_CAN_SET_REC_CARRIER  (LIRC_CAN_SET_SEND_CARRIER << 16)
#define LIRC_CAN_SET_REC_DUTY_CYCLE (LIRC_CAN_SET_SEND_DUTY_CYCLE << 16)

#define LIRC_CAN_SET_REC_DUTY_CYCLE_RANGE 0x40000000
#define LIRC_CAN_SET_REC_CARRIER_RANGE    0x80000000
#define LIRC_CAN_GET_REC_RESOLUTION       0x20000000
#define LIRC_CAN_SET_REC_TIMEOUT          0x10000000

```

```
#define LIRC_CAN_SET_REC_FILTER          0x08000000

#define LIRC_CAN_MEASURE_CARRIER       0x02000000
#define LIRC_CAN_USE_WIDEBAND_RECEIVER  0x04000000

#define LIRC_CAN_SEND(x) ((x)&LIRC_CAN_SEND_MASK)
#define LIRC_CAN_REC(x) ((x)&LIRC_CAN_REC_MASK)

#define LIRC_CAN_NOTIFY_DECODE          0x01000000

/**** IOCTL commands for lirc driver ****/

#define LIRC_GET_FEATURES                _IOR('i', 0x00000000, __u32)

#define LIRC_GET_SEND_MODE               _IOR('i', 0x00000001, __u32)
#define LIRC_GET_REC_MODE                _IOR('i', 0x00000002, __u32)
#define LIRC_GET_REC_RESOLUTION          _IOR('i', 0x00000007, __u32)

#define LIRC_GET_MIN_TIMEOUT              _IOR('i', 0x00000008, __u32)
#define LIRC_GET_MAX_TIMEOUT              _IOR('i', 0x00000009, __u32)

/* code length in bits, currently only for LIRC_MODE_LIRCCODE */
#define LIRC_GET_LENGTH                   _IOR('i', 0x0000000f, __u32)

#define LIRC_SET_SEND_MODE                _IOW('i', 0x00000011, __u32)
#define LIRC_SET_REC_MODE                 _IOW('i', 0x00000012, __u32)
/* Note: these can reset the according pulse_width */
#define LIRC_SET_SEND_CARRIER            _IOW('i', 0x00000013, __u32)
#define LIRC_SET_REC_CARRIER             _IOW('i', 0x00000014, __u32)
#define LIRC_SET_SEND_DUTY_CYCLE          _IOW('i', 0x00000015, __u32)
#define LIRC_SET_TRANSMITTER_MASK        _IOW('i', 0x00000017, __u32)

/*
 * when a timeout != 0 is set the driver will send a
 * LIRC_MODE2_TIMEOUT data packet, otherwise LIRC_MODE2_TIMEOUT is
 * never sent, timeout is disabled by default
 */
#define LIRC_SET_REC_TIMEOUT               _IOW('i', 0x00000018, __u32)

/* 1 enables, 0 disables timeout reports in MODE2 */
#define LIRC_SET_REC_TIMEOUT_REPORTS      _IOW('i', 0x00000019, __u32)

/*
 * if enabled from the next key press on the driver will send
 * LIRC_MODE2_FREQUENCY packets
 */
#define LIRC_SET_MEASURE_CARRIER_MODE    _IOW('i', 0x0000001d, __u32)

/*
 * to set a range use LIRC_SET_REC_CARRIER_RANGE with the
 * lower bound first and later LIRC_SET_REC_CARRIER with the upper bound
 */
```

```

*/
#define LIRC_SET_REC_CARRIER_RANGE    _IOW('i', 0x0000001f, __u32)

#define LIRC_SET_WIDEBAND_RECEIVER    _IOW('i', 0x00000023, __u32)

/*
 * Return the recording timeout, which is either set by
 * the ioctl LIRC_SET_REC_TIMEOUT or by the kernel after setting the protocols.
 */
#define LIRC_GET_REC_TIMEOUT          _IOR('i', 0x00000024, __u32)

/**
 * struct lirc_scancode - decoded scancode with protocol for use with
 * LIRC_MODE_SCANCODE
 *
 * @timestamp: Timestamp in nanoseconds using CLOCK_MONOTONIC when IR
 * was decoded.
 * @flags: should be 0 for transmit. When receiving scancodes,
 * LIRC_SCANCODE_FLAG_TOGGLE or LIRC_SCANCODE_FLAG_REPEAT can be set
 * depending on the protocol
 * @rc_proto: see enum rc_proto
 * @keycode: the translated keycode. Set to 0 for transmit.
 * @scancode: the scancode received or to be sent
 */
struct lirc_scancode {
    __u64    timestamp;
    __u16    flags;
    __u16    rc_proto;
    __u32    keycode;
    __u64    scancode;
};

/* Set if the toggle bit of rc-5 or rc-6 is enabled */
#define LIRC_SCANCODE_FLAG_TOGGLE    1
/* Set if this is a nec or sanyo repeat */
#define LIRC_SCANCODE_FLAG_REPEAT    2

/**
 * enum rc_proto - the Remote Controller protocol
 *
 * @RC_PROTO_UNKNOWN: Protocol not known
 * @RC_PROTO_OTHER: Protocol known but proprietary
 * @RC_PROTO_RC5: Philips RC5 protocol
 * @RC_PROTO_RC5X_20: Philips RC5x 20 bit protocol
 * @RC_PROTO_RC5_SZ: StreamZap variant of RC5
 * @RC_PROTO_JVC: JVC protocol
 * @RC_PROTO_SONY12: Sony 12 bit protocol
 * @RC_PROTO_SONY15: Sony 15 bit protocol
 * @RC_PROTO_SONY20: Sony 20 bit protocol
 * @RC_PROTO_NEC: NEC protocol
 * @RC_PROTO_NECX: Extended NEC protocol

```

```
* @RC_PROTO_NEC32: NEC 32 bit protocol
* @RC_PROTO_SANY0: Sanyo protocol
* @RC_PROTO_MCIR2_KBD: RC6-ish MCE keyboard
* @RC_PROTO_MCIR2_MSE: RC6-ish MCE mouse
* @RC_PROTO_RC6_0: Philips RC6-0-16 protocol
* @RC_PROTO_RC6_6A_20: Philips RC6-6A-20 protocol
* @RC_PROTO_RC6_6A_24: Philips RC6-6A-24 protocol
* @RC_PROTO_RC6_6A_32: Philips RC6-6A-32 protocol
* @RC_PROTO_RC6_MCE: MCE (Philips RC6-6A-32 subtype) protocol
* @RC_PROTO_SHARP: Sharp protocol
* @RC_PROTO_XMP: XMP protocol
* @RC_PROTO_CEC: CEC protocol
* @RC_PROTO_IMON: iMon Pad protocol
* @RC_PROTO_RCMM12: RC-MM protocol 12 bits
* @RC_PROTO_RCMM24: RC-MM protocol 24 bits
* @RC_PROTO_RCMM32: RC-MM protocol 32 bits
* @RC_PROTO_XBOX_DVD: Xbox DVD Movie Playback Kit protocol
* @RC_PROTO_MAX: Maximum value of enum rc_proto
*/
enum rc_proto {
    RC_PROTO_UNKNOWN          = 0,
    RC_PROTO_OTHER            = 1,
    RC_PROTO_RC5              = 2,
    RC_PROTO_RC5X_20          = 3,
    RC_PROTO_RC5_SZ           = 4,
    RC_PROTO_JVC              = 5,
    RC_PROTO_SONY12           = 6,
    RC_PROTO_SONY15           = 7,
    RC_PROTO_SONY20           = 8,
    RC_PROTO_NEC              = 9,
    RC_PROTO_NECX             = 10,
    RC_PROTO_NEC32            = 11,
    RC_PROTO_SANY0            = 12,
    RC_PROTO_MCIR2_KBD        = 13,
    RC_PROTO_MCIR2_MSE        = 14,
    RC_PROTO_RC6_0            = 15,
    RC_PROTO_RC6_6A_20        = 16,
    RC_PROTO_RC6_6A_24        = 17,
    RC_PROTO_RC6_6A_32        = 18,
    RC_PROTO_RC6_MCE          = 19,
    RC_PROTO_SHARP            = 20,
    RC_PROTO_XMP              = 21,
    RC_PROTO_CEC              = 22,
    RC_PROTO_IMON             = 23,
    RC_PROTO_RCMM12           = 24,
    RC_PROTO_RCMM24           = 25,
    RC_PROTO_RCMM32           = 26,
    RC_PROTO_XBOX_DVD         = 27,
    RC_PROTO_MAX              = RC_PROTO_XBOX_DVD,
};
```

```
#endif
```

3.4.7 Revision and Copyright

Authors:

- Carvalho Chehab, Mauro <mchehab@kernel.org>
- Initial version.

Copyright © 2009-2016 : Mauro Carvalho Chehab

3.4.8 Revision History

revision 3.15 / 2014-02-06 (*mcc*)

Added the interface description and the RC sysfs class description.

revision 1.0 / 2009-09-06 (*mcc*)

Initial revision

3.5 Part IV - Media Controller API

3.5.1 Introduction

Media devices increasingly handle multiple related functions. Many USB cameras include microphones, video capture hardware can also output video, or SoC camera interfaces also perform memory-to-memory operations similar to video codecs.

Independent functions, even when implemented in the same hardware, can be modelled as separate devices. A USB camera with a microphone will be presented to userspace applications as V4L2 and ALSA capture devices. The devices' relationships (when using a webcam, end-users shouldn't have to manually select the associated USB microphone), while not made available directly to applications by the drivers, can usually be retrieved from sysfs.

With more and more advanced SoC devices being introduced, the current approach will not scale. Device topologies are getting increasingly complex and can't always be represented by a tree structure. Hardware blocks are shared between different functions, creating dependencies between seemingly unrelated devices.

Kernel abstraction APIs such as V4L2 and ALSA provide means for applications to access hardware parameters. As newer hardware expose an increasingly high number of those parameters, drivers need to guess what applications really require based on limited information, thereby implementing policies that belong to userspace.

The media controller API aims at solving those problems.

3.5.2 Media device model

Discovering a device internal topology, and configuring it at runtime, is one of the goals of the media controller API. To achieve this, hardware devices and Linux Kernel interfaces are modelled as graph objects on an oriented graph. The object types that constitute the graph are:

- An **entity** is a basic media hardware or software building block. It can correspond to a large variety of logical blocks such as physical hardware devices (CMOS sensor for instance), logical hardware devices (a building block in a System-on-Chip image processing pipeline), DMA channels or physical connectors.
- An **interface** is a graph representation of a Linux Kernel userspace API interface, like a device node or a sysfs file that controls one or more entities in the graph.
- A **pad** is a data connection endpoint through which an entity can interact with other entities. Data (not restricted to video) produced by an entity flows from the entity's output to one or more entity inputs. Pads should not be confused with physical pins at chip boundaries.
- A **data link** is a point-to-point oriented connection between two pads, either on the same entity or on different entities. Data flows from a source pad to a sink pad.
- An **interface link** is a point-to-point bidirectional control connection between a Linux Kernel interface and an entity.

3.5.3 Types and flags used to represent the media graph elements

Table 261: Media entity functions

MEDIA_ENT_F_UNKNOWN MEDIA_ENT_F_V4L2_SUBDEV_UNKNOWN	and	Unknown entity. That generally indicates that a driver didn't initialize properly the entity, which is a Kernel bug
MEDIA_ENT_F_IO_V4L		Data streaming input and/or output entity.
MEDIA_ENT_F_IO_VBI		V4L VBI streaming input or output entity
MEDIA_ENT_F_IO_SWRADIO		V4L Software Digital Radio (SDR) streaming input or output entity
MEDIA_ENT_F_IO_DTV		DVB Digital TV streaming input or output entity
MEDIA_ENT_F_DTV_DEMOD		Digital TV demodulator entity.
MEDIA_ENT_F_TS_DEMUX		MPEG Transport stream demux entity. Could be implemented on hardware or in Kernelspace by the Linux DVB subsystem.
MEDIA_ENT_F_DTV_CA		Digital TV Conditional Access module (CAM) entity
MEDIA_ENT_F_DTV_NET_DECAP		Digital TV network ULE/MLE desencapsulation entity. Could be implemented on hardware or in Kernelspace
MEDIA_ENT_F_CONN_RF		Connector for a Radio Frequency (RF) signal.
MEDIA_ENT_F_CONN_SVIDEO		Connector for a S-Video signal.
MEDIA_ENT_F_CONN_COMPOSITE		Connector for a RGB composite signal.
MEDIA_ENT_F_CAM_SENSOR		Camera video sensor entity.
MEDIA_ENT_F_FLASH		Flash controller entity.

Continued on next page

Table 261 - continued from previous page

MEDIA_ENT_F_LENS	Lens controller entity.
MEDIA_ENT_F_ATV_DECODER	Analog video decoder, the basic function of the video decoder is to accept analogue video from a wide variety of sources such as broadcast, DVD players, cameras and video cassette recorders, in either NTSC, PAL, SECAM or HD format, separating the stream into its component parts, luminance and chrominance, and output it in some digital video standard, with appropriate timing signals.
MEDIA_ENT_F_TUNER	Digital TV, analog TV, radio and/or software radio tuner, with consists on a PLL tuning stage that converts radio frequency (RF) signal into an Intermediate Frequency (IF). Modern tuners have internally IF-PLL decoders for audio and video, but older models have those stages implemented on separate entities.
MEDIA_ENT_F_IF_VID_DECODER	IF-PLL video decoder. It receives the IF from a PLL and decodes the analog TV video signal. This is commonly found on some very old analog tuners, like Philips MK3 designs. They all contain a tda9887 (or some software compatible similar chip, like tda9885). Those devices use a different I2C address than the tuner PLL.
MEDIA_ENT_F_IF_AUD_DECODER	IF-PLL sound decoder. It receives the IF from a PLL and decodes the analog TV audio signal. This is commonly found on some very old analog hardware, like Micronas msp3400, Philips tda9840, tda985x, etc. Those devices use a different I2C address than the tuner PLL and should be controlled together with the IF-PLL video decoder.
MEDIA_ENT_F_AUDIO_CAPTURE	Audio Capture Function Entity.
MEDIA_ENT_F_AUDIO_PLAYBACK	Audio Playback Function Entity.
MEDIA_ENT_F_AUDIO_MIXER	Audio Mixer Function Entity.
MEDIA_ENT_F_PROC_VIDEO_COMPOSER	Video composer (blender). An entity capable of video composing must have at least two sink pads and one source pad, and composes input video frames onto output video frames. Composition can be performed using alpha blending, color keying, raster operations (ROP), stitching or any other means.

Continued on next page

Table 261 - continued from previous page

MEDIA_ENT_F_PROC_VIDEO_PIXEL_FORMATTER	Video pixel formatter. An entity capable of pixel formatting must have at least one sink pad and one source pad. Read pixel formatters read pixels from memory and perform a subset of unpacking, cropping, color keying, alpha multiplication and pixel encoding conversion. Write pixel formatters perform a subset of dithering, pixel encoding conversion and packing and write pixels to memory.
MEDIA_ENT_F_PROC_VIDEO_PIXEL_ENC_CONVERTER	Video pixel encoding converter. An entity capable of pixel encoding conversion must have at least one sink pad and one source pad, and convert the encoding of pixels received on its sink pad(s) to a different encoding output on its source pad(s). Pixel encoding conversion includes but isn't limited to RGB to/from HSV, RGB to/from YUV and CFA (Bayer) to RGB conversions.
MEDIA_ENT_F_PROC_VIDEO_LUT	Video look-up table. An entity capable of video lookup table processing must have one sink pad and one source pad. It uses the values of the pixels received on its sink pad to look up entries in internal tables and output them on its source pad. The lookup processing can be performed on all components separately or combine them for multi-dimensional table lookups.
MEDIA_ENT_F_PROC_VIDEO_SCALER	Video scaler. An entity capable of video scaling must have at least one sink pad and one source pad, and scale the video frame(s) received on its sink pad(s) to a different resolution output on its source pad(s). The range of supported scaling ratios is entity-specific and can differ between the horizontal and vertical directions (in particular scaling can be supported in one direction only). Binning and sub-sampling (occasionally also referred to as skipping) are considered as scaling.
MEDIA_ENT_F_PROC_VIDEO_STATISTICS	Video statistics computation (histogram, 3A, etc.). An entity capable of statistics computation must have one sink pad and one source pad. It computes statistics over the frames received on its sink pad and outputs the statistics data on its source pad.
MEDIA_ENT_F_PROC_VIDEO_ENCODER	Video (MPEG, HEVC, VPx, etc.) encoder. An entity capable of compressing video frames. Must have one sink pad and at least one source pad.

Continued on next page

Table 261 - continued from previous page

MEDIA_ENT_F_PROC_VIDEO_DECODER	Video (MPEG, HEVC, VPx, etc.) decoder. An entity capable of decompressing a compressed video stream into uncompressed video frames. Must have one sink pad and at least one source pad.
MEDIA_ENT_F_PROC_VIDEO_ISP	An Image Signal Processor (ISP) device. ISPs generally are one of a kind devices that have their specific control interfaces using a combination of custom V4L2 controls and IOCTLs, and parameters supplied in a metadata buffer.
MEDIA_ENT_F_VID_MUX	Video multiplexer. An entity capable of multiplexing must have at least two sink pads and one source pad, and must pass the video frame(s) received from the active sink pad to the source pad.
MEDIA_ENT_F_VID_IF_BRIDGE	Video interface bridge. A video interface bridge entity must have at least one sink pad and at least one source pad. It receives video frames on its sink pad from an input video bus of one type (HDMI, eDP, MIPI CSI-2, etc.), and outputs them on its source pad to an output video bus of another type (eDP, MIPI CSI-2, parallel, etc.).
MEDIA_ENT_F_DV_DECODER	Digital video decoder. The basic function of the video decoder is to accept digital video from a wide variety of sources and output it in some digital video standard, with appropriate timing signals.
MEDIA_ENT_F_DV_ENCODER	Digital video encoder. The basic function of the video encoder is to accept digital video from some digital video standard with appropriate timing signals (usually a parallel video bus with sync signals) and output this to a digital video output connector such as HDMI or DisplayPort.

Table 262: Media entity flags

MEDIA_ENT_FL_DEFAULT	Default entity for its type. Used to discover the default audio, VBI and video devices, the default camera sensor, etc.
MEDIA_ENT_FL_CONNECTOR	The entity represents a connector.

Table 263: Media interface types

MEDIA_INTF_T_DVB_FE	Device node interface for the Digital TV frontend	typically, /dev/dvb/adapter?/frontend?
MEDIA_INTF_T_DVB_DEMUX	Device node interface for the Digital TV demux	typically, /dev/dvb/adapter?/demux?
MEDIA_INTF_T_DVB_DVR	Device node interface for the Digital TV DVR	typically, /dev/dvb/adapter?/dvr?
MEDIA_INTF_T_DVB_CA	Device node interface for the Digital TV Conditional Access	typically, /dev/dvb/adapter?/ca?
MEDIA_INTF_T_DVB_NET	Device node interface for the Digital TV network control	typically, /dev/dvb/adapter?/net?
MEDIA_INTF_T_V4L_VIDEO	Device node interface for video (V4L)	typically, /dev/video?
MEDIA_INTF_T_V4L_VBI	Device node interface for VBI (V4L)	typically, /dev/vbi?
MEDIA_INTF_T_V4L_RADIO	Device node interface for radio (V4L)	typically, /dev/radio?
MEDIA_INTF_T_V4L_SUBDEV	Device node interface for a V4L subdevice	typically, /dev/v4l-subdev?
MEDIA_INTF_T_V4L_SWRADIO	Device node interface for Software Defined Radio (V4L)	typically, /dev/swradio?
MEDIA_INTF_T_V4L_TOUCH	Device node interface for Touch device (V4L)	typically, /dev/v4l-touch?
MEDIA_INTF_T_ALSA_PCM_CAPTURE	Device node interface for ALSA PCM Capture	typically, /dev/snd/pcmC?D?c
MEDIA_INTF_T_ALSA_PCM_PLAYBACK	Device node interface for ALSA PCM Playback	typically, /dev/snd/pcmC?D?p
MEDIA_INTF_T_ALSA_CONTROL	Device node interface for ALSA Control	typically, /dev/snd/controlC?
MEDIA_INTF_T_ALSA_COMPRESS	Device node interface for ALSA Compress	typically, /dev/snd/compr?
MEDIA_INTF_T_ALSA_RAWMIDI	Device node interface for ALSA Raw MIDI	typically, /dev/snd/midi?
MEDIA_INTF_T_ALSA_HWDEP	Device node interface for ALSA Hardware Dependent	typically, /dev/snd/hwC?D?
MEDIA_INTF_T_ALSA_SEQUENCER	Device node interface for ALSA Sequencer	typically, /dev/snd/seq
MEDIA_INTF_T_ALSA_TIMER	Device node interface for ALSA Timer	typically, /dev/snd/timer

Table 264: Media pad flags

MEDIA_PAD_FL_SINK	Input pad, relative to the entity. Input pads sink data and are targets of links.
MEDIA_PAD_FL_SOURCE	Output pad, relative to the entity. Output pads source data and are origins of links.
MEDIA_PAD_FL_MUST_CONNECT	If this flag is set and the pad is linked to any other pad, then at least one of those links must be enabled for the entity to be able to stream. There could be temporary reasons (e.g. device configuration dependent) for the pad to need enabled links even when this flag isn't set; the absence of the flag doesn't imply there is none.

One and only one of MEDIA_PAD_FL_SINK and MEDIA_PAD_FL_SOURCE must be set for every pad.

Table 265: Media link flags

MEDIA_LNK_FL_ENABLED	The link is enabled and can be used to transfer media data. When two or more links target a sink pad, only one of them can be enabled at a time.
MEDIA_LNK_FL_IMMUTABLE	The link enabled state can't be modified at runtime. An immutable link is always enabled.
MEDIA_LNK_FL_DYNAMIC	The link enabled state can be modified during streaming. This flag is set by drivers and is read-only for applications.
MEDIA_LNK_FL_LINK_TYPE	This is a bitmask that defines the type of the link. Currently, two types of links are supported: MEDIA_LNK_FL_DATA_LINK if the link is between two pads MEDIA_LNK_FL_INTERFACE_LINK if the link is between an interface and an entity

3.5.4 Request API

The Request API has been designed to allow V4L2 to deal with requirements of modern devices (stateless codecs, complex camera pipelines, ...) and APIs (Android Codec v2). One such requirement is the ability for devices belonging to the same pipeline to reconfigure and collaborate closely on a per-frame basis. Another is support of stateless codecs, which require controls to be applied to specific frames (aka 'per-frame controls') in order to be used efficiently.

While the initial use-case was V4L2, it can be extended to other subsystems as well, as long as they use the media controller.

Supporting these features without the Request API is not always possible and if it is, it is terribly inefficient: user-space would have to flush all activity on the media pipeline, reconfigure it for the next frame, queue the buffers to be processed with that configuration, and wait until they are all available for dequeuing before considering the next frame. This defeats the purpose of having buffer queues since in practice only one buffer would be queued at a time.

The Request API allows a specific configuration of the pipeline (media controller topology + configuration for each media entity) to be associated with specific buffers. This allows user-space to schedule several tasks ("requests") with different configurations in advance, knowing that the configuration will be applied when needed to get the expected result. Configuration values at the time of request completion are also available for reading.

3.5.4.1 General Usage

The Request API extends the Media Controller API and cooperates with subsystem-specific APIs to support request usage. At the Media Controller level, requests are allocated from the supporting Media Controller device node. Their life cycle is then managed through the request file descriptors in an opaque way. Configuration data, buffer handles and processing results stored in requests are accessed through subsystem-specific APIs extended for request support, such as V4L2 APIs that take an explicit `request_fd` parameter.

3.5.4.2 Request Allocation

User-space allocates requests using `ioctl MEDIA_IOC_REQUEST_ALLOC` for the media device node. This returns a file descriptor representing the request. Typically, several such requests will be allocated.

3.5.4.3 Request Preparation

Standard V4L2 `ioctls` can then receive a request file descriptor to express the fact that the `ioctl` is part of said request, and is not to be applied immediately. See `ioctl MEDIA_IOC_REQUEST_ALLOC` for a list of `ioctls` that support this. Configurations set with a `request_fd` parameter are stored instead of being immediately applied, and buffers queued to a request do not enter the regular buffer queue until the request itself is queued.

3.5.4.4 Request Submission

Once the configuration and buffers of the request are specified, it can be queued by calling `ioctl MEDIA_REQUEST_IOC_QUEUE` on the request file descriptor. A request must contain at least one buffer, otherwise `ENOENT` is returned. A queued request cannot be modified anymore.

Caution: For *memory-to-memory devices* you can use requests only for output buffers, not for capture buffers. Attempting to add a capture buffer to a request will result in an `EBADR` error.

If the request contains configurations for multiple entities, individual drivers may synchronize so the requested pipeline's topology is applied before the buffers are processed. Media controller drivers do a best effort implementation since perfect atomicity may not be possible due to hardware limitations.

Caution: It is not allowed to mix queuing requests with directly queuing buffers: whichever method is used first locks this in place until `VIDIOC_STREAMOFF` is called or the device is *closed*. Attempts to directly queue a buffer when earlier a buffer was queued via a request or vice versa will result in an `EBUSY` error.

Controls can still be set without a request and are applied immediately, regardless of whether a request is in use or not.

Caution: Setting the same control through a request and also directly can lead to undefined behavior!

User-space can `poll()` a request file descriptor in order to wait until the request completes. A request is considered complete once all its associated buffers are available for dequeuing and all the associated controls have been updated with the values at the time of completion. Note that user-space does not need to wait for the request to complete to dequeue its buffers: buffers that are available halfway through a request can be dequeued independently of the request's state.

A completed request contains the state of the device after the request was executed. User-space can query that state by calling `ioctl VIDIOC_G_EXT_CTRL`s with the request file descriptor. Calling `ioctl VIDIOC_G_EXT_CTRL`s for a request that has been queued but not yet completed will return `EBUSY` since the control values might be changed at any time by the driver while the request is in flight.

3.5.4.5 Recycling and Destruction

Finally, a completed request can either be discarded or be reused. Calling `close()` on a request file descriptor will make that file descriptor unusable and the request will be freed once it is no longer in use by the kernel. That is, if the request is queued and then the file descriptor is closed, then it won't be freed until the driver completed the request.

The `ioctl MEDIA_REQUEST_IOC_REINIT` will clear a request's state and make it available again. No state is retained by this operation: the request is as if it had just been allocated.

3.5.4.6 Example for a Codec Device

For use-cases such as *codecs*, the request API can be used to associate specific controls to be applied by the driver for the OUTPUT buffer, allowing user-space to queue many such buffers in advance. It can also take advantage of requests' ability to capture the state of controls when the request completes to read back information that may be subject to change.

Put into code, after obtaining a request, user-space can assign controls and one OUTPUT buffer to it:

```

struct v4l2_buffer buf;
struct v4l2_ext_controls ctrls;
int req_fd;
...
if (ioctl(media_fd, MEDIA_IOC_REQUEST_ALLOC, &req_fd))
    return errno;
...
ctrls.which = V4L2_CTRL_WHICH_REQUEST_VAL;
ctrls.request_fd = req_fd;
if (ioctl(codec_fd, VIDIOC_S_EXT_CTRL, &ctrls))
    return errno;
...
buf.type = V4L2_BUF_TYPE_VIDEO_OUTPUT;
buf.flags |= V4L2_BUF_FLAG_REQUEST_FD;
buf.request_fd = req_fd;

```

```
if (ioctl(codec_fd, VIDIOC_QBUF, &buf))
    return errno;
```

Note that it is not allowed to use the Request API for CAPTURE buffers since there are no per-frame settings to report there.

Once the request is fully prepared, it can be queued to the driver:

```
if (ioctl(req_fd, MEDIA_REQUEST_IOC_QUEUE))
    return errno;
```

User-space can then either wait for the request to complete by calling `poll()` on its file descriptor, or start dequeuing CAPTURE buffers. Most likely, it will want to get CAPTURE buffers as soon as possible and this can be done using a regular `VIDIOC_DQBUF`:

```
struct v4l2_buffer buf;

memset(&buf, 0, sizeof(buf));
buf.type = V4L2_BUF_TYPE_VIDEO_CAPTURE;
if (ioctl(codec_fd, VIDIOC_DQBUF, &buf))
    return errno;
```

Note that this example assumes for simplicity that for every OUTPUT buffer there will be one CAPTURE buffer, but this does not have to be the case.

We can then, after ensuring that the request is completed via polling the request file descriptor, query control values at the time of its completion via a call to `VIDIOC_G_EXT_CTRL`s. This is particularly useful for volatile controls for which we want to query values as soon as the capture buffer is produced.

```
struct pollfd pfd = { .events = POLLPRI, .fd = req_fd };
poll(&pfd, 1, -1);
...
ctrls.which = V4L2_CTRL_WHICH_REQUEST_VAL;
ctrls.request_fd = req_fd;
if (ioctl(codec_fd, VIDIOC_G_EXT_CTRL, &ctrls))
    return errno;
```

Once we don't need the request anymore, we can either recycle it for reuse with `ioctl MEDIA_REQUEST_IOC_REINIT`...

```
if (ioctl(req_fd, MEDIA_REQUEST_IOC_REINIT))
    return errno;
```

... or close its file descriptor to completely dispose of it.

```
close(req_fd);
```


3.5.4.7 Example for a Simple Capture Device

With a simple capture device, requests can be used to specify controls to apply for a given CAPTURE buffer.

```

struct v4l2_buffer buf;
struct v4l2_ext_controls ctrls;
int req_fd;
...
if (ioctl(media_fd, MEDIA_IOC_REQUEST_ALLOC, &req_fd))
    return errno;
...
ctrls.which = V4L2_CTRL_WHICH_REQUEST_VAL;
ctrls.request_fd = req_fd;
if (ioctl(camera_fd, VIDIOC_S_EXT_CTRL, &ctrls))
    return errno;
...
buf.type = V4L2_BUF_TYPE_VIDEO_CAPTURE;
buf.flags |= V4L2_BUF_FLAG_REQUEST_FD;
buf.request_fd = req_fd;
if (ioctl(camera_fd, VIDIOC_QBUF, &buf))
    return errno;

```

Once the request is fully prepared, it can be queued to the driver:

```

if (ioctl(req_fd, MEDIA_REQUEST_IOC_QUEUE))
    return errno;

```

User-space can then dequeue buffers, wait for the request completion, query controls and recycle the request as in the M2M example above.

3.5.5 Function Reference

3.5.5.1 media open()

Name

media-open - Open a media device

Synopsis

```
#include <fcntl.h>
```

```
int open(const char *device_name, int flags)
```

Arguments

device_name Device to be opened.

flags Open flags. Access mode must be either `O_RDONLY` or `O_RDWR`. Other flags have no effect.

Description

To open a media device applications call `open()` with the desired device name. The function has no side effects; the device configuration remain unchanged.

When the device is opened in read-only mode, attempts to modify its configuration will result in an error, and `errno` will be set to `EBADF`.

Return Value

`open()` returns the new file descriptor on success. On error, `-1` is returned, and `errno` is set appropriately. Possible error codes are:

EACCES The requested access to the file is not allowed.

EMFILE The process already has the maximum number of files open.

ENFILE The system limit on the total number of open files has been reached.

ENOMEM Insufficient kernel memory was available.

ENXIO No device corresponding to this device special file exists.

3.5.5.2 media close()

Name

`media-close` - Close a media device

Synopsis

```
#include <unistd.h>
```

```
int close(int fd)
```

Arguments

fd File descriptor returned by `open()`.

Description

Closes the media device. Resources associated with the file descriptor are freed. The device configuration remain unchanged.

Return Value

`close()` returns 0 on success. On error, -1 is returned, and `errno` is set appropriately. Possible error codes are:

EBADF `fd` is not a valid open file descriptor.

3.5.5.3 media ioctl()

Name

media-ioctl - Control a media device

Synopsis

```
#include <sys/ioctl.h>
```

```
int ioctl(int fd, int request, void *argp)
```

Arguments

fd File descriptor returned by `open()`.

request Media ioctl request code as defined in the `media.h` header file, for example `MEDIA_IOC_SETUP_LINK`.

argp Pointer to a request-specific structure.

Description

The `ioctl()` function manipulates media device parameters. The argument `fd` must be an open file descriptor.

The `ioctl` request code specifies the media function to be called. It has encoded in it whether the argument is an input, output or read/write parameter, and the size of the argument `argp` in bytes.

Macros and structures definitions specifying media ioctl requests and their parameters are located in the `media.h` header file. All media ioctl requests, their respective function and parameters are specified in *Function Reference*.

Return Value

On success 0 is returned, on error -1 and the `errno` variable is set appropriately. The generic error codes are described at the *Generic Error Codes* chapter.

Request-specific error codes are listed in the individual requests descriptions.

When an ioctl that takes an output or read/write parameter fails, the parameter remains unmodified.

3.5.5.4 ioctl MEDIA_IOC_DEVICE_INFO

Name

`MEDIA_IOC_DEVICE_INFO` - Query device information

Synopsis

`MEDIA_IOC_DEVICE_INFO`

```
int ioctl(int fd, MEDIA_IOC_DEVICE_INFO, struct media_device_info *argp)
```

Arguments

fd File descriptor returned by `open()`.

argp Pointer to `struct media_device_info`.

Description

All media devices must support the `MEDIA_IOC_DEVICE_INFO` ioctl. To query device information, applications call the ioctl with a pointer to a `struct media_device_info`. The driver fills the structure and returns the information to the application. The ioctl never fails.

media_device_info

Table 266: struct media_device_info

char	driver[16]	Name of the driver implementing the media API as a NUL-terminated ASCII string. The driver version is stored in the <code>driver_version</code> field. Driver specific applications can use this information to verify the driver identity. It is also useful to work around known bugs, or to identify drivers in error reports.
char	model[32]	Device model name as a NUL-terminated UTF-8 string. The device version is stored in the <code>device_version</code> field and is not be appended to the model name.
char	serial[40]	Serial number as a NUL-terminated ASCII string.
char	bus_info[32]	Location of the device in the system as a NUL-terminated ASCII string. This includes the bus type name (PCI, USB, ...) and a bus-specific identifier.
__u32	media_version	Media API version, formatted with the <code>KERNEL_VERSION()</code> macro.
__u32	hw_revision	Hardware device revision in a driver-specific format.
__u32	driver_version	Media device driver version, formatted with the <code>KERNEL_VERSION()</code> macro. Together with the <code>driver</code> field this identifies a particular driver.
__u32	reserved[31]	Reserved for future extensions. Drivers and applications must set this array to zero.

The `serial` and `bus_info` fields can be used to distinguish between multiple instances of otherwise identical hardware. The serial number takes precedence when provided and can be assumed to be unique. If the serial number is an empty string, the `bus_info` field can be used instead. The `bus_info` field is guaranteed to be unique, but can vary across reboots or device unplug/replug.

Return Value

On success 0 is returned, on error -1 and the `errno` variable is set appropriately. The generic error codes are described at the [Generic Error Codes](#) chapter.

3.5.5.5 ioctl MEDIA_IOC_G_TOPOLOGY

Name

MEDIA_IOC_G_TOPOLOGY - Enumerate the graph topology and graph element properties

Synopsis

MEDIA_IOC_G_TOPOLOGY

```
int ioctl(int fd, MEDIA_IOC_G_TOPOLOGY, struct media_v2_topology *argp)
```

Arguments

fd File descriptor returned by `open()`.

argp Pointer to struct `media_v2_topology`.

Description

The typical usage of this ioctl is to call it twice. On the first call, the structure defined at struct `media_v2_topology` should be zeroed. At return, if no errors happen, this ioctl will return the `topology_version` and the total number of entities, interfaces, pads and links.

Before the second call, the userspace should allocate arrays to store the graph elements that are desired, putting the pointers to them at the `ptr_entities`, `ptr_interfaces`, `ptr_links` and/or `ptr_pads`, keeping the other values untouched.

If the `topology_version` remains the same, the ioctl should fill the desired arrays with the media graph elements.

`media_v2_topology`

Table 267: struct media_v2_topology

__u64	topology_version	Version of the media graph topology. When the graph is created, this field starts with zero. Every time a graph element is added or removed, this field is incremented.
__u32	num_entities	Number of entities in the graph
__u32	reserved1	Applications and drivers shall set this to 0.
__u64	ptr_entities	A pointer to a memory area where the entities array will be stored, converted to a 64-bits integer. It can be zero. if zero, the ioctl won't store the entities. It will just update num_entities
__u32	num_interfaces	Number of interfaces in the graph
__u32	reserved2	Applications and drivers shall set this to 0.
__u64	ptr_interfaces	A pointer to a memory area where the interfaces array will be stored, converted to a 64-bits integer. It can be zero. if zero, the ioctl won't store the interfaces. It will just update num_interfaces
__u32	num_pads	Total number of pads in the graph
__u32	reserved3	Applications and drivers shall set this to 0.
__u64	ptr_pads	A pointer to a memory area where the pads array will be stored, converted to a 64-bits integer. It can be zero. if zero, the ioctl won't store the pads. It will just update num_pads
__u32	num_links	Total number of data and interface links in the graph
__u32	reserved4	Applications and drivers shall set this to 0.
__u64	ptr_links	A pointer to a memory area where the links array will be stored, converted to a 64-bits integer. It can be zero. if zero, the ioctl won't store the links. It will just update num_links

media_v2_entity

Table 268: struct media_v2_entity

__u32	id	Unique ID for the entity. Do not expect that the ID will always be the same for each instance of the device. In other words, do not hardcode entity IDs in an application.
char	name[64]	Entity name as an UTF-8 NULL-terminated string. This name must be unique within the media topology.
__u32	function	Entity main function, see Media entity functions for details.
__u32	flags	Entity flags, see Media entity flags for details. Only valid if MEDIA_V2_ENTITY_HAS_FLAGS(media_version) returns true. The media_version is defined in struct media_device_info and can be retrieved using <code>ioctl MEDIA_IOC_DEVICE_INFO</code> .
__u32	reserved[5]	Reserved for future extensions. Drivers and applications must set this array to zero.

media_v2_interface

Table 269: struct media_v2_interface

__u32	id	Unique ID for the interface. Do not expect that the ID will always be the same for each instance of the device. In other words, do not hardcode interface IDs in an application.
__u32	intf_type	Interface type, see Media interface types for details.
__u32	flags	Interface flags. Currently unused.
__u32	reserved[9]	Reserved for future extensions. Drivers and applications must set this array to zero.
struct media_v2_intf_devnode	devnode	Used only for device node interfaces. See media_v2_intf_devnode for details.

media_v2_intf_devnode

Table 270: struct media_v2_intf_devnode

__u32	major	Device node major number.
__u32	minor	Device node minor number.

media_v2_pad

Table 271: struct media_v2_pad

__u32	id	Unique ID for the pad. Do not expect that the ID will always be the same for each instance of the device. In other words, do not hardcode pad IDs in an application.
__u32	entity_id	Unique ID for the entity where this pad belongs.
__u32	flags	Pad flags, see Media pad flags for more details.
__u32	index	Pad index, starts at 0. Only valid if <code>MEDIA_V2_PAD_HAS_INDEX(media_version)</code> returns true. The <code>media_version</code> is defined in struct <code>media_device_info</code> and can be retrieved using <code>ioctl MEDIA_IOC_DEVICE_INFO</code> .
__u32	reserved[4]	Reserved for future extensions. Drivers and applications must set this array to zero.

media_v2_link

Table 272: struct media_v2_link

__u32	id	Unique ID for the link. Do not expect that the ID will always be the same for each instance of the device. In other words, do not hardcode link IDs in an application.
__u32	source_id	On pad to pad links: unique ID for the source pad. On interface to entity links: unique ID for the interface.
__u32	sink_id	On pad to pad links: unique ID for the sink pad. On interface to entity links: unique ID for the entity.
__u32	flags	Link flags, see Media link flags for more details.
__u32	reserved[6]	Reserved for future extensions. Drivers and applications must set this array to zero.

Return Value

On success 0 is returned, on error -1 and the `errno` variable is set appropriately. The generic error codes are described at the [Generic Error Codes](#) chapter.

ENOSPC This is returned when either one or more of the `num_entities`, `num_interfaces`, `num_links` or `num_pads` are non-zero and are smaller than the actual number of elements inside the graph. This may happen if the `topology_version` changed when compared to the last time this `ioctl` was called. Userspace should usually free the area for the pointers, zero the struct elements and call this `ioctl` again.

3.5.5.6 `ioctl` MEDIA_IOC_ENUM_ENTITIES

Name

`MEDIA_IOC_ENUM_ENTITIES` - Enumerate entities and their properties

Synopsis

`MEDIA_IOC_ENUM_ENTITIES`

```
int ioctl(int fd, MEDIA_IOC_ENUM_ENTITIES, struct media_entity_desc *argp)
```

Arguments

fd File descriptor returned by `open()`.

argp Pointer to struct `media_entity_desc`.

Description

To query the attributes of an entity, applications set the `id` field of a struct `media_entity_desc` structure and call the `MEDIA_IOC_ENUM_ENTITIES` `ioctl` with a pointer to this structure. The driver fills the rest of the structure or returns an `EINVAL` error code when the `id` is invalid.

Entities can be enumerated by or'ing the `id` with the `MEDIA_ENT_ID_FLAG_NEXT` flag. The driver will return information about the entity with the smallest `id` strictly larger than the requested one ('next entity'), or the `EINVAL` error code if there is none.

Entity IDs can be non-contiguous. Applications must *not* try to enumerate entities by calling `MEDIA_IOC_ENUM_ENTITIES` with increasing `id`'s until they get an error.

`media_entity_desc`

Table 273: struct media_entity_desc

__u32	id		Entity ID, set by the application. When the ID is or'ed with MEDIA_ENT_ID_FLAG_NEXT, the driver clears the flag and returns the first entity with a larger ID. Do not expect that the ID will always be the same for each instance of the device. In other words, do not hard-code entity IDs in an application.
char	name[32]		Entity name as an UTF-8
			terminated string. This

Return Value

On success 0 is returned, on error -1 and the `errno` variable is set appropriately. The generic error codes are described at the [Generic Error Codes](#) chapter.

EINVAL The struct `media_entity_desc` `id` references a non-existing entity.

3.5.5.7 ioctl MEDIA_IOC_ENUM_LINKS

Name

`MEDIA_IOC_ENUM_LINKS` - Enumerate all pads and links for a given entity

Synopsis

`MEDIA_IOC_ENUM_LINKS`

```
int ioctl(int fd, MEDIA_IOC_ENUM_LINKS, struct media_links_enum *argp)
```

Arguments

fd File descriptor returned by `open()`.

argp Pointer to struct `media_links_enum`.

Description

To enumerate pads and/or links for a given entity, applications set the `entity` field of a struct `media_links_enum` structure and initialize the struct `media_pad_desc` and struct `media_link_desc` structure arrays pointed by the `pads` and `links` fields. They then call the `MEDIA_IOC_ENUM_LINKS` `ioctl` with a pointer to this structure.

If the `pads` field is not `NULL`, the driver fills the `pads` array with information about the entity's pads. The array must have enough room to store all the entity's pads. The number of pads can be retrieved with `ioctl MEDIA_IOC_ENUM_ENTITIES`.

If the `links` field is not `NULL`, the driver fills the `links` array with information about the entity's outbound links. The array must have enough room to store all the entity's outbound links. The number of outbound links can be retrieved with `ioctl MEDIA_IOC_ENUM_ENTITIES`.

Only forward links that originate at one of the entity's source pads are returned during the enumeration process.

`media_links_enum`

Table 274: struct media_links_enum

__u32	entity	Entity id, set by the application.
struct media_pad_desc	*pads	Pointer to a pads array allocated by the application. Ignored if NULL.
struct media_link_desc	*links	Pointer to a links array allocated by the application. Ignored if NULL.
__u32	reserved[4]	Reserved for future extensions. Drivers and applications must set the array to zero.

media_pad_desc

Table 275: struct media_pad_desc

__u32	entity	ID of the entity this pad belongs to.
__u16	index	Pad index, starts at 0.
__u32	flags	Pad flags, see <i>Media pad flags</i> for more details.
__u32	reserved[2]	Reserved for future extensions. Drivers and applications must set the array to zero.

media_link_desc

Table 276: struct media_link_desc

struct media_pad_desc	source	Pad at the origin of this link.
struct media_pad_desc	sink	Pad at the target of this link.
__u32	flags	Link flags, see <i>Media link flags</i> for more details.
__u32	reserved[2]	Reserved for future extensions. Drivers and applications must set the array to zero.

Return Value

On success 0 is returned, on error -1 and the `errno` variable is set appropriately. The generic error codes are described at the [Generic Error Codes](#) chapter.

EINVAL The struct `media_links_enum` id references a non-existing entity.

3.5.5.8 ioctl MEDIA_IOC_SETUP_LINK**Name**

`MEDIA_IOC_SETUP_LINK` - Modify the properties of a link

Synopsis

MEDIA_IOC_SETUP_LINK

```
int ioctl(int fd, MEDIA_IOC_SETUP_LINK, struct media_link_desc *argp)
```

Arguments

fd File descriptor returned by `open()`.

argp Pointer to struct `media_link_desc`.

Description

To change link properties applications fill a struct `media_link_desc` with link identification information (source and sink pad) and the new requested link flags. They then call the `MEDIA_IOC_SETUP_LINK` `ioctl` with a pointer to that structure.

The only configurable property is the `ENABLED` link flag to enable/disable a link. Links marked with the `IMMUTABLE` link flag can not be enabled or disabled.

Link configuration has no side effect on other links. If an enabled link at the sink pad prevents the link from being enabled, the driver returns with an `EBUSY` error code.

Only links marked with the `DYNAMIC` link flag can be enabled/disabled while streaming media data. Attempting to enable or disable a streaming non-dynamic link will return an `EBUSY` error code.

If the specified link can't be found the driver returns with an `EINVAL` error code.

Return Value

On success 0 is returned, on error -1 and the `errno` variable is set appropriately. The generic error codes are described at the [Generic Error Codes](#) chapter.

EINVAL The struct `media_link_desc` references a non-existing link, or the link is immutable and an attempt to modify its configuration was made.

3.5.5.9 ioctl MEDIA_IOC_REQUEST_ALLOC

Name

`MEDIA_IOC_REQUEST_ALLOC` - Allocate a request

Synopsis

MEDIA_IOC_REQUEST_ALLOC

```
int ioctl(int fd, MEDIA_IOC_REQUEST_ALLOC, int *argp)
```

Arguments

fd File descriptor returned by `open()`.

argp Pointer to an integer.

Description

If the media device supports *requests*, then this `ioctl` can be used to allocate a request. If it is not supported, then `errno` is set to `ENOTTY`. A request is accessed through a file descriptor that is returned in `*argp`.

If the request was successfully allocated, then the request file descriptor can be passed to the `VIDIOC_QBUF`, `VIDIOC_G_EXT_CTRL`s, `VIDIOC_S_EXT_CTRL`s and `VIDIOC_TRY_EXT_CTRL`s `ioctls`.

In addition, the request can be queued by calling `ioctl MEDIA_REQUEST_IOC_QUEUE` and re-initialized by calling `ioctl MEDIA_REQUEST_IOC_REINIT`.

Finally, the file descriptor can be *polled* to wait for the request to complete.

The request will remain allocated until all the file descriptors associated with it are closed by `close()` and the driver no longer uses the request internally. See also *here* for more information.

Return Value

On success 0 is returned, on error -1 and the `errno` variable is set appropriately. The generic error codes are described at the *Generic Error Codes* chapter.

ENOTTY The driver has no support for requests.

3.5.5.10 request close()

Name

request-close - Close a request file descriptor

Synopsis

```
#include <unistd.h>
```

```
int close(int fd)
```

Arguments

fd File descriptor returned by *ioctl* `MEDIA_IOC_REQUEST_ALLOC`.

Description

Closes the request file descriptor. Resources associated with the request are freed once all file descriptors associated with the request are closed and the driver has completed the request. See [here](#) for more information.

Return Value

`close()` returns 0 on success. On error, -1 is returned, and `errno` is set appropriately. Possible error codes are:

EBADF `fd` is not a valid open file descriptor.

3.5.5.11 request ioctl()

Name

request-ioctl - Control a request file descriptor

Synopsis

```
#include <sys/ioctl.h>
```

```
int ioctl(int fd, int cmd, void *argp)
```

Arguments

fd File descriptor returned by *ioctl* `MEDIA_IOC_REQUEST_ALLOC`.

cmd The request *ioctl* command code as defined in the `media.h` header file, for example *ioctl* `MEDIA_REQUEST_IOC_QUEUE`.

argp Pointer to a request-specific structure.

Description

The *ioctl()* function manipulates request parameters. The argument *fd* must be an open file descriptor.

The *ioctl cmd* code specifies the request function to be called. It has encoded in it whether the argument is an input, output or read/write parameter, and the size of the argument *argp* in bytes.

Macros and structures definitions specifying request *ioctl* commands and their parameters are located in the *media.h* header file. All request *ioctl* commands, their respective function and parameters are specified in *Function Reference*.

Return Value

On success 0 is returned, on error -1 and the *errno* variable is set appropriately. The generic error codes are described at the *Generic Error Codes* chapter.

Command-specific error codes are listed in the individual command descriptions.

When an *ioctl* that takes an output or read/write parameter fails, the parameter remains unmodified.

3.5.5.12 request poll()

Name

request-poll - Wait for some event on a file descriptor

Synopsis

```
#include <sys/poll.h>
```

```
int poll(struct pollfd *ufds, unsigned int nfds, int timeout)
```

Arguments

ufds List of file descriptor events to be watched

nfds Number of file descriptor events at the **ufds* array

timeout Timeout to wait for events

Description

With the `poll()` function applications can wait for a request to complete.

On success `poll()` returns the number of file descriptors that have been selected (that is, file descriptors for which the `revents` field of the respective struct `pollfd` is non-zero). Request file descriptor set the `POLLPRI` flag in `revents` when the request was completed. When the function times out it returns a value of zero, on failure it returns -1 and the `errno` variable is set appropriately.

Attempting to poll for a request that is not yet queued will set the `POLLERR` flag in `revents`.

Return Value

On success, `poll()` returns the number of structures which have non-zero `revents` fields, or zero if the call timed out. On error -1 is returned, and the `errno` variable is set appropriately:

EBADF One or more of the `ufds` members specify an invalid file descriptor.

EFAULT `ufds` references an inaccessible memory area.

EINTR The call was interrupted by a signal.

EINVAL The `nfds` value exceeds the `RLIMIT_NOFILE` value. Use `getrlimit()` to obtain this value.

3.5.5.13 `ioctl MEDIA_REQUEST_IOC_QUEUE`

Name

`MEDIA_REQUEST_IOC_QUEUE` - Queue a request

Synopsis

`MEDIA_REQUEST_IOC_QUEUE`

```
int ioctl(int request_fd, MEDIA_REQUEST_IOC_QUEUE)
```

Arguments

request_fd File descriptor returned by `ioctl MEDIA_IOC_REQUEST_ALLOC`.

Description

If the media device supports *requests*, then this request ioctl can be used to queue a previously allocated request.

If the request was successfully queued, then the file descriptor can be *polled* to wait for the request to complete.

If the request was already queued before, then EBUSY is returned. Other errors can be returned if the contents of the request contained invalid or inconsistent data, see the next section for a list of common error codes. On error both the request and driver state are unchanged.

Once a request is queued, then the driver is required to gracefully handle errors that occur when the request is applied to the hardware. The exception is the EIO error which signals a fatal error that requires the application to stop streaming to reset the hardware state.

It is not allowed to mix queuing requests with queuing buffers directly (without a request). EBUSY will be returned if the first buffer was queued directly and you next try to queue a request, or vice versa.

A request must contain at least one buffer, otherwise this ioctl will return an ENOENT error.

Return Value

On success 0 is returned, on error -1 and the errno variable is set appropriately. The generic error codes are described at the *Generic Error Codes* chapter.

EBUSY The request was already queued or the application queued the first buffer directly, but later attempted to use a request. It is not permitted to mix the two APIs.

ENOENT The request did not contain any buffers. All requests are required to have at least one buffer. This can also be returned if some required configuration is missing in the request.

ENOMEM Out of memory when allocating internal data structures for this request.

EINVAL The request has invalid data.

EIO The hardware is in a bad state. To recover, the application needs to stop streaming to reset the hardware state and then try to restart streaming.

3.5.5.14 ioctl MEDIA_REQUEST_IOC_REINIT

Name

MEDIA_REQUEST_IOC_REINIT - Re-initialize a request

Synopsis

MEDIA_REQUEST_IOC_REINIT

```
int ioctl(int request_fd, MEDIA_REQUEST_IOC_REINIT)
```

Arguments

request_fd File descriptor returned by *ioctl MEDIA_IOC_REQUEST_ALLOC*.

Description

If the media device supports *requests*, then this request ioctl can be used to re-initialize a previously allocated request.

Re-initializing a request will clear any existing data from the request. This avoids having to `close()` a completed request and allocate a new request. Instead the completed request can just be re-initialized and it is ready to be used again.

A request can only be re-initialized if it either has not been queued yet, or if it was queued and completed. Otherwise it will set `errno` to `EBUSY`. No other error codes can be returned.

Return Value

On success 0 is returned, on error -1 and the `errno` variable is set appropriately.

EBUSY The request is queued but not yet completed.

3.5.6 Media Controller Header File

3.5.6.1 media.h

```
/* SPDX-License-Identifier: GPL-2.0 WITH Linux-syscall-note */
/*
 * Multimedia device API
 *
 * Copyright (C) 2010 Nokia Corporation
 *
 * Contacts: Laurent Pinchart <laurent.pinchart@ideasonboard.com>
 *           Sakari Ailus <sakari.ailus@iki.fi>
 *
 * This program is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License version 2 as
 * published by the Free Software Foundation.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
 * GNU General Public License for more details.
```

```
*/

#ifdef __LINUX_MEDIA_H
#define __LINUX_MEDIA_H

#ifdef __KERNEL__
#include <stdint.h>
#endif
#include <linux/ioctl.h>
#include <linux/types.h>

struct media_device_info {
    char driver[16];
    char model[32];
    char serial[40];
    char bus_info[32];
    __u32 media_version;
    __u32 hw_revision;
    __u32 driver_version;
    __u32 reserved[31];
};

/*
 * Base number ranges for entity functions
 *
 * NOTE: Userspace should not rely on these ranges to identify a group
 * of function types, as newer functions can be added with any name within
 * the full u32 range.
 *
 * Some older functions use the MEDIA_ENT_F_OLD*_BASE range. Do not
 * change this, this is for backwards compatibility. When adding new
 * functions always use MEDIA_ENT_F_BASE.
 */
#define MEDIA_ENT_F_BASE                0x00000000
#define MEDIA_ENT_F_OLD_BASE           0x00010000
#define MEDIA_ENT_F_OLD_SUBDEV_BASE    0x00020000

/*
 * Initial value to be used when a new entity is created
 * Drivers should change it to something useful.
 */
#define MEDIA_ENT_F_UNKNOWN             MEDIA_ENT_F_BASE

/*
 * Subdevs are initialized with MEDIA_ENT_F_V4L2_SUBDEV_UNKNOWN in order
 * to preserve backward compatibility. Drivers must change to the proper
 * subdev type before registering the entity.
 */
#define MEDIA_ENT_F_V4L2_SUBDEV_UNKNOWN MEDIA_ENT_F_OLD_SUBDEV_BASE

/*
```

```

* DVB entity functions
*/
#define MEDIA_ENT_F_DTV_DEMOD (MEDIA_ENT_F_BASE + 0x00001)
#define MEDIA_ENT_F_TS_DEMUX (MEDIA_ENT_F_BASE + 0x00002)
#define MEDIA_ENT_F_DTV_CA (MEDIA_ENT_F_BASE + 0x00003)
#define MEDIA_ENT_F_DTV_NET_DECAP (MEDIA_ENT_F_BASE + 0x00004)

/*
* I/O entity functions
*/
#define MEDIA_ENT_F_IO_V4L (MEDIA_ENT_F_OLD_BASE + 1)
#define MEDIA_ENT_F_IO_DTV (MEDIA_ENT_F_BASE + 0x01001)
#define MEDIA_ENT_F_IO_VBI (MEDIA_ENT_F_BASE + 0x01002)
#define MEDIA_ENT_F_IO_SWRADIO (MEDIA_ENT_F_BASE + 0x01003)

/*
* Sensor functions
*/
#define MEDIA_ENT_F_CAM_SENSOR (MEDIA_ENT_F_OLD_SUBDEV_BASE + ↵
↵1)
#define MEDIA_ENT_F_FLASH (MEDIA_ENT_F_OLD_SUBDEV_BASE + ↵
↵2)
#define MEDIA_ENT_F_LENS (MEDIA_ENT_F_OLD_SUBDEV_BASE + ↵
↵3)

/*
* Digital TV, analog TV, radio and/or software defined radio tuner functions.
*
* It is a responsibility of the master/bridge drivers to add connectors
* and links for MEDIA_ENT_F_TUNER. Please notice that some old tuners
* may require the usage of separate I2C chips to decode analog TV signals,
* when the master/bridge chipset doesn't have its own TV standard decoder.
* On such cases, the IF-PLL staging is mapped via one or two entities:
* MEDIA_ENT_F_IF_VID_DECODER and/or MEDIA_ENT_F_IF_AUD_DECODER.
*/
#define MEDIA_ENT_F_TUNER (MEDIA_ENT_F_OLD_SUBDEV_BASE + ↵
↵5)

/*
* Analog TV IF-PLL decoder functions
*
* It is a responsibility of the master/bridge drivers to create links
* for MEDIA_ENT_F_IF_VID_DECODER and MEDIA_ENT_F_IF_AUD_DECODER.
*/
#define MEDIA_ENT_F_IF_VID_DECODER (MEDIA_ENT_F_BASE + 0x02001)
#define MEDIA_ENT_F_IF_AUD_DECODER (MEDIA_ENT_F_BASE + 0x02002)

/*
* Audio entity functions
*/
#define MEDIA_ENT_F_AUDIO_CAPTURE (MEDIA_ENT_F_BASE + 0x03001)

```

```
#define MEDIA_ENT_F_AUDIO_PLAYBACK (MEDIA_ENT_F_BASE + 0x03002)
#define MEDIA_ENT_F_AUDIO_MIXER (MEDIA_ENT_F_BASE + 0x03003)

/*
 * Processing entity functions
 */
#define MEDIA_ENT_F_PROC_VIDEO_COMPOSER (MEDIA_ENT_F_BASE + 0x4001)
#define MEDIA_ENT_F_PROC_VIDEO_PIXEL_FORMATTER (MEDIA_ENT_F_BASE + 0x4002)
#define MEDIA_ENT_F_PROC_VIDEO_PIXEL_ENC_CONV (MEDIA_ENT_F_BASE + 0x4003)
#define MEDIA_ENT_F_PROC_VIDEO_LUT (MEDIA_ENT_F_BASE + 0x4004)
#define MEDIA_ENT_F_PROC_VIDEO_SCALER (MEDIA_ENT_F_BASE + 0x4005)
#define MEDIA_ENT_F_PROC_VIDEO_STATISTICS (MEDIA_ENT_F_BASE + 0x4006)
#define MEDIA_ENT_F_PROC_VIDEO_ENCODER (MEDIA_ENT_F_BASE + 0x4007)
#define MEDIA_ENT_F_PROC_VIDEO_DECODER (MEDIA_ENT_F_BASE + 0x4008)
#define MEDIA_ENT_F_PROC_VIDEO_ISP (MEDIA_ENT_F_BASE + 0x4009)

/*
 * Switch and bridge entity functions
 */
#define MEDIA_ENT_F_VID_MUX (MEDIA_ENT_F_BASE + 0x5001)
#define MEDIA_ENT_F_VID_IF_BRIDGE (MEDIA_ENT_F_BASE + 0x5002)

/*
 * Video decoder/encoder functions
 */
#define MEDIA_ENT_F_ATV_DECODER (MEDIA_ENT_F_OLD_SUBDEV_BASE + 4)
#define MEDIA_ENT_F_DV_DECODER (MEDIA_ENT_F_BASE + 0x6001)
#define MEDIA_ENT_F_DV_ENCODER (MEDIA_ENT_F_BASE + 0x6002)

/* Entity flags */
#define MEDIA_ENT_FL_DEFAULT (1 << 0)
#define MEDIA_ENT_FL_CONNECTOR (1 << 1)

/* OR with the entity id value to find the next entity */
#define MEDIA_ENT_ID_FLAG_NEXT (1U << 31)

struct media_entity_desc {
    __u32 id;
    char name[32];
    __u32 type;
    __u32 revision;
    __u32 flags;
    __u32 group_id;
    __u16 pads;
    __u16 links;

    __u32 reserved[4];

    union {
        /* Node specifications */
```

```

        struct {
            __u32 major;
            __u32 minor;
        } dev;

#if !defined(__KERNEL__)
/*
 * TODO: this shouldn't have been added without
 * actual drivers that use this. When the first real driver
 * appears that sets this information, special attention
 * should be given whether this information is 1) enough, and
 * 2) can deal with udev rules that rename devices. The struct
 * dev would not be sufficient for this since that does not
 * contain the subdevice information. In addition, struct dev
 * can only refer to a single device, and not to multiple (e.g.
 * pcm and mixer devices).
 */
    struct {
        __u32 card;
        __u32 device;
        __u32 subdevice;
    } alsa;

/*
 * DEPRECATED: previous node specifications. Kept just to
 * avoid breaking compilation. Use media_entity_desc.dev
 * instead.
 */
    struct {
        __u32 major;
        __u32 minor;
    } v4l;
    struct {
        __u32 major;
        __u32 minor;
    } fb;
    int dvb;

#endif

/* Sub-device specifications */
/* Nothing needed yet */
    __u8 raw[184];
};

};

#define MEDIA_PAD_FL_SINK (1 << 0)
#define MEDIA_PAD_FL_SOURCE (1 << 1)
#define MEDIA_PAD_FL_MUST_CONNECT (1 << 2)

struct media_pad_desc {
    __u32 entity; /* entity ID */

```

```
    __u16 index;          /* pad index */
    __u32 flags;         /* pad flags */
    __u32 reserved[2];
};

#define MEDIA_LNK_FL_ENABLED          (1 << 0)
#define MEDIA_LNK_FL_IMMUTABLE      (1 << 1)
#define MEDIA_LNK_FL_DYNAMIC        (1 << 2)

#define MEDIA_LNK_FL_LINK_TYPE      (0xf << 28)
# define MEDIA_LNK_FL_DATA_LINK     (0 << 28)
# define MEDIA_LNK_FL_INTERFACE_LINK (1 << 28)

struct media_link_desc {
    struct media_pad_desc source;
    struct media_pad_desc sink;
    __u32 flags;
    __u32 reserved[2];
};

struct media_links_enum {
    __u32 entity;
    /* Should have enough room for pads elements */
    struct media_pad_desc __user *pads;
    /* Should have enough room for links elements */
    struct media_link_desc __user *links;
    __u32 reserved[4];
};

/* Interface type ranges */

#define MEDIA_INTF_T_DVB_BASE        0x00000100
#define MEDIA_INTF_T_V4L_BASE       0x00000200

/* Interface types */

#define MEDIA_INTF_T_DVB_FE          (MEDIA_INTF_T_DVB_BASE)
#define MEDIA_INTF_T_DVB_DEMUX      (MEDIA_INTF_T_DVB_BASE + 1)
#define MEDIA_INTF_T_DVB_DVR        (MEDIA_INTF_T_DVB_BASE + 2)
#define MEDIA_INTF_T_DVB_CA         (MEDIA_INTF_T_DVB_BASE + 3)
#define MEDIA_INTF_T_DVB_NET        (MEDIA_INTF_T_DVB_BASE + 4)

#define MEDIA_INTF_T_V4L_VIDEO      (MEDIA_INTF_T_V4L_BASE)
#define MEDIA_INTF_T_V4L_VBI        (MEDIA_INTF_T_V4L_BASE + 1)
#define MEDIA_INTF_T_V4L_RADIO      (MEDIA_INTF_T_V4L_BASE + 2)
#define MEDIA_INTF_T_V4L_SUBDEV     (MEDIA_INTF_T_V4L_BASE + 3)
#define MEDIA_INTF_T_V4L_SWRADIO    (MEDIA_INTF_T_V4L_BASE + 4)
#define MEDIA_INTF_T_V4L_TOUCH      (MEDIA_INTF_T_V4L_BASE + 5)

#define MEDIA_INTF_T_ALSA_BASE      0x00000300
#define MEDIA_INTF_T_ALSA_PCM_CAPTURE (MEDIA_INTF_T_ALSA_BASE)
```



```

#define MEDIA_INTF_T_ALSA_PCM_PLAYBACK      (MEDIA_INTF_T_ALSA_BASE + 1)
#define MEDIA_INTF_T_ALSA_CONTROL          (MEDIA_INTF_T_ALSA_BASE + 2)

#if defined(__KERNEL__)

/*
 * Connector functions
 *
 * For now these should not be used in userspace, as some definitions may
 * change.
 *
 * It is the responsibility of the entity drivers to add connectors and links.
 */
#define MEDIA_ENT_F_CONN_RF                (MEDIA_ENT_F_BASE + 0x30001)
#define MEDIA_ENT_F_CONN_SVIDEO           (MEDIA_ENT_F_BASE + 0x30002)
#define MEDIA_ENT_F_CONN_COMPOSITE        (MEDIA_ENT_F_BASE + 0x30003)

#endif

/*
 * MC next gen API definitions
 */

/*
 * Appeared in 4.19.0.
 *
 * The media_version argument comes from the media_version field in
 * struct media_device_info.
 */
#define MEDIA_V2_ENTITY_HAS_FLAGS(media_version) \
    ((media_version) >= ((4 << 16) | (19 << 8) | 0))

struct media_v2_entity {
    __u32 id;
    char name[64];
    __u32 function;          /* Main function of the entity */
    __u32 flags;
    __u32 reserved[5];
} __attribute__((packed));

/* Should match the specific fields at media_intf_devnode */
struct media_v2_intf_devnode {
    __u32 major;
    __u32 minor;
} __attribute__((packed));

struct media_v2_interface {
    __u32 id;
    __u32 intf_type;
    __u32 flags;
    __u32 reserved[9];
}

```

```
        union {
            struct media_v2_intf_devnode devnode;
            __u32 raw[16];
        };
} __attribute__((packed));

/*
 * Appeared in 4.19.0.
 *
 * The media_version argument comes from the media_version field in
 * struct media_device_info.
 */
#define MEDIA_V2_PAD_HAS_INDEX(media_version) \
    ((media_version) >= ((4 << 16) | (19 << 8) | 0))

struct media_v2_pad {
    __u32 id;
    __u32 entity_id;
    __u32 flags;
    __u32 index;
    __u32 reserved[4];
} __attribute__((packed));

struct media_v2_link {
    __u32 id;
    __u32 source_id;
    __u32 sink_id;
    __u32 flags;
    __u32 reserved[6];
} __attribute__((packed));

struct media_v2_topology {
    __u64 topology_version;

    __u32 num_entities;
    __u32 reserved1;
    __u64 ptr_entities;

    __u32 num_interfaces;
    __u32 reserved2;
    __u64 ptr_interfaces;

    __u32 num_pads;
    __u32 reserved3;
    __u64 ptr_pads;

    __u32 num_links;
    __u32 reserved4;
    __u64 ptr_links;
} __attribute__((packed));
```

```

/* ioctls */

#define MEDIA_IOC_DEVICE_INFO    _IOWR('|', 0x00, struct media_device_info)
#define MEDIA_IOC_ENUM_ENTITIES  _IOWR('|', 0x01, struct media_entity_desc)
#define MEDIA_IOC_ENUM_LINKS    _IOWR('|', 0x02, struct media_links_enum)
#define MEDIA_IOC_SETUP_LINK    _IOWR('|', 0x03, struct media_link_desc)
#define MEDIA_IOC_G_TOPOLOGY    _IOWR('|', 0x04, struct media_v2_topology)
#define MEDIA_IOC_REQUEST_ALLOC  _IOR ('|', 0x05, int)

/*
 * These ioctls are called on the request file descriptor as returned
 * by MEDIA_IOC_REQUEST_ALLOC.
 */
#define MEDIA_REQUEST_IOC_QUEUE    _IO('|', 0x80)
#define MEDIA_REQUEST_IOC_REINIT   _IO('|', 0x81)

#ifdef __KERNEL__

/*
 * Legacy symbols used to avoid userspace compilation breakages.
 * Do not use any of this in new applications!
 *
 * Those symbols map the entity function into types and should be
 * used only on legacy programs for legacy hardware. Don't rely
 * on those for MEDIA_IOC_G_TOPOLOGY.
 */
#define MEDIA_ENT_TYPE_SHIFT      16
#define MEDIA_ENT_TYPE_MASK      0x00ff0000
#define MEDIA_ENT_SUBTYPE_MASK   0x0000ffff

#define MEDIA_ENT_T_DEVNODE_UNKNOWN (MEDIA_ENT_F_OLD_BASE | \
    MEDIA_ENT_SUBTYPE_MASK)

#define MEDIA_ENT_T_DEVNODE        MEDIA_ENT_F_OLD_BASE
#define MEDIA_ENT_T_DEVNODE_V4L    MEDIA_ENT_F_IO_V4L
#define MEDIA_ENT_T_DEVNODE_FB     (MEDIA_ENT_F_OLD_BASE + 2)
#define MEDIA_ENT_T_DEVNODE_ALSA   (MEDIA_ENT_F_OLD_BASE + 3)
#define MEDIA_ENT_T_DEVNODE_DVB    (MEDIA_ENT_F_OLD_BASE + 4)

#define MEDIA_ENT_T_UNKNOWN        MEDIA_ENT_F_UNKNOWN
#define MEDIA_ENT_T_V4L2_VIDEO      MEDIA_ENT_F_IO_V4L
#define MEDIA_ENT_T_V4L2_SUBDEV     MEDIA_ENT_F_V4L2_SUBDEV_UNKNOWN
#define MEDIA_ENT_T_V4L2_SUBDEV_SENSOR MEDIA_ENT_F_CAM_SENSOR
#define MEDIA_ENT_T_V4L2_SUBDEV_FLASH MEDIA_ENT_F_FLASH
#define MEDIA_ENT_T_V4L2_SUBDEV_LENS MEDIA_ENT_F_LENS
#define MEDIA_ENT_T_V4L2_SUBDEV_DECODER MEDIA_ENT_F_ATV_DECODER
#define MEDIA_ENT_T_V4L2_SUBDEV_TUNER MEDIA_ENT_F_TUNER

#define MEDIA_ENT_F_DTV_DECODER     MEDIA_ENT_F_DV_DECODER

```

```
/*
 * There is still no full ALSA support in the media controller. These
 * defines should not have been added and we leave them here only
 * in case some application tries to use these defines.
 *
 * The ALSA defines that are in use have been moved into __KERNEL__
 * scope. As support gets added to these interface types, they should
 * be moved into __KERNEL__ scope with the code that uses them.
 */
#define MEDIA_INTF_T_ALSA_COMPRESS (MEDIA_INTF_T_ALSA_BASE + 3)
#define MEDIA_INTF_T_ALSA_RAWMIDI (MEDIA_INTF_T_ALSA_BASE + 4)
#define MEDIA_INTF_T_ALSA_HWDEP (MEDIA_INTF_T_ALSA_BASE + 5)
#define MEDIA_INTF_T_ALSA_SEQUENCER (MEDIA_INTF_T_ALSA_BASE + 6)
#define MEDIA_INTF_T_ALSA_TIMER (MEDIA_INTF_T_ALSA_BASE + 7)

/* Obsolete symbol for media_version, no longer used in the kernel */
#define MEDIA_API_VERSION ((0 << 16) | (1 << 8) | 0)

#endif

#endif /* __LINUX_MEDIA_H */
```

3.5.7 Revision and Copyright

Authors:

- Pinchart, Laurent <laurent.pinchart@ideasonboard.com>
- Initial version.
- Carvalho Chehab, Mauro <mchehab@kernel.org>
- MEDIA_IOC_G_TOPOLOGY documentation and documentation improvements.

Copyright © 2010 : Laurent Pinchart

Copyright © 2015-2016 : Mauro Carvalho Chehab

3.5.8 Revision History

revision 1.1.0 / 2015-12-12 (*mcc*)

revision 1.0.0 / 2010-11-10 (*lp*)

Initial revision

3.6 Part V - Consumer Electronics Control API

This part describes the CEC: Consumer Electronics Control

3.6.1 Introduction

HDMI connectors provide a single pin for use by the Consumer Electronics Control protocol. This protocol allows different devices connected by an HDMI cable to communicate. The protocol for CEC version 1.4 is defined in supplements 1 (CEC) and 2 (HEAC or HDMI Ethernet and Audio Return Channel) of the HDMI 1.4a (*HDMI*) specification and the extensions added to CEC version 2.0 are defined in chapter 11 of the HDMI 2.0 (*HDMI2*) specification.

The bitrate is very slow (effectively no more than 36 bytes per second) and is based on the ancient AV.link protocol used in old SCART connectors. The protocol closely resembles a crazy Rube Goldberg contraption and is an unholy mix of low and high level messages. Some messages, especially those part of the HEAC protocol layered on top of CEC, need to be handled by the kernel, others can be handled either by the kernel or by userspace.

In addition, CEC can be implemented in HDMI receivers, transmitters and in USB devices that have an HDMI input and an HDMI output and that control just the CEC pin.

Drivers that support CEC will create a CEC device node (`/dev/cecX`) to give userspace access to the CEC adapter. The `ioctl CEC_ADAP_G_CAPS` ioctl will tell userspace what it is allowed to do.

In order to check the support and test it, it is suggested to download the `v4l-utils` package. It provides three tools to handle CEC:

- `cec-ctl`: the Swiss army knife of CEC. Allows you to configure, transmit and monitor CEC messages.
- `cec-compliance`: does a CEC compliance test of a remote CEC device to determine how compliant the CEC implementation is.
- `cec-follower`: emulates a CEC follower.

3.6.2 Function Reference

3.6.2.1 `cec open()`

Name

`cec-open` - Open a cec device

Synopsis

```
#include <fcntl.h>
```

int **open**(const char **device_name*, int *flags*)

Arguments

device_name Device to be opened.

flags Open flags. Access mode must be O_RDWR.

When the O_NONBLOCK flag is given, the *CEC_RECEIVE* and *CEC_DQEVENT* ioctls will return the EAGAIN error code when no message or event is available, and ioctls *CEC_TRANSMIT*, *CEC_ADAP_S_PHYS_ADDR* and *CEC_ADAP_S_LOG_ADDRS* all return 0.

Other flags have no effect.

Description

To open a cec device applications call `open()` with the desired device name. The function has no side effects; the device configuration remain unchanged.

When the device is opened in read-only mode, attempts to modify its configuration will result in an error, and `errno` will be set to EBADF.

Return Value

`open()` returns the new file descriptor on success. On error, -1 is returned, and `errno` is set appropriately. Possible error codes include:

EACCES The requested access to the file is not allowed.

EMFILE The process already has the maximum number of files open.

ENFILE The system limit on the total number of open files has been reached.

ENOMEM Insufficient kernel memory was available.

ENXIO No device corresponding to this device special file exists.

3.6.2.2 cec close()

Name

cec-close - Close a cec device

Synopsis

```
#include <unistd.h>
```

```
int close(int fd)
```

Arguments

fd File descriptor returned by `open()`.

Description

Closes the cec device. Resources associated with the file descriptor are freed. The device configuration remain unchanged.

Return Value

`close()` returns 0 on success. On error, -1 is returned, and `errno` is set appropriately. Possible error codes are:

EBADF `fd` is not a valid open file descriptor.

3.6.2.3 cec ioctl()

Name

cec-ioctl - Control a cec device

Synopsis

```
#include <sys/ioctl.h>
```

```
int ioctl(int fd, int request, void *argp)
```

Arguments

fd File descriptor returned by `open()`.

request CEC ioctl request code as defined in the `cec.h` header file, for example `CEC_ADAP_G_CAPS`.

argp Pointer to a request-specific structure.

Description

The `ioctl()` function manipulates cec device parameters. The argument `fd` must be an open file descriptor.

The `ioctl` request code specifies the cec function to be called. It has encoded in it whether the argument is an input, output or read/write parameter, and the size of the argument `argp` in bytes.

Macros and structures definitions specifying cec `ioctl` requests and their parameters are located in the `cec.h` header file. All cec `ioctl` requests, their respective function and parameters are specified in *Function Reference*.

Return Value

On success 0 is returned, on error -1 and the `errno` variable is set appropriately. The generic error codes are described at the *Generic Error Codes* chapter.

Request-specific error codes are listed in the individual requests descriptions.

When an `ioctl` that takes an output or read/write parameter fails, the parameter remains unmodified.

3.6.2.4 cec poll()

Name

`cec-poll` - Wait for some event on a file descriptor

Synopsis

```
#include <sys/poll.h>
```

```
int poll(struct pollfd *ufds, unsigned int nfds, int timeout)
```

Arguments

ufds List of FD events to be watched

nfds Number of FD events at the `*ufds` array

timeout Timeout to wait for events

Description

With the `poll()` function applications can wait for CEC events.

On success `poll()` returns the number of file descriptors that have been selected (that is, file descriptors for which the `revents` field of the respective struct `pollfd` is non-zero). CEC devices set the `POLLIN` and `POLLRDNORM` flags in the `revents` field if there are messages in the receive queue. If the transmit queue has room for new messages, the `POLLOUT` and `POLLWRNORM` flags are set. If there are events in the event queue, then the `POLLPRI` flag is set. When the function times out it returns a value of zero, on failure it returns -1 and the `errno` variable is set appropriately.

For more details see the `poll()` manual page.

Return Value

On success, `poll()` returns the number structures which have non-zero `revents` fields, or zero if the call timed out. On error -1 is returned, and the `errno` variable is set appropriately:

EBADF One or more of the `ufds` members specify an invalid file descriptor.

EFAULT `ufds` references an inaccessible memory area.

EINTR The call was interrupted by a signal.

EINVAL The `nfds` value exceeds the `RLIMIT_NOFILE` value. Use `getrlimit()` to obtain this value.

3.6.2.5 ioctl CEC_ADAP_G_CAPS

Name

`CEC_ADAP_G_CAPS` - Query device capabilities

Synopsis

`CEC_ADAP_G_CAPS`

```
int ioctl(int fd, CEC_ADAP_G_CAPS, struct cec_caps *argp)
```

Arguments

fd File descriptor returned by `open()`.

argp

Description

All cec devices must support *ioctl CEC_ADAP_G_CAPS*. To query device information, applications call the *ioctl* with a pointer to a struct *cec_caps*. The driver fills the structure and returns the information to the application. The *ioctl* never fails.

cec_caps

Table 277: struct *cec_caps*

char	driver[32]	The name of the cec adapter driver.
char	name[32]	The name of this CEC adapter. The combination <i>driver</i> and <i>name</i> must be unique.
__u32	available_logical_addresses	The maximum number of logical addresses that can be configured.
__u32	capabilities	The capabilities of the CEC adapter, see <i>CEC Capabilities Flags</i> .
__u32	version	CEC Framework API version, formatted with the <i>KERNEL_VERSION()</i> macro.

Table 278: CEC Capabilities Flags

CEC_CAP_PHYS_ADDR	0x00000001	Userspace has to configure the physical address by calling <i>ioctl CEC_ADAP_S_PHYS_ADDR</i> . If this capability isn't set, then setting the physical address is handled by the kernel whenever the EDID is set (for an HDMI receiver) or read (for an HDMI transmitter).
CEC_CAP_LOG_ADDRS	0x00000002	Userspace has to configure the logical addresses by calling <i>ioctl CEC_ADAP_S_LOG_ADDRS</i> . If this capability isn't set, then the kernel will have configured this.
CEC_CAP_TRANSMIT	0x00000004	Userspace can transmit CEC messages by calling <i>ioctl CEC_TRANSMIT</i> . This implies that userspace can be a follower as well, since being able to transmit messages is a prerequisite of becoming a follower. If this capability isn't set, then the kernel will handle all CEC transmits and process all CEC messages it receives.
CEC_CAP_PASSTHROUGH	0x00000008	Userspace can use the passthrough mode by calling <i>ioctl CEC_S_MODE</i> .
CEC_CAP_RC	0x00000010	This adapter supports the remote control protocol.
CEC_CAP_MONITOR_ALL	0x00000020	The CEC hardware can monitor all messages, not just directed and broadcast messages.
CEC_CAP_NEEDS_HPD	0x00000040	The CEC hardware is only active if the HDMI Hotplug Detect pin is high. This makes it impossible to use CEC to wake up displays that set the HPD pin low when in standby mode, but keep the CEC bus alive.
CEC_CAP_MONITOR_PIN	0x00000080	The CEC hardware can monitor CEC pin changes from low to high voltage and vice versa. When in pin monitoring mode the application will receive <code>CEC_EVENT_PIN_CEC_LOW</code> and <code>CEC_EVENT_PIN_CEC_HIGH</code> events.
CEC_CAP_CONNECTOR_INFO	0x00000100	If this capability is set, then <i>ioctl CEC_ADAP_G_CONNECTOR_INFO</i> can be used.

Return Value

On success 0 is returned, on error -1 and the `errno` variable is set appropriately. The generic error codes are described at the [Generic Error Codes](#) chapter.

3.6.2.6 `ioctl`s `CEC_ADAP_G_LOG_ADDRS` and `CEC_ADAP_S_LOG_ADDRS`

Name

`CEC_ADAP_G_LOG_ADDRS`, `CEC_ADAP_S_LOG_ADDRS` - Get or set the logical addresses

Synopsis

`CEC_ADAP_G_LOG_ADDRS`

```
int ioctl(int fd, CEC_ADAP_G_LOG_ADDRS, struct cec_log_addrs *argp)
```

`CEC_ADAP_S_LOG_ADDRS`

```
int ioctl(int fd, CEC_ADAP_S_LOG_ADDRS, struct cec_log_addrs *argp)
```

Arguments

fd File descriptor returned by `open()`.

argp Pointer to `struct cec_log_addrs`.

Description

To query the current CEC logical addresses, applications call `ioctl CEC_ADAP_G_LOG_ADDRS` with a pointer to a `struct cec_log_addrs` where the driver stores the logical addresses.

To set new logical addresses, applications fill in `struct cec_log_addrs` and call `ioctl CEC_ADAP_S_LOG_ADDRS` with a pointer to this struct. The `ioctl CEC_ADAP_S_LOG_ADDRS` is only available if `CEC_CAP_LOG_ADDRS` is set (the `ENOTTY` error code is returned otherwise). The `ioctl CEC_ADAP_S_LOG_ADDRS` can only be called by a file descriptor in initiator mode (see `ioctl`s `CEC_G_MODE` and `CEC_S_MODE`), if not the `EBUSY` error code will be returned.

To clear existing logical addresses set `num_log_addrs` to 0. All other fields will be ignored in that case. The adapter will go to the unconfigured state and the `cec_version`, `vendor_id` and `osd_name` fields are all reset to their default values (CEC version 2.0, no vendor ID and an empty OSD name).

If the physical address is valid (see `ioctl CEC_ADAP_S_PHYS_ADDR`), then this `ioctl` will block until all requested logical addresses have been claimed. If the file descriptor is in non-blocking mode then it will not wait for the logical addresses to be claimed, instead it just returns 0.

A `CEC_EVENT_STATE_CHANGE` event is sent when the logical addresses are claimed or cleared.

Attempting to call `ioctl CEC_ADAP_S_LOG_ADDRS` when logical address types are already defined will return with error `EBUSY`.

cec_log_addrs

Table 279: struct cec_log_addrs

__u8	log_addr[CEC_MAX_LOG_ADDRS]	The actual logical addresses that were claimed. This is set by the driver. If no logical address could be claimed, then it is set to CEC_LOG_ADDR_INVALID. If this adapter is Unregistered, then log_addr[0] is set to 0xf and all others to CEC_LOG_ADDR_INVALID.
__u16	log_addr_mask	The bitmask of all logical addresses this adapter has claimed. If this adapter is Unregistered then log_addr_mask sets bit 15 and clears all other bits. If this adapter is not configured at all, then log_addr_mask is set to 0. Set by the driver.
__u8	cec_version	The CEC version that this adapter shall use. See <i>CEC Versions</i> . Used to implement the CEC_MSG_CEC_VERSION and CEC_MSG_REPORT_FEATURES messages. Note that <i>CEC_OP_CEC_VERSION_1_3A</i> is not allowed by the CEC framework.
__u8	num_log_addrs	Number of logical addresses to set up. Must be \leq available_log_addrs as returned by <i>ioctl CEC_ADAP_G_CAPS</i> . All arrays in this structure are only filled up to index available_log_addrs-1. The remaining array elements will be ignored. Note that the CEC 2.0 standard allows for a maximum of 2 logical addresses, although some hardware has support for more. CEC_MAX_LOG_ADDRS is 4. The driver will return the actual number of logical addresses it could claim, which may be less than what was requested. If this field is set to 0, then the CEC adapter shall clear all claimed logical addresses and all other fields will be ignored.
__u32	vendor_id	The vendor ID is a 24-bit number that identifies the specific vendor or entity. Based on this ID vendor specific commands may be defined. If you do not want a vendor ID then set it to CEC_VENDOR_ID_NONE.
__u32	flags	Flags. See <i>Flags for struct cec_log_addrs</i> for a list of available flags.

Continued on next page

Table 279 - continued from previous page

char	osd_name[15]	The On-Screen Display name as is returned by the CEC_MSG_SET_OSD_NAME message.
__u8	primary_device_type[CEC_MAX_LOG_ADDRS]	Primary device type for each logical address. See <i>CEC Primary Device Types</i> for possible types.
__u8	log_addr_type[CEC_MAX_LOG_ADDRS]	Logical address types. See <i>CEC Logical Address Types</i> for possible types. The driver will update this with the actual logical address type that it claimed (e.g. it may have to fallback to <i>CEC_LOG_ADDR_TYPE_UNREGISTERED</i>).
__u8	all_device_types[CEC_MAX_LOG_ADDRS]	CEC 2.0 specific: the bit mask of all device types. See <i>CEC All Device Types Flags</i> . It is used in the CEC 2.0 CEC_MSG_REPORT_FEATURES message. For CEC 1.4 you can either leave this field to 0, or fill it in according to the CEC 2.0 guidelines to give the CEC framework more information about the device type, even though the framework won't use it directly in the CEC message.
__u8	features[CEC_MAX_LOG_ADDRS][12]	Features for each logical address. It is used in the CEC 2.0 CEC_MSG_REPORT_FEATURES message. The 12 bytes include both the RC Profile and the Device Features. For CEC 1.4 you can either leave this field to all 0, or fill it in according to the CEC 2.0 guidelines to give the CEC framework more information about the device type, even though the framework won't use it directly in the CEC message.

Table 280: Flags for struct `cec_log_addr`

<code>CEC_LOG_ADDRS_FL_ALLOW_UNREG_FALLBACK</code>	1	By default if no logical address of the requested type can be claimed, then it will go back to the unconfigured state. If this flag is set, then it will fallback to the Unregistered logical address. Note that if the Unregistered logical address was explicitly requested, then this flag has no effect.
<code>CEC_LOG_ADDRS_FL_ALLOW_RC_PASSTHRU</code>	2	By default the <code>CEC_MSG_USER_CONTROL_PRESSED</code> and <code>CEC_MSG_USER_CONTROL_RELEASED</code> messages are only passed on to the follower(s), if any. If this flag is set, then these messages are also passed on to the remote control input subsystem and will appear as keystrokes. This features needs to be enabled explicitly. If CEC is used to enter e.g. passwords, then you may not want to enable this to avoid trivial snooping of the keystrokes.
<code>CEC_LOG_ADDRS_FL_CDC_ONLY</code>	4	If this flag is set, then the device is CDC-Only. CDC-Only CEC devices are CEC devices that can only handle CDC messages. All other messages are ignored.

Table 281: CEC Versions

<code>CEC_OP_CEC_VERSION_1_3A</code>	4	CEC version according to the HDMI 1.3a standard.
<code>CEC_OP_CEC_VERSION_1_4B</code>	5	CEC version according to the HDMI 1.4b standard.
<code>CEC_OP_CEC_VERSION_2_0</code>	6	CEC version according to the HDMI 2.0 standard.

Table 282: CEC Primary Device Types

CEC_OP_PRIM_DEVTYPE_TV	0	Use for a TV.
CEC_OP_PRIM_DEVTYPE_RECORD	1	Use for a recording device.
CEC_OP_PRIM_DEVTYPE_TUNER	3	Use for a device with a tuner.
CEC_OP_PRIM_DEVTYPE_PLAYBACK	4	Use for a playback device.
CEC_OP_PRIM_DEVTYPE_AUDIOSYSTEM	5	Use for an audio system (e.g. an audio/video receiver).
CEC_OP_PRIM_DEVTYPE_SWITCH	6	Use for a CEC switch.
CEC_OP_PRIM_DEVTYPE_VIDEOPROCESSOR	7	Use for a video processor device.

Table 283: CEC Logical Address Types

CEC_LOG_ADDR_TYPE_TV	0	Use for a TV.
CEC_LOG_ADDR_TYPE_RECORD	1	Use for a recording device.
CEC_LOG_ADDR_TYPE_TUNER	2	Use for a tuner device.
CEC_LOG_ADDR_TYPE_PLAYBACK	3	Use for a playback device.
CEC_LOG_ADDR_TYPE_AUDIOSYSTEM	4	Use for an audio system device.
CEC_LOG_ADDR_TYPE_SPECIFIC	5	Use for a second TV or for a video processor device.
CEC_LOG_ADDR_TYPE_UNREGISTERED	6	Use this if you just want to remain unregistered. Used for pure CEC switches or CDC-only devices (CDC: Capability Discovery and Control).

Table 284: CEC All Device Types Flags

CEC_OP_ALL_DEVTYPE_TV	0x80	This supports the TV type.
CEC_OP_ALL_DEVTYPE_RECORD	0x40	This supports the Recording type.
CEC_OP_ALL_DEVTYPE_TUNER	0x20	This supports the Tuner type.
CEC_OP_ALL_DEVTYPE_PLAYBACK	0x10	This supports the Playback type.
CEC_OP_ALL_DEVTYPE_AUDIOSYSTEM	0x08	This supports the Audio System type.
CEC_OP_ALL_DEVTYPE_SWITCH	0x04	This supports the CEC Switch or Video Processing type.

Return Value

On success 0 is returned, on error -1 and the `errno` variable is set appropriately. The generic error codes are described at the [Generic Error Codes](#) chapter.

The `ioctl CEC_ADAP_S_LOG_ADDRS` can return the following error codes:

ENOTTY The `CEC_CAP_LOG_ADDRS` capability wasn't set, so this `ioctl` is not supported.

EBUSY The CEC adapter is currently configuring itself, or it is already configured and `num_log_addrs` is non-zero, or another filehandle is in exclusive follower or initiator mode, or the filehandle is in mode `CEC_MODE_NO_INITIATOR`.

EINVAL The contents of struct `cec_log_addrs` is invalid.

3.6.2.7 `ioctls CEC_ADAP_G_PHYS_ADDR` and `CEC_ADAP_S_PHYS_ADDR`

Name

`CEC_ADAP_G_PHYS_ADDR`, `CEC_ADAP_S_PHYS_ADDR` - Get or set the physical address

Synopsis

`CEC_ADAP_G_PHYS_ADDR`

```
int ioctl(int fd, CEC_ADAP_G_PHYS_ADDR, __u16 *argp)
```

`CEC_ADAP_S_PHYS_ADDR`

```
int ioctl(int fd, CEC_ADAP_S_PHYS_ADDR, __u16 *argp)
```

Arguments

fd File descriptor returned by `open()`.

argp Pointer to the CEC address.

Description

To query the current physical address applications call `ioctl CEC_ADAP_G_PHYS_ADDR` with a pointer to a `__u16` where the driver stores the physical address.

To set a new physical address applications store the physical address in a `__u16` and call `ioctl CEC_ADAP_S_PHYS_ADDR` with a pointer to this integer. The `ioctl CEC_ADAP_S_PHYS_ADDR` is only available if `CEC_CAP_PHYS_ADDR` is set (the `ENOTTY` error code will be returned otherwise). The `ioctl CEC_ADAP_S_PHYS_ADDR` can only be called by a file descriptor in initiator mode (see [ioctls CEC_G_MODE](#) and [CEC_S_MODE](#)), if not the `EBUSY` error code will be returned.

To clear an existing physical address use `CEC_PHYS_ADDR_INVALID`. The adapter will go to the unconfigured state.

If logical address types have been defined (see [ioctl CEC_ADAP_S_LOG_ADDRS](#)), then this `ioctl` will block until all requested logical addresses have been claimed. If the file descriptor is in

non-blocking mode then it will not wait for the logical addresses to be claimed, instead it just returns 0.

A `CEC_EVENT_STATE_CHANGE` event is sent when the physical address changes.

The physical address is a 16-bit number where each group of 4 bits represent a digit of the physical address a.b.c.d where the most significant 4 bits represent 'a'. The CEC root device (usually the TV) has address 0.0.0.0. Every device that is hooked up to an input of the TV has address a.0.0.0 (where 'a' is ≥ 1), devices hooked up to those in turn have addresses a.b.0.0, etc. So a topology of up to 5 devices deep is supported. The physical address a device shall use is stored in the EDID of the sink.

For example, the EDID for each HDMI input of the TV will have a different physical address of the form a.0.0.0 that the sources will read out and use as their physical address.

Return Value

On success 0 is returned, on error -1 and the `errno` variable is set appropriately. The generic error codes are described at the [Generic Error Codes](#) chapter.

The `ioctl CEC_ADAP_S_PHYS_ADDR` can return the following error codes:

ENOTTY The `CEC_CAP_PHYS_ADDR` capability wasn't set, so this `ioctl` is not supported.

EBUSY Another filehandle is in exclusive follower or initiator mode, or the filehandle is in mode `CEC_MODE_NO_INITIATOR`.

EINVAL The physical address is malformed.

3.6.2.8 `ioctl CEC_ADAP_G_CONNECTOR_INFO`

Name

`CEC_ADAP_G_CONNECTOR_INFO` - Query HDMI connector information

Synopsis

`CEC_ADAP_G_CONNECTOR_INFO`

```
int ioctl(int fd, CEC_ADAP_G_CONNECTOR_INFO, struct cec_connector_info *argp)
```

Arguments

fd File descriptor returned by `open()`.

argp

Description

Using this ioctl an application can learn which HDMI connector this CEC device corresponds to. While calling this ioctl the application should provide a pointer to a `cec_connector_info` struct which will be populated by the kernel with the info provided by the adapter's driver. This ioctl is only available if the `CEC_CAP_CONNECTOR_INFO` capability is set.

`cec_connector_info`

Table 285: struct `cec_connector_info`

<code>__u32</code>	<code>type</code>	The type of connector this adapter is associated with.
	union (anonymous) {	
	struct <code>drm_cec_connector_info</code>	<i>struct <code>cec_drm_connector_info</code></i>
	}	

Table 286: Connector types

<code>CEC_CONNECTOR_TYPE_NO_CONNECTOR</code>	No connector is associated with the adapter/the information is not provided by the driver.
<code>CEC_CONNECTOR_TYPE_DRM</code>	Indicates that a DRM connector is associated with this adapter. Information about the connector can be found in <i>struct <code>cec_drm_connector_info</code></i> .

`cec_drm_connector_info`

Table 287: struct `cec_drm_connector_info`

<code>__u32</code>	<code>card_no</code>	DRM card number: the number from a card's path, e.g. 0 in case of <code>/dev/card0</code> .
<code>__u32</code>	<code>connector_id</code>	DRM connector ID.

3.6.2.9 ioctl `CEC_DQEVENT`

Name

`CEC_DQEVENT` - Dequeue a CEC event

Synopsis

CEC_DQEVENT

```
int ioctl(int fd, CEC_DQEVENT, struct cec_event *argp)
```

Arguments

fd File descriptor returned by `open()`.

argp

Description

CEC devices can send asynchronous events. These can be retrieved by calling `CEC_DQEVENT()`. If the file descriptor is in non-blocking mode and no event is pending, then it will return -1 and set `errno` to the `EAGAIN` error code.

The internal event queues are per-filehandle and per-event type. If there is no more room in a queue then the last event is overwritten with the new one. This means that intermediate results can be thrown away but that the latest event is always available. This also means that it is possible to read two successive events that have the same value (e.g. two `CEC_EVENT_STATE_CHANGE` events with the same state). In that case the intermediate state changes were lost but it is guaranteed that the state did change in between the two events.

`cec_event_state_change`

Table 288: struct `cec_event_state_change`

<code>__u16</code>	<code>phys_addr</code>	The current physical address. This is <code>CEC_PHYS_ADDR_INVALID</code> if no valid physical address is set.
<code>__u16</code>	<code>log_addr_mask</code>	The current set of claimed logical addresses. This is 0 if no logical addresses are claimed or if <code>phys_addr</code> is <code>CEC_PHYS_ADDR_INVALID</code> . If bit 15 is set (<code>1 << CEC_LOG_ADDR_UNREGISTERED</code>) then this device has the unregistered logical address. In that case all other bits are 0.
<code>__u16</code>	<code>have_conn_info</code>	If non-zero, then HDMI connector information is available. This field is only valid if <code>CEC_CAP_CONNECTOR_INFO</code> is set. If that capability is set and <code>have_conn_info</code> is zero, then that indicates that the HDMI connector device is not instantiated, either because the HDMI driver is still configuring the device or because the HDMI device was unbound.

`cec_event_lost_msgs`

Table 289: struct cec_event_lost_msgs

__u32	lost_msgs	Set to the number of lost messages since the filehandle was opened or since the last time this event was dequeued for this filehandle. The messages lost are the oldest messages. So when a new message arrives and there is no more room, then the oldest message is discarded to make room for the new one. The internal size of the message queue guarantees that all messages received in the last two seconds will be stored. Since messages should be replied to within a second according to the CEC specification, this is more than enough.
-------	-----------	--

cec_event

Table 290: struct cec_event

__u64	ts	Timestamp of the event in ns. The timestamp has been taken from the CLOCK_MONOTONIC clock. To access the same clock from userspace use clock_gettime().	
__u32	event	The CEC event type, see CEC Events Types .	
__u32	flags	Event flags, see CEC Event Flags .	
union (anonymous) {			
struct	state_change cec_event_state_change	The new adapter state as sent by the CEC_EVENT_STATE_CHANGE event.	
struct	lost_msgs cec_event_lost_msgs	The number of lost messages as sent by the CEC_EVENT_LOST_MSGS event.	
}			

Table 291: CEC Events Types

CEC_EVENT_STATE_CHANGE	1	Generated when the CEC Adapter's state changes. When open() is called an initial event will be generated for that filehandle with the CEC Adapter's state at that time.
CEC_EVENT_LOST_MSGS	2	Generated if one or more CEC messages were lost because the application didn't dequeue CEC messages fast enough.
CEC_EVENT_PIN_CEC_LOW	3	Generated if the CEC pin goes from a high voltage to a low voltage. Only applies to adapters that have the CEC_CAP_MONITOR_PIN capability set.
CEC_EVENT_PIN_CEC_HIGH	4	Generated if the CEC pin goes from a low voltage to a high voltage. Only applies to adapters that have the CEC_CAP_MONITOR_PIN capability set.
CEC_EVENT_PIN_HPD_LOW	5	Generated if the HPD pin goes from a high voltage to a low voltage. Only applies to adapters that have the CEC_CAP_MONITOR_PIN capability set. When open() is called, the HPD pin can be read and if the HPD is low, then an initial event will be generated for that filehandle.
CEC_EVENT_PIN_HPD_HIGH	6	Generated if the HPD pin goes from a low voltage to a high voltage. Only applies to adapters that have the CEC_CAP_MONITOR_PIN capability set. When open() is called, the HPD pin can be read and if the HPD is high, then an initial event will be generated for that filehandle.
CEC_EVENT_PIN_5V_LOW	6	Generated if the 5V pin goes from a high voltage to a low voltage. Only applies to adapters that have the CEC_CAP_MONITOR_PIN capability set. When open() is called, the 5V pin can be read and if the 5V is low, then an initial event will be generated for that filehandle.
CEC_EVENT_PIN_5V_HIGH	7	Generated if the 5V pin goes from a low voltage to a high voltage. Only applies to adapters that have the CEC_CAP_MONITOR_PIN capability set. When open() is called, the 5V pin can be read and if the 5V is high, then an initial event will be generated for that filehandle.

Table 292: CEC Event Flags

CEC_EVENT_FL_INITIAL_STATE	1	Set for the initial events that are generated when the device is opened. See the table above for which events do this. This allows applications to learn the initial state of the CEC adapter at open() time.
CEC_EVENT_FL_DROPPED_EVENTS	2	Set if one or more events of the given event type have been dropped. This is an indication that the application cannot keep up.

Return Value

On success 0 is returned, on error -1 and the `errno` variable is set appropriately. The generic error codes are described at the [Generic Error Codes](#) chapter.

The `ioctl CEC_DQEVENT` can return the following error codes:

EAGAIN This is returned when the filehandle is in non-blocking mode and there are no pending events.

ERESTARTSYS An interrupt (e.g. Ctrl-C) arrived while in blocking mode waiting for events to arrive.

3.6.2.10 `ioctls CEC_G_MODE` and `CEC_S_MODE`

`CEC_G_MODE`, `CEC_S_MODE` - Get or set exclusive use of the CEC adapter

Synopsis

`CEC_G_MODE`

```
int ioctl(int fd, CEC_G_MODE, __u32 *argp)
```

`CEC_S_MODE`

```
int ioctl(int fd, CEC_S_MODE, __u32 *argp)
```

Arguments

fd File descriptor returned by `open()`.

argp Pointer to CEC mode.

Description

By default any filehandle can use *ioctl CEC_RECEIVE and CEC_TRANSMIT*, but in order to prevent applications from stepping on each others toes it must be possible to obtain exclusive access to the CEC adapter. This ioctl sets the filehandle to initiator and/or follower mode which can be exclusive depending on the chosen mode. The initiator is the filehandle that is used to initiate messages, i.e. it commands other CEC devices. The follower is the filehandle that receives messages sent to the CEC adapter and processes them. The same filehandle can be both initiator and follower, or this role can be taken by two different filehandles.

When a CEC message is received, then the CEC framework will decide how it will be processed. If the message is a reply to an earlier transmitted message, then the reply is sent back to the filehandle that is waiting for it. In addition the CEC framework will process it.

If the message is not a reply, then the CEC framework will process it first. If there is no follower, then the message is just discarded and a feature abort is sent back to the initiator if the framework couldn't process it. If there is a follower, then the message is passed on to the follower who will use *ioctl CEC_RECEIVE* to dequeue the new message. The framework expects the follower to make the right decisions.

The CEC framework will process core messages unless requested otherwise by the follower. The follower can enable the passthrough mode. In that case, the CEC framework will pass on most core messages without processing them and the follower will have to implement those messages. There are some messages that the core will always process, regardless of the passthrough mode. See *Core Message Processing* for details.

If there is no initiator, then any CEC filehandle can use *ioctl CEC_TRANSMIT*. If there is an exclusive initiator then only that initiator can call *ioctl CEC_RECEIVE and CEC_TRANSMIT*. The follower can of course always call *ioctl CEC_TRANSMIT*.

Available initiator modes are:

Table 293: Initiator Modes

CEC_MODE_NO_INITIATOR	0x0	This is not an initiator, i.e. it cannot transmit CEC messages or make any other changes to the CEC adapter.
CEC_MODE_INITIATOR	0x1	This is an initiator (the default when the device is opened) and it can transmit CEC messages and make changes to the CEC adapter, unless there is an exclusive initiator.
CEC_MODE_EXCL_INITIATOR	0x2	This is an exclusive initiator and this file descriptor is the only one that can transmit CEC messages and make changes to the CEC adapter. If someone else is already the exclusive initiator then an attempt to become one will return the EBUSY error code error.

Available follower modes are:

Table 294: Follower Modes

CEC_MODE_NO_FOLLOWER	0x00	This is not a follower (the default when the device is opened).
CEC_MODE_FOLLOWER	0x10	This is a follower and it will receive CEC messages unless there is an exclusive follower. You cannot become a follower if <i>CEC_CAP_TRANSMIT</i> is not set or if <i>CEC_MODE_NO_INITIATOR</i> was specified, the EINVAL error code is returned in that case.
CEC_MODE_EXCL_FOLLOWER	0x20	This is an exclusive follower and only this file descriptor will receive CEC messages for processing. If someone else is already the exclusive follower then an attempt to become one will return the EBUSY error code. You cannot become a follower if <i>CEC_CAP_TRANSMIT</i> is not set or if <i>CEC_MODE_NO_INITIATOR</i> was specified, the EINVAL error code is returned in that case.
CEC_MODE_EXCL_FOLLOWER_PASSTHRU	0x30	This is an exclusive follower and only this file descriptor will receive CEC messages for processing. In addition it will put the CEC device into passthrough mode, allowing the exclusive follower to handle most core messages instead of relying on the CEC framework for that. If someone else is already the exclusive follower then an attempt to become one will return the EBUSY error code. You cannot become a follower if <i>CEC_CAP_TRANSMIT</i> is not set or if <i>CEC_MODE_NO_INITIATOR</i> was specified, the EINVAL error code is returned in that case.
CEC_MODE_MONITOR_PIN	0xd0	Put the file descriptor into pin monitoring mode. Can only be used in combination with <i>CEC_MODE_NO_INITIATOR</i> , otherwise the EINVAL error code will be returned. This mode requires that the <i>CEC_CAP_MONITOR_PIN</i> capability is set, otherwise the EINVAL error code is returned. While in pin monitoring mode this file descriptor can receive the CEC_EVENT_PIN_CEC_LOW and CEC_EVENT_PIN_CEC_HIGH events to see the low-level CEC pin transitions. This is very useful for debugging. This mode is only allowed if the process has the CAP_NET_ADMIN capability. If that is not set, then the EPERM error code is returned.

Continued on next page

Table 294 - continued from previous page

CEC_MODE_MONITOR	0xe0	Put the file descriptor into monitor mode. Can only be used in combination with <i>CEC_MODE_NO_INITIATOR</i> , otherwise the EINVAL error code will be returned. In monitor mode all messages this CEC device transmits and all messages it receives (both broadcast messages and directed messages for one its logical addresses) will be reported. This is very useful for debugging. This is only allowed if the process has the CAP_NET_ADMIN capability. If that is not set, then the EPERM error code is returned.
CEC_MODE_MONITOR_ALL	0xf0	Put the file descriptor into 'monitor all' mode. Can only be used in combination with <i>CEC_MODE_NO_INITIATOR</i> , otherwise the EINVAL error code will be returned. In 'monitor all' mode all messages this CEC device transmits and all messages it receives, including directed messages for other CEC devices, will be reported. This is very useful for debugging, but not all devices support this. This mode requires that the <i>CEC_CAP_MONITOR_ALL</i> capability is set, otherwise the EINVAL error code is returned. This is only allowed if the process has the CAP_NET_ADMIN capability. If that is not set, then the EPERM error code is returned.

Core message processing details:

Table 295: Core Message Processing

CEC_MSG_GET_CEC_VERSION	The core will return the CEC version that was set with <i>ioctl CEC_ADAP_S_LOG_ADDRS</i> , except when in passthrough mode. In passthrough mode the core does nothing and this message has to be handled by a follower instead.
CEC_MSG_GIVE_DEVICE_VENDOR_ID	The core will return the vendor ID that was set with <i>ioctl CEC_ADAP_S_LOG_ADDRS</i> , except when in passthrough mode. In passthrough mode the core does nothing and this message has to be handled by a follower instead.
CEC_MSG_ABORT	The core will return a Feature Abort message with reason 'Feature Refused' as per the specification, except when in passthrough mode. In passthrough mode the core does nothing and this message has to be handled by a follower instead.
CEC_MSG_GIVE_PHYSICAL_ADDR	The core will report the current physical address, except when in passthrough mode. In passthrough mode the core does nothing and this message has to be handled by a follower instead.
CEC_MSG_GIVE_OSD_NAME	The core will report the current OSD name that was set with <i>ioctl CEC_ADAP_S_LOG_ADDRS</i> , except when in passthrough mode. In passthrough mode the core does nothing and this message has to be handled by a follower instead.
CEC_MSG_GIVE_FEATURES	The core will do nothing if the CEC version is older than 2.0, otherwise it will report the current features that were set with <i>ioctl CEC_ADAP_S_LOG_ADDRS</i> , except when in passthrough mode. In passthrough mode the core does nothing (for any CEC version) and this message has to be handled by a follower instead.
CEC_MSG_USER_CONTROL_PRESSED	If <i>CEC_CAP_RC</i> is set and if <i>CEC_LOG_ADDRS_FL_ALLOW_RC_PASSTHRU</i> is set, then generate a remote control key press. This message is always passed on to the follower(s).
CEC_MSG_USER_CONTROL_RELEASED	If <i>CEC_CAP_RC</i> is set and if <i>CEC_LOG_ADDRS_FL_ALLOW_RC_PASSTHRU</i> is set, then generate a remote control key release. This message is always passed on to the follower(s).
CEC_MSG_REPORT_PHYSICAL_ADDR	The CEC framework will make note of the reported physical address and then just pass the message on to the follower(s).

Return Value

On success 0 is returned, on error -1 and the `errno` variable is set appropriately. The generic error codes are described at the [Generic Error Codes](#) chapter.

The `ioctl CEC_S_MODE` can return the following error codes:

EINVAL The requested mode is invalid.

EPERM Monitor mode is requested, but the process does not have the `CAP_NET_ADMIN` capability.

EBUSY Someone else is already an exclusive follower or initiator.

3.6.2.11 `ioctls CEC_RECEIVE` and `CEC_TRANSMIT`

Name

`CEC_RECEIVE`, `CEC_TRANSMIT` - Receive or transmit a CEC message

Synopsis

`CEC_RECEIVE`

```
int ioctl(int fd, CEC_RECEIVE, struct cec_msg *argp)
```

`CEC_TRANSMIT`

```
int ioctl(int fd, CEC_TRANSMIT, struct cec_msg *argp)
```

Arguments

fd File descriptor returned by `open()`.

argp Pointer to `struct cec_msg`.

Description

To receive a CEC message the application has to fill in the `timeout` field of `struct cec_msg` and pass it to `ioctl CEC_RECEIVE`. If the file descriptor is in non-blocking mode and there are no received messages pending, then it will return -1 and set `errno` to the `EAGAIN` error code. If the file descriptor is in blocking mode and `timeout` is non-zero and no message arrived within `timeout` milliseconds, then it will return -1 and set `errno` to the `ETIMEDOUT` error code.

A received message can be:

1. a message received from another CEC device (the `sequence` field will be 0, `tx_status` will be 0 and `rx_status` will be non-zero).
2. the transmit result of an earlier non-blocking transmit (the `sequence` field will be non-zero, `tx_status` will be non-zero and `rx_status` will be 0).
3. the reply to an earlier non-blocking transmit (the `sequence` field will be non-zero, `tx_status` will be 0 and `rx_status` will be non-zero).

To send a CEC message the application has to fill in the struct `cec_msg` and pass it to `ioctl CEC_TRANSMIT`. The `ioctl CEC_TRANSMIT` is only available if `CEC_CAP_TRANSMIT` is set. If there is no more room in the transmit queue, then it will return -1 and set `errno` to the `EBUSY` error code. The transmit queue has enough room for 18 messages (about 1 second worth of 2-byte messages). Note that the CEC kernel framework will also reply to core messages (see *Core Message Processing*), so it is not a good idea to fully fill up the transmit queue.

If the file descriptor is in non-blocking mode then the transmit will return 0 and the result of the transmit will be available via `ioctl CEC_RECEIVE` once the transmit has finished. If a non-blocking transmit also specified waiting for a reply, then the reply will arrive in a later message. The sequence field can be used to associate both transmit results and replies with the original transmit.

Normally calling `ioctl CEC_TRANSMIT` when the physical address is invalid (due to e.g. a disconnect) will return `ENONET`.

However, the CEC specification allows sending messages from 'Unregistered' to 'TV' when the physical address is invalid since some TVs pull the hotplug detect pin of the HDMI connector low when they go into standby, or when switching to another input.

When the hotplug detect pin goes low the EDID disappears, and thus the physical address, but the cable is still connected and CEC still works. In order to detect/wake up the device it is allowed to send poll and 'Image/Text View On' messages from initiator 0xf ('Unregistered') to destination 0 ('TV').

cec_msg

Table 296: struct `cec_msg`

__u64	tx_ts	Timestamp in ns of when the last byte of the message was transmitted. The timestamp has been taken from the <code>CLOCK_MONOTONIC</code> clock. To access the same clock from userspace use <code>clock_gettime()</code> .
__u64	rx_ts	Timestamp in ns of when the last byte of the message was received. The timestamp has been taken from the <code>CLOCK_MONOTONIC</code> clock. To access the same clock from userspace use <code>clock_gettime()</code> .
__u32	len	The length of the message. For <code>ioctl CEC_TRANSMIT</code> this is filled in by the application. The driver will fill this in for <code>ioctl CEC_RECEIVE</code> . For <code>ioctl CEC_TRANSMIT</code> it will be filled in by the driver with the length of the reply message if reply was set.
__u32	timeout	The timeout in milliseconds. This is the time the device will wait for a message to be received before timing out. If it is set to 0, then it will wait indefinitely when it is called by <code>ioctl CEC_RECEIVE</code> . If it is 0 and it is called by <code>ioctl CEC_TRANSMIT</code> , then it will be replaced by 1000 if the reply is non-zero or ignored if reply is 0.

Continued on next page

Table 296 - continued from previous page

__u32	sequence	<p>A non-zero sequence number is automatically assigned by the CEC framework for all transmitted messages. It is used by the CEC framework when it queues the transmit result for a non-blocking transmit. This allows the application to associate the received message with the original transmit.</p> <p>In addition, if a non-blocking transmit will wait for a reply (i.e. timeout was not 0), then the sequence field of the reply will be set to the sequence value of the original transmit. This allows the application to associate the received message with the original transmit.</p>
__u32	flags	Flags. See <i>Flags for struct cec_msg</i> for a list of available flags.
__u8	msg[16]	The message payload. For <i>ioctl CEC_TRANSMIT</i> this is filled in by the application. The driver will fill this in for <i>ioctl CEC_RECEIVE</i> . For <i>ioctl CEC_TRANSMIT</i> it will be filled in by the driver with the payload of the reply message if timeout was set.
__u8	reply	<p>Wait until this message is replied. If reply is 0 and the timeout is 0, then don't wait for a reply but return after transmitting the message. Ignored by <i>ioctl CEC_RECEIVE</i>. The case where reply is 0 (this is the opcode for the Feature Abort message) and timeout is non-zero is specifically allowed to make it possible to send a message and wait up to timeout milliseconds for a Feature Abort reply. In this case rx_status will either be set to <i>CEC_RX_STATUS_TIMEOUT</i> or <i>CEC_RX_STATUS_FEATURE_ABORT</i>.</p> <p>If the transmitter message is CEC_MSG_INITIATE_ARC then the reply values CEC_MSG_REPORT_ARC_INITIATED and CEC_MSG_REPORT_ARC_TERMINATED are processed differently: either value will match both possible replies. The reason is that the CEC_MSG_INITIATE_ARC message is the only CEC message that has two possible replies other than Feature Abort. The reply field will be updated with the actual reply so that it is synchronized with the contents of the received message.</p>
__u8	rx_status	The status bits of the received message. See <i>CEC Receive Status</i> for the possible status values.
__u8	tx_status	The status bits of the transmitted message. See <i>CEC Transmit Status</i> for the possible status values. When calling <i>ioctl CEC_TRANSMIT</i> in non-blocking mode, this field will be 0 if the transmit started, or non-0 if the transmit result is known immediately. The latter would be the case when attempting to transmit a Poll message to yourself. That results in a <i>CEC_TX_STATUS_NACK</i> without ever actually transmitting the Poll message.
__u8	tx_arb_lost_cnt	A counter of the number of transmit attempts that resulted in the Arbitration Lost error. This is only set if the hardware supports this, otherwise it is always 0. This counter is only valid if the <i>CEC_TX_STATUS_ARB_LOST</i> status bit is set.

Continued on next page

Table 296 - continued from previous page

__u8	tx_nack_cnt	A counter of the number of transmit attempts that resulted in the Not Acknowledged error. This is only set if the hardware supports this, otherwise it is always 0. This counter is only valid if the <i>CEC_TX_STATUS_NACK</i> status bit is set.
__u8	tx_low_drive_cnt	A counter of the number of transmit attempts that resulted in the Arbitration Lost error. This is only set if the hardware supports this, otherwise it is always 0. This counter is only valid if the <i>CEC_TX_STATUS_LOW_DRIVE</i> status bit is set.
__u8	tx_error_cnt	A counter of the number of transmit errors other than Arbitration Lost or Not Acknowledged. This is only set if the hardware supports this, otherwise it is always 0. This counter is only valid if the <i>CEC_TX_STATUS_ERROR</i> status bit is set.

Table 297: Flags for struct cec_msg

CEC_MSG_FL_REPLY_TO_FOLLOWERS	1	If a CEC transmit expects a reply, then by default that reply is only sent to the filehandle that called <i>ioctl CEC_TRANSMIT</i> . If this flag is set, then the reply is also sent to all followers, if any. If the filehandle that called <i>ioctl CEC_TRANSMIT</i> is also a follower, then that filehandle will receive the reply twice: once as the result of the <i>ioctl CEC_TRANSMIT</i> , and once via <i>ioctl CEC_RECEIVE</i> .
CEC_MSG_FL_RAW	2	Normally CEC messages are validated before transmitting them. If this flag is set when <i>ioctl CEC_TRANSMIT</i> is called, then no validation takes place and the message is transmitted as-is. This is useful when debugging CEC issues. This flag is only allowed if the process has the <i>CAP_SYS_RAWIO</i> capability. If that is not set, then the <i>EPERM</i> error code is returned.

Table 298: CEC Transmit Status

CEC_TX_STATUS_OK	0x01	The message was transmitted successfully. This is mutually exclusive with <i>CEC_TX_STATUS_MAX_RETRIES</i> . Other bits can still be set if earlier attempts met with failure before the transmit was eventually successful.
CEC_TX_STATUS_ARB_LOST	0x02	CEC line arbitration was lost, i.e. another transmit started at the same time with a higher priority. Optional status, not all hardware can detect this error condition.
CEC_TX_STATUS_NACK	0x04	Message was not acknowledged. Note that some hardware cannot tell apart a 'Not Acknowledged' status from other error conditions, i.e. the result of a transmit is just OK or FAIL. In that case this status will be returned when the transmit failed.
CEC_TX_STATUS_LOW_DRIVE	0x08	Low drive was detected on the CEC bus. This indicates that a follower detected an error on the bus and requests a retransmission. Optional status, not all hardware can detect this error condition.
CEC_TX_STATUS_ERROR	0x10	Some error occurred. This is used for any errors that do not fit CEC_TX_STATUS_ARB_LOST or CEC_TX_STATUS_LOW_DRIVE, either because the hardware could not tell which error occurred, or because the hardware tested for other conditions besides those two. Optional status.
CEC_TX_STATUS_MAX_RETRIES	0x20	The transmit failed after one or more retries. This status bit is mutually exclusive with <i>CEC_TX_STATUS_OK</i> . Other bits can still be set to explain which failures were seen.
CEC_TX_STATUS_ABORTED	0x40	The transmit was aborted due to an HDMI disconnect, or the adapter was unconfigured, or a transmit was interrupted, or the driver returned an error when attempting to start a transmit.
CEC_TX_STATUS_TIMEOUT	0x80	The transmit timed out. This should not normally happen and this indicates a driver problem.

Table 299: CEC Receive Status

CEC_RX_STATUS_OK	0x01	The message was received successfully.
CEC_RX_STATUS_TIMEOUT	0x02	The reply to an earlier transmitted message timed out.
CEC_RX_STATUS_FEATURE_ABORTED	0x04	The message was received successfully but the reply was CEC_MSG_FEATURE_ABORT. This status is only set if this message was the reply to an earlier transmitted message.
CEC_RX_STATUS_ABORTED	0x08	The wait for a reply to an earlier transmitted message was aborted because the HDMI cable was disconnected, the adapter was unconfigured or the CEC_TRANSMIT that waited for a reply was interrupted.

Return Value

On success 0 is returned, on error -1 and the `errno` variable is set appropriately. The generic error codes are described at the [Generic Error Codes](#) chapter.

The `ioctl CEC_RECEIVE` can return the following error codes:

EAGAIN No messages are in the receive queue, and the filehandle is in non-blocking mode.

ETIMEDOUT The timeout was reached while waiting for a message.

ERESTARTSYS The wait for a message was interrupted (e.g. by Ctrl-C).

The `ioctl CEC_TRANSMIT` can return the following error codes:

ENOTTY The `CEC_CAP_TRANSMIT` capability wasn't set, so this `ioctl` is not supported.

EPERM The CEC adapter is not configured, i.e. `ioctl CEC_ADAP_S_LOG_ADDRS` has never been called, or `CEC_MSG_FL_RAW` was used from a process that did not have the `CAP_SYS_RAWIO` capability.

ENONET The CEC adapter is not configured, i.e. `ioctl CEC_ADAP_S_LOG_ADDRS` was called, but the physical address is invalid so no logical address was claimed. An exception is made in this case for transmits from initiator 0xf ('Unregistered') to destination 0 ('TV'). In that case the transmit will proceed as usual.

EBUSY Another filehandle is in exclusive follower or initiator mode, or the filehandle is in mode `CEC_MODE_NO_INITIATOR`. This is also returned if the transmit queue is full.

EINVAL The contents of struct `cec_msg` is invalid.

ERESTARTSYS The wait for a successful transmit was interrupted (e.g. by Ctrl-C).

3.6.3 CEC Pin Framework Error Injection

The CEC Pin Framework is a core CEC framework for CEC hardware that only has low-level support for the CEC bus. Most hardware today will have high-level CEC support where the hardware deals with driving the CEC bus, but some older devices aren't that fancy. However, this framework also allows you to connect the CEC pin to a GPIO on e.g. a Raspberry Pi and you have now made a CEC adapter.

What makes doing this so interesting is that since we have full control over the bus it is easy to support error injection. This is ideal to test how well CEC adapters can handle error conditions.

Currently only the cec-gpio driver (when the CEC line is directly connected to a pull-up GPIO line) and the AllWinner A10/A20 drm driver support this framework.

If CONFIG_CEC_PIN_ERROR_INJ is enabled, then error injection is available through debugfs. Specifically, in /sys/kernel/debug/cec/cecX/ there is now an error-inj file.

Note: The error injection commands are not a stable ABI and may change in the future.

With `cat error-inj` you can see both the possible commands and the current error injection status:

```
$ cat /sys/kernel/debug/cec/cec0/error-inj
# Clear error injections:
# clear          clear all rx and tx error injections
# rx-clear       clear all rx error injections
# tx-clear       clear all tx error injections
# <op> clear     clear all rx and tx error injections for <op>
# <op> rx-clear  clear all rx error injections for <op>
# <op> tx-clear  clear all tx error injections for <op>
#
# RX error injection:
# <op>[,<mode>] rx-nack          NACK the message instead of sending an ACK
# <op>[,<mode>] rx-low-drive <bit> force a low-drive condition at this bit position
# <op>[,<mode>] rx-add-byte      add a spurious byte to the received CEC message
# <op>[,<mode>] rx-remove-byte   remove the last byte from the received CEC
↳message
# any[,<mode>] rx-arb-lost [<poll>] generate a POLL message to trigger an
↳arbitration lost
#
# TX error injection settings:
# tx-ignore-nack-until-eom       ignore early NACKs until EOM
# tx-custom-low-usecs <usecs>   define the 'low' time for the custom pulse
# tx-custom-high-usecs <usecs>  define the 'high' time for the custom pulse
# tx-custom-pulse                transmit the custom pulse once the bus is idle
#
# TX error injection:
# <op>[,<mode>] tx-no-eom        don't set the EOM bit
# <op>[,<mode>] tx-early-eom     set the EOM bit one byte too soon
# <op>[,<mode>] tx-add-bytes <num> append <num> (1-255) spurious bytes to the
↳message
# <op>[,<mode>] tx-remove-byte   drop the last byte from the message
# <op>[,<mode>] tx-short-bit <bit> make this bit shorter than allowed
# <op>[,<mode>] tx-long-bit <bit>  make this bit longer than allowed
# <op>[,<mode>] tx-custom-bit <bit> send the custom pulse instead of this bit
# <op>[,<mode>] tx-short-start   send a start pulse that's too short
```

```
# <op>[,<mode>] tx-long-start      send a start pulse that's too long
# <op>[,<mode>] tx-custom-start    send the custom pulse instead of the start pulse
# <op>[,<mode>] tx-last-bit <bit>  stop sending after this bit
# <op>[,<mode>] tx-low-drive <bit> force a low-drive condition at this bit position
#
# <op>          CEC message opcode (0-255) or 'any'
# <mode>        'once' (default), 'always', 'toggle' or 'off'
# <bit>         CEC message bit (0-159)
#              10 bits per 'byte': bits 0-7: data, bit 8: EOM, bit 9: ACK
# <poll>        CEC poll message used to test arbitration lost (0x00-0xff, default 0x0f)
# <usecs>       microseconds (0-10000000, default 1000)

clear
```

You can write error injection commands to `error-inj` using `echo 'cmd' >error-inj` or `cat cmd.txt >error-inj`. The `cat error-inj` output contains the current error commands. You can save the output to a file and use it as an input to `error-inj` later.

3.6.3.1 Basic Syntax

Leading spaces/tabs are ignored. If the next character is a `#` or the end of the line was reached, then the whole line is ignored. Otherwise a command is expected.

The error injection commands fall in two main groups: those relating to receiving CEC messages and those relating to transmitting CEC messages. In addition, there are commands to clear existing error injection commands and to create custom pulses on the CEC bus.

Most error injection commands can be executed for specific CEC opcodes or for all opcodes (any). Each command also has a 'mode' which can be `off` (can be used to turn off an existing error injection command), `once` (the default) which will trigger the error injection only once for the next received or transmitted message, `always` to always trigger the error injection and `toggle` to toggle the error injection on or off for every transmit or receive.

So `'any rx-nack'` will NACK the next received CEC message, `'any,always rx-nack'` will NACK all received CEC messages and `'0x82,toggle rx-nack'` will only NACK if an Active Source message was received and do that only for every other received message.

After an error was injected with mode `once` the error injection command is cleared automatically, so `once` is a one-time deal.

All combinations of `<op>` and error injection commands can co-exist. So this is fine:

```
0x9e tx-add-bytes 1
0x9e tx-early-eom
0x9f tx-add-bytes 2
any rx-nack
```

All four error injection commands will be active simultaneously.

However, if the same `<op>` and command combination is specified, but with different arguments:

```
0x9e tx-add-bytes 1
0x9e tx-add-bytes 2
```

Then the second will overwrite the first.

3.6.3.2 Clear Error Injections

clear Clear all error injections.

rx-clear Clear all receive error injections

tx-clear Clear all transmit error injections

<op> clear Clear all error injections for the given opcode.

<op> rx-clear Clear all receive error injections for the given opcode.

<op> tx-clear Clear all transmit error injections for the given opcode.

3.6.3.3 Receive Messages

<op>[,<mode>] rx-nack NACK broadcast messages and messages directed to this CEC adapter. Every byte of the message will be NACKed in case the transmitter keeps transmitting after the first byte was NACKed.

<op>[,<mode>] rx-low-drive <bit> Force a Low Drive condition at this bit position. If **<op>** specifies a specific CEC opcode then the bit position must be at least 18, otherwise the opcode hasn't been received yet. This tests if the transmitter can handle the Low Drive condition correctly and reports the error correctly. Note that a Low Drive in the first 4 bits can also be interpreted as an Arbitration Lost condition by the transmitter. This is implementation dependent.

<op>[,<mode>] rx-add-byte Add a spurious 0x55 byte to the received CEC message, provided the message was 15 bytes long or less. This is useful to test the high-level protocol since spurious bytes should be ignored.

<op>[,<mode>] rx-remove-byte Remove the last byte from the received CEC message, provided it was at least 2 bytes long. This is useful to test the high-level protocol since messages that are too short should be ignored.

<op>[,<mode>] rx-arb-lost <poll> Generate a POLL message to trigger an Arbitration Lost condition. This command is only allowed for **<op>** values of `next` or `all`. As soon as a start bit has been received the CEC adapter will switch to transmit mode and it will transmit a POLL message. By default this is 0x0f, but it can also be specified explicitly via the **<poll>** argument.

This command can be used to test the Arbitration Lost condition in the remote CEC transmitter. Arbitration happens when two CEC adapters start sending a message at the same time. In that case the initiator with the most leading zeroes wins and the other transmitter has to stop transmitting ('Arbitration Lost'). This is very hard to test, except by using this error injection command.

This does not work if the remote CEC transmitter has logical address 0 ('TV') since that will always win.

3.6.3.4 Transmit Messages

tx-ignore-nack-until-eom This setting changes the behavior of transmitting CEC messages. Normally as soon as the receiver NACKs a byte the transmit will stop, but the specification also allows that the full message is transmitted and only at the end will the transmitter look at the ACK bit. This is not recommended behavior since there is no point in keeping the CEC bus busy for longer than is strictly needed. Especially given how slow the bus is.

This setting can be used to test how well a receiver deals with transmitters that ignore NACKs until the very end of the message.

<op>[,<mode>] tx-no-eom Don't set the EOM bit. Normally the last byte of the message has the EOM (End-Of-Message) bit set. With this command the transmit will just stop without ever sending an EOM. This can be used to test how a receiver handles this case. Normally receivers have a time-out after which they will go back to the Idle state.

<op>[,<mode>] tx-early-eom Set the EOM bit one byte too soon. This obviously only works for messages of two bytes or more. The EOM bit will be set for the second-to-last byte and not for the final byte. The receiver should ignore the last byte in this case. Since the resulting message is likely to be too short for this same reason the whole message is typically ignored. The receiver should be in Idle state after the last byte was transmitted.

<op>[,<mode>] tx-add-bytes <num> Append <num> (1-255) spurious bytes to the message. The extra bytes have the value of the byte position in the message. So if you transmit a two byte message (e.g. a Get CEC Version message) and add 2 bytes, then the full message received by the remote CEC adapter is 0x40 0x9f 0x02 0x03.

This command can be used to test buffer overflows in the receiver. E.g. what does it do when it receives more than the maximum message size of 16 bytes.

<op>[,<mode>] tx-remove-byte Drop the last byte from the message, provided the message is at least two bytes long. The receiver should ignore messages that are too short.

<op>[,<mode>] tx-short-bit <bit> Make this bit period shorter than allowed. The bit position cannot be an Ack bit. If <op> specifies a specific CEC opcode then the bit position must be at least 18, otherwise the opcode hasn't been received yet. Normally the period of a data bit is between 2.05 and 2.75 milliseconds. With this command the period of this bit is 1.8 milliseconds, this is done by reducing the time the CEC bus is high. This bit period is less than is allowed and the receiver should respond with a Low Drive condition.

This command is ignored for 0 bits in bit positions 0 to 3. This is because the receiver also looks for an Arbitration Lost condition in those first four bits and it is undefined what will happen if it sees a too-short 0 bit.

<op>[,<mode>] tx-long-bit <bit> Make this bit period longer than is valid. The bit position cannot be an Ack bit. If <op> specifies a specific CEC opcode then the bit position must be at least 18, otherwise the opcode hasn't been received yet. Normally the period of a data bit is between 2.05 and 2.75 milliseconds. With this command the period of this bit is 2.9 milliseconds, this is done by increasing the time the CEC bus is high.

Even though this bit period is longer than is valid it is undefined what a receiver will do. It might just accept it, or it might time out and return to Idle state. Unfortunately the CEC specification is silent about this.

This command is ignored for 0 bits in bit positions 0 to 3. This is because the receiver also looks for an Arbitration Lost condition in those first four bits and it is undefined what will happen if it sees a too-long 0 bit.

<op>[,<mode>] tx-short-start Make this start bit period shorter than allowed. Normally the period of a start bit is between 4.3 and 4.7 milliseconds. With this command the period of the start bit is 4.1 milliseconds, this is done by reducing the time the CEC bus is high. This start bit period is less than is allowed and the receiver should return to Idle state when this is detected.

<op>[,<mode>] tx-long-start Make this start bit period longer than is valid. Normally the period of a start bit is between 4.3 and 4.7 milliseconds. With this command the period of the start bit is 5 milliseconds, this is done by increasing the time the CEC bus is high. This start bit period is more than is valid and the receiver should return to Idle state when this is detected.

Even though this start bit period is longer than is valid it is undefined what a receiver will do. It might just accept it, or it might time out and return to Idle state. Unfortunately the CEC specification is silent about this.

<op>[,<mode>] tx-last-bit <bit> Just stop transmitting after this bit. If <op> specifies a specific CEC opcode then the bit position must be at least 18, otherwise the opcode hasn't been received yet. This command can be used to test how the receiver reacts when a message just suddenly stops. It should time out and go back to Idle state.

<op>[,<mode>] tx-low-drive <bit> Force a Low Drive condition at this bit position. If <op> specifies a specific CEC opcode then the bit position must be at least 18, otherwise the opcode hasn't been received yet. This can be used to test how the receiver handles Low Drive conditions. Note that if this happens at bit positions 0-3 the receiver can interpret this as an Arbitration Lost condition. This is implementation dependent.

3.6.3.5 Custom Pulses

tx-custom-low-usecs <usecs> This defines the duration in microseconds that the custom pulse pulls the CEC line low. The default is 1000 microseconds.

tx-custom-high-usecs <usecs> This defines the duration in microseconds that the custom pulse keeps the CEC line high (unless another CEC adapter pulls it low in that time). The default is 1000 microseconds. The total period of the custom pulse is tx-custom-low-usecs + tx-custom-high-usecs.

<op>[,<mode>] tx-custom-bit <bit> Send the custom bit instead of a regular data bit. The bit position cannot be an Ack bit. If <op> specifies a specific CEC opcode then the bit position must be at least 18, otherwise the opcode hasn't been received yet.

<op>[,<mode>] tx-custom-start Send the custom bit instead of a regular start bit.

tx-custom-pulse Transmit a single custom pulse as soon as the CEC bus is idle.

3.6.4 CEC Header File

3.6.4.1 cec.h

```
/* SPDX-License-Identifier: ((GPL-2.0 WITH Linux-syscall-note) OR
↳BSD-3-Clause) */
/*
 * cec - HDMI Consumer Electronics Control public header
 *
 * Copyright 2016 Cisco Systems, Inc. and/or its affiliates. All rights
↳reserved.
 */

#ifndef _CEC_UAPI_H
#define _CEC_UAPI_H

#include <linux/types.h>
#include <linux/string.h>

#define CEC_MAX_MSG_SIZE          16

/**
 * struct cec_msg - CEC message structure.
 * @tx_ts:      Timestamp in nanoseconds using CLOCK_MONOTONIC. Set by the
 *             driver when the message transmission has finished.
 * @rx_ts:      Timestamp in nanoseconds using CLOCK_MONOTONIC. Set by the
 *             driver when the message was received.
 * @len:        Length in bytes of the message.
 * @timeout:    The timeout (in ms) that is used to timeout CEC_RECEIVE.
 *             Set to 0 if you want to wait forever. This timeout can also be
 *             used with CEC_TRANSMIT as the timeout for waiting for a reply.
 *             If 0, then it will use a 1 second timeout instead of waiting
 *             forever as is done with CEC_RECEIVE.
 * @sequence:   The framework assigns a sequence number to messages that are
 *             sent. This can be used to track replies to previously sent
 *             messages.
 * @flags:      Set to 0.
 * @msg:        The message payload.
 * @reply:      This field is ignored with CEC_RECEIVE and is only used by
 *             CEC_TRANSMIT. If non-zero, then wait for a reply with this
 *             opcode. Set to CEC_MSG_FEATURE_ABORT if you want to wait for
 *             a possible ABORT reply. If there was an error when sending the
 *             msg or FeatureAbort was returned, then reply is set to 0.
 *             If reply is non-zero upon return, then len/msg are set to
 *             the received message.
 *             If reply is zero upon return and status has the
 *             CEC_TX_STATUS_FEATURE_ABORT bit set, then len/msg are set to
 *             the received feature abort message.
 *             If reply is zero upon return and status has the
 *             CEC_TX_STATUS_MAX_RETRIES bit set, then no reply was seen at
 *             all. If reply is non-zero for CEC_TRANSMIT and the message is a
```



```

*         broadcast, then -EINVAL is returned.
*         if reply is non-zero, then timeout is set to 1000 (the required
*         maximum response time).
* @rx_status: The message receive status bits. Set by the driver.
* @tx_status: The message transmit status bits. Set by the driver.
* @tx_arb_lost_cnt: The number of 'Arbitration Lost' events. Set by the
->driver.
* @tx_ack_cnt: The number of 'Not Acknowledged' events. Set by the driver.
* @tx_low_drive_cnt: The number of 'Low Drive Detected' events. Set by the
*         driver.
* @tx_error_cnt: The number of 'Error' events. Set by the driver.
*/
struct cec_msg {
    __u64 tx_ts;
    __u64 rx_ts;
    __u32 len;
    __u32 timeout;
    __u32 sequence;
    __u32 flags;
    __u8 msg[CEC_MAX_MSG_SIZE];
    __u8 reply;
    __u8 rx_status;
    __u8 tx_status;
    __u8 tx_arb_lost_cnt;
    __u8 tx_ack_cnt;
    __u8 tx_low_drive_cnt;
    __u8 tx_error_cnt;
};

/**
 * cec_msg_initiator - return the initiator's logical address.
 * @msg:             the message structure
 */
static inline __u8 cec_msg_initiator(const struct cec_msg *msg)
{
    return msg->msg[0] >> 4;
}

/**
 * cec_msg_destination - return the destination's logical address.
 * @msg:             the message structure
 */
static inline __u8 cec_msg_destination(const struct cec_msg *msg)
{
    return msg->msg[0] & 0xf;
}

/**
 * cec_msg_opcode - return the opcode of the message, -1 for poll
 * @msg:             the message structure
 */

```

```
static inline int cec_msg_opcode(const struct cec_msg *msg)
{
    return msg->len > 1 ? msg->msg[1] : -1;
}

/**
 * cec_msg_is_broadcast - return true if this is a broadcast message.
 * @msg:                the message structure
 */
static inline int cec_msg_is_broadcast(const struct cec_msg *msg)
{
    return (msg->msg[0] & 0xf) == 0xf;
}

/**
 * cec_msg_init - initialize the message structure.
 * @msg:                the message structure
 * @initiator:          the logical address of the initiator
 * @destination:        the logical address of the destination (0xf for broadcast)
 *
 * The whole structure is zeroed, the len field is set to 1 (i.e. a poll
 * message) and the initiator and destination are filled in.
 */
static inline void cec_msg_init(struct cec_msg *msg,
                               __u8 initiator, __u8 destination)
{
    memset(msg, 0, sizeof(*msg));
    msg->msg[0] = (initiator << 4) | destination;
    msg->len = 1;
}

/**
 * cec_msg_set_reply_to - fill in destination/initiator in a reply message.
 * @msg:                the message structure for the reply
 * @orig:               the original message structure
 *
 * Set the msg destination to the orig initiator and the msg initiator to the
 * orig destination. Note that msg and orig may be the same pointer, in which
 * case the change is done in place.
 */
static inline void cec_msg_set_reply_to(struct cec_msg *msg,
                                       struct cec_msg *orig)
{
    /* The destination becomes the initiator and vice versa */
    msg->msg[0] = (cec_msg_destination(orig) << 4) |
                 cec_msg_initiator(orig);
    msg->reply = msg->timeout = 0;
}

/* cec_msg flags field */
#define CEC_MSG_FL_REPLY_TO_FOLLOWERS    (1 << 0)
```

```

#define CEC_MSG_FL_RAW (1 << 1)

/* cec_msg tx/rx_status field */
#define CEC_TX_STATUS_OK (1 << 0)
#define CEC_TX_STATUS_ARB_LOST (1 << 1)
#define CEC_TX_STATUS_NACK (1 << 2)
#define CEC_TX_STATUS_LOW_DRIVE (1 << 3)
#define CEC_TX_STATUS_ERROR (1 << 4)
#define CEC_TX_STATUS_MAX_RETRIES (1 << 5)
#define CEC_TX_STATUS_ABORTED (1 << 6)
#define CEC_TX_STATUS_TIMEOUT (1 << 7)

#define CEC_RX_STATUS_OK (1 << 0)
#define CEC_RX_STATUS_TIMEOUT (1 << 1)
#define CEC_RX_STATUS_FEATURE_ABORT (1 << 2)
#define CEC_RX_STATUS_ABORTED (1 << 3)

static inline int cec_msg_status_is_ok(const struct cec_msg *msg)
{
    if (msg->tx_status && !(msg->tx_status & CEC_TX_STATUS_OK))
        return 0;
    if (msg->rx_status && !(msg->rx_status & CEC_RX_STATUS_OK))
        return 0;
    if (!msg->tx_status && !msg->rx_status)
        return 0;
    return !(msg->rx_status & CEC_RX_STATUS_FEATURE_ABORT);
}

#define CEC_LOG_ADDR_INVALID 0xff
#define CEC_PHYS_ADDR_INVALID 0xffff

/*
 * The maximum number of logical addresses one device can be assigned to.
 * The CEC 2.0 spec allows for only 2 logical addresses at the moment. The
 * Analog Devices CEC hardware supports 3. So let's go wild and go for 4.
 */
#define CEC_MAX_LOG_ADDRS 4

/* The logical addresses defined by CEC 2.0 */
#define CEC_LOG_ADDR_TV 0
#define CEC_LOG_ADDR_RECORD_1 1
#define CEC_LOG_ADDR_RECORD_2 2
#define CEC_LOG_ADDR_TUNER_1 3
#define CEC_LOG_ADDR_PLAYBACK_1 4
#define CEC_LOG_ADDR_AUDIOSYSTEM 5
#define CEC_LOG_ADDR_TUNER_2 6
#define CEC_LOG_ADDR_TUNER_3 7
#define CEC_LOG_ADDR_PLAYBACK_2 8
#define CEC_LOG_ADDR_RECORD_3 9
#define CEC_LOG_ADDR_TUNER_4 10
#define CEC_LOG_ADDR_PLAYBACK_3 11

```

```
#define CEC_LOG_ADDR_BACKUP_1          12
#define CEC_LOG_ADDR_BACKUP_2          13
#define CEC_LOG_ADDR_SPECIFIC          14
#define CEC_LOG_ADDR_UNREGISTERED      15 /* as initiator address */
#define CEC_LOG_ADDR_BROADCAST         15 /* as destination address */

/* The logical address types that the CEC device wants to claim */
#define CEC_LOG_ADDR_TYPE_TV           0
#define CEC_LOG_ADDR_TYPE_RECORD      1
#define CEC_LOG_ADDR_TYPE_TUNER       2
#define CEC_LOG_ADDR_TYPE_PLAYBACK    3
#define CEC_LOG_ADDR_TYPE_AUDIOSYSTEM 4
#define CEC_LOG_ADDR_TYPE_SPECIFIC    5
#define CEC_LOG_ADDR_TYPE_UNREGISTERED 6
/*
 * Switches should use UNREGISTERED.
 * Processors should use SPECIFIC.
 */

#define CEC_LOG_ADDR_MASK_TV           (1 << CEC_LOG_ADDR_TV)
#define CEC_LOG_ADDR_MASK_RECORD      ((1 << CEC_LOG_ADDR_RECORD_1) | \
                                       (1 << CEC_LOG_ADDR_RECORD_2) | \
                                       (1 << CEC_LOG_ADDR_RECORD_3))
#define CEC_LOG_ADDR_MASK_TUNER       ((1 << CEC_LOG_ADDR_TUNER_1) | \
                                       (1 << CEC_LOG_ADDR_TUNER_2) | \
                                       (1 << CEC_LOG_ADDR_TUNER_3) | \
                                       (1 << CEC_LOG_ADDR_TUNER_4))
#define CEC_LOG_ADDR_MASK_PLAYBACK    ((1 << CEC_LOG_ADDR_PLAYBACK_1) | \
                                       (1 << CEC_LOG_ADDR_PLAYBACK_2) | \
                                       (1 << CEC_LOG_ADDR_PLAYBACK_3))
#define CEC_LOG_ADDR_MASK_AUDIOSYSTEM (1 << CEC_LOG_ADDR_AUDIOSYSTEM)
#define CEC_LOG_ADDR_MASK_BACKUP      ((1 << CEC_LOG_ADDR_BACKUP_1) | \
                                       (1 << CEC_LOG_ADDR_BACKUP_2))
#define CEC_LOG_ADDR_MASK_SPECIFIC    (1 << CEC_LOG_ADDR_SPECIFIC)
#define CEC_LOG_ADDR_MASK_UNREGISTERED (1 << CEC_LOG_ADDR_UNREGISTERED)

static inline int cec_has_tv(__u16 log_addr_mask)
{
    return log_addr_mask & CEC_LOG_ADDR_MASK_TV;
}

static inline int cec_has_record(__u16 log_addr_mask)
{
    return log_addr_mask & CEC_LOG_ADDR_MASK_RECORD;
}

static inline int cec_has_tuner(__u16 log_addr_mask)
{
    return log_addr_mask & CEC_LOG_ADDR_MASK_TUNER;
}
```

```

static inline int cec_has_playback(__u16 log_addr_mask)
{
    return log_addr_mask & CEC_LOG_ADDR_MASK_PLAYBACK;
}

static inline int cec_has_audiosystem(__u16 log_addr_mask)
{
    return log_addr_mask & CEC_LOG_ADDR_MASK_AUDIOSYSTEM;
}

static inline int cec_has_backup(__u16 log_addr_mask)
{
    return log_addr_mask & CEC_LOG_ADDR_MASK_BACKUP;
}

static inline int cec_has_specific(__u16 log_addr_mask)
{
    return log_addr_mask & CEC_LOG_ADDR_MASK_SPECIFIC;
}

static inline int cec_is_unregistered(__u16 log_addr_mask)
{
    return log_addr_mask & CEC_LOG_ADDR_MASK_UNREGISTERED;
}

static inline int cec_is_unconfigured(__u16 log_addr_mask)
{
    return log_addr_mask == 0;
}

/*
 * Use this if there is no vendor ID (CEC_G_VENDOR_ID) or if the vendor ID
 * should be disabled (CEC_S_VENDOR_ID)
 */
#define CEC_VENDOR_ID_NONE                0xffffffff

/* The message handling modes */
/* Modes for initiator */
#define CEC_MODE_NO_INITIATOR              (0x0 << 0)
#define CEC_MODE_INITIATOR                (0x1 << 0)
#define CEC_MODE_EXCL_INITIATOR           (0x2 << 0)
#define CEC_MODE_INITIATOR_MSK            0x0f

/* Modes for follower */
#define CEC_MODE_NO_FOLLOWER               (0x0 << 4)
#define CEC_MODE_FOLLOWER                 (0x1 << 4)
#define CEC_MODE_EXCL_FOLLOWER            (0x2 << 4)
#define CEC_MODE_EXCL_FOLLOWER_PASSTHRU  (0x3 << 4)
#define CEC_MODE_MONITOR_PIN              (0xd << 4)
#define CEC_MODE_MONITOR                  (0xe << 4)
#define CEC_MODE_MONITOR_ALL              (0xf << 4)

```

```
#define CEC_MODE_FOLLOWER_MSK                0xf0

/* Userspace has to configure the physical address */
#define CEC_CAP_PHYS_ADDR                    (1 << 0)
/* Userspace has to configure the logical addresses */
#define CEC_CAP_LOG_ADDRS                    (1 << 1)
/* Userspace can transmit messages (and thus become follower as well) */
#define CEC_CAP_TRANSMIT                     (1 << 2)
/*
 * Passthrough all messages instead of processing them.
 */
#define CEC_CAP_PASSTHROUGH                  (1 << 3)
/* Supports remote control */
#define CEC_CAP_RC                            (1 << 4)
/* Hardware can monitor all messages, not just directed and broadcast. */
#define CEC_CAP_MONITOR_ALL                  (1 << 5)
/* Hardware can use CEC only if the HDMI HPD pin is high. */
#define CEC_CAP_NEEDS_HPD                    (1 << 6)
/* Hardware can monitor CEC pin transitions */
#define CEC_CAP_MONITOR_PIN                  (1 << 7)
/* CEC_ADAP_G_CONNECTOR_INFO is available */
#define CEC_CAP_CONNECTOR_INFO              (1 << 8)

/**
 * struct cec_caps - CEC capabilities structure.
 * @driver: name of the CEC device driver.
 * @name: name of the CEC device. @driver + @name must be unique.
 * @available_log_addrs: number of available logical addresses.
 * @capabilities: capabilities of the CEC adapter.
 * @version: version of the CEC adapter framework.
 */
struct cec_caps {
    char driver[32];
    char name[32];
    __u32 available_log_addrs;
    __u32 capabilities;
    __u32 version;
};

/**
 * struct cec_log_addrs - CEC logical addresses structure.
 * @log_addr: the claimed logical addresses. Set by the driver.
 * @log_addr_mask: current logical address mask. Set by the driver.
 * @cec_version: the CEC version that the adapter should implement. Set by the
 * caller.
 * @num_log_addrs: how many logical addresses should be claimed. Set by the
 * caller.
 * @vendor_id: the vendor ID of the device. Set by the caller.
 * @flags: flags.
 * @osd_name: the OSD name of the device. Set by the caller.
 * @primary_device_type: the primary device type for each logical address.
 */
```

```

*      Set by the caller.
* @log_addr_type: the logical address types. Set by the caller.
* @all_device_types: CEC 2.0: all device types represented by the logical
*      address. Set by the caller.
* @features: CEC 2.0: The logical address features. Set by the caller.
*/
struct cec_log_addrs {
    __u8 log_addr[CEC_MAX_LOG_ADDRS];
    __u16 log_addr_mask;
    __u8 cec_version;
    __u8 num_log_addrs;
    __u32 vendor_id;
    __u32 flags;
    char osd_name[15];
    __u8 primary_device_type[CEC_MAX_LOG_ADDRS];
    __u8 log_addr_type[CEC_MAX_LOG_ADDRS];

    /* CEC 2.0 */
    __u8 all_device_types[CEC_MAX_LOG_ADDRS];
    __u8 features[CEC_MAX_LOG_ADDRS][12];
};

/* Allow a fallback to unregistered */
#define CEC_LOG_ADDRS_FL_ALLOW_UNREG_FALLBACK (1 << 0)
/* Passthrough RC messages to the input subsystem */
#define CEC_LOG_ADDRS_FL_ALLOW_RC_PASSTHRU (1 << 1)
/* CDC-Only device: supports only CDC messages */
#define CEC_LOG_ADDRS_FL_CDC_ONLY (1 << 2)

/**
 * struct cec_drm_connector_info - tells which drm connector is
 * associated with the CEC adapter.
 * @card_no: drm card number
 * @connector_id: drm connector ID
 */
struct cec_drm_connector_info {
    __u32 card_no;
    __u32 connector_id;
};

#define CEC_CONNECTOR_TYPE_NO_CONNECTOR 0
#define CEC_CONNECTOR_TYPE_DRM 1

/**
 * struct cec_connector_info - tells if and which connector is
 * associated with the CEC adapter.
 * @type: connector type (if any)
 * @drm: drm connector info
 * @raw: array to pad the union
 */
struct cec_connector_info {

```

```
    __u32 type;
    union {
        struct cec_drm_connector_info drm;
        __u32 raw[16];
    };
};

/* Events */

/* Event that occurs when the adapter state changes */
#define CEC_EVENT_STATE_CHANGE 1
/*
 * This event is sent when messages are lost because the application
 * didn't empty the message queue in time
 */
#define CEC_EVENT_LOST_MSGS 2
#define CEC_EVENT_PIN_CEC_LOW 3
#define CEC_EVENT_PIN_CEC_HIGH 4
#define CEC_EVENT_PIN_HPD_LOW 5
#define CEC_EVENT_PIN_HPD_HIGH 6
#define CEC_EVENT_PIN_5V_LOW 7
#define CEC_EVENT_PIN_5V_HIGH 8

#define CEC_EVENT_FL_INITIAL_STATE (1 << 0)
#define CEC_EVENT_FL_DROPPED_EVENTS (1 << 1)

/**
 * struct cec_event_state_change - used when the CEC adapter changes state.
 * @phys_addr: the current physical address
 * @log_addr_mask: the current logical address mask
 * @have_conn_info: if non-zero, then HDMI connector information is available.
 * This field is only valid if CEC_CAP_CONNECTOR_INFO is set. If that
 * capability is set and @have_conn_info is zero, then that indicates
 * that the HDMI connector device is not instantiated, either because
 * the HDMI driver is still configuring the device or because the HDMI
 * device was unbound.
 */
struct cec_event_state_change {
    __u16 phys_addr;
    __u16 log_addr_mask;
    __u16 have_conn_info;
};

/**
 * struct cec_event_lost_msgs - tells you how many messages were lost.
 * @lost_msgs: how many messages were lost.
 */
struct cec_event_lost_msgs {
    __u32 lost_msgs;
};
```



```

/**
 * struct cec_event - CEC event structure
 * @ts: the timestamp of when the event was sent.
 * @event: the event.
 * @flags: event flags.
 * @state_change: the event payload for CEC_EVENT_STATE_CHANGE.
 * @lost_msgs: the event payload for CEC_EVENT_LOST_MSGS.
 * @raw: array to pad the union.
 */
struct cec_event {
    __u64 ts;
    __u32 event;
    __u32 flags;
    union {
        struct cec_event_state_change state_change;
        struct cec_event_lost_msgs lost_msgs;
        __u32 raw[16];
    };
};

/* ioctls */

/* Adapter capabilities */
#define CEC_ADAP_G_CAPS          _IOWR('a', 0, struct cec_caps)

/*
 * phys_addr is either 0 (if this is the CEC root device)
 * or a valid physical address obtained from the sink's EDID
 * as read by this CEC device (if this is a source device)
 * or a physical address obtained and modified from a sink
 * EDID and used for a sink CEC device.
 * If nothing is connected, then phys_addr is 0xffff.
 * See HDMI 1.4b, section 8.7 (Physical Address).
 *
 * The CEC_ADAP_S_PHYS_ADDR ioctl may not be available if that is handled
 * internally.
 */
#define CEC_ADAP_G_PHYS_ADDR    _IOR('a', 1, __u16)
#define CEC_ADAP_S_PHYS_ADDR    _IOW('a', 2, __u16)

/*
 * Configure the CEC adapter. It sets the device type and which
 * logical types it will try to claim. It will return which
 * logical addresses it could actually claim.
 * An error is returned if the adapter is disabled or if there
 * is no physical address assigned.
 */

#define CEC_ADAP_G_LOG_ADDRS    _IOR('a', 3, struct cec_log_addrs)
#define CEC_ADAP_S_LOG_ADDRS    _IOWR('a', 4, struct cec_log_addrs)

```

```
/* Transmit/receive a CEC command */
#define CEC_TRANSMIT          _IOWR('a', 5, struct cec_msg)
#define CEC_RECEIVE          _IOWR('a', 6, struct cec_msg)

/* Dequeue CEC events */
#define CEC_DQEVENT          _IOWR('a', 7, struct cec_event)

/*
 * Get and set the message handling mode for this filehandle.
 */
#define CEC_G_MODE           _IOR('a', 8, __u32)
#define CEC_S_MODE           _IOW('a', 9, __u32)

/* Get the connector info */
#define CEC_ADAP_G_CONNECTOR_INFO _IOR('a', 10, struct cec_connector_info)

/*
 * The remainder of this header defines all CEC messages and operands.
 * The format matters since it the cec-ctl utility parses it to generate
 * code for implementing all these messages.
 *
 * Comments ending with 'Feature' group messages for each feature.
 * If messages are part of multiple features, then the "Has also"
 * comment is used to list the previously defined messages that are
 * supported by the feature.
 *
 * Before operands are defined a comment is added that gives the
 * name of the operand and in brackets the variable name of the
 * corresponding argument in the cec-funcs.h function.
 */

/* Messages */

/* One Touch Play Feature */
#define CEC_MSG_ACTIVE_SOURCE          0x82
#define CEC_MSG_IMAGE_VIEW_ON         0x04
#define CEC_MSG_TEXT_VIEW_ON          0x0d

/* Routing Control Feature */

/*
 * Has also:
 *     CEC_MSG_ACTIVE_SOURCE
 */

#define CEC_MSG_INACTIVE_SOURCE        0x9d
#define CEC_MSG_REQUEST_ACTIVE_SOURCE 0x85
#define CEC_MSG_ROUTING_CHANGE        0x80
#define CEC_MSG_ROUTING_INFORMATION   0x81
#define CEC_MSG_SET_STREAM_PATH        0x86
```

```

/* Standby Feature */
#define CEC_MSG_STANDBY                                0x36

/* One Touch Record Feature */
#define CEC_MSG_RECORD_OFF                            0x0b
#define CEC_MSG_RECORD_ON                             0x09
/* Record Source Type Operand (rec_src_type) */
#define CEC_OP_RECORD_SRC_OWN                         1
#define CEC_OP_RECORD_SRC_DIGITAL                     2
#define CEC_OP_RECORD_SRC_ANALOG                      3
#define CEC_OP_RECORD_SRC_EXT_PLUG                    4
#define CEC_OP_RECORD_SRC_EXT_PHYS_ADDR               5
/* Service Identification Method Operand (service_id_method) */
#define CEC_OP_SERVICE_ID_METHOD_BY_DIG_ID            0
#define CEC_OP_SERVICE_ID_METHOD_BY_CHANNEL           1
/* Digital Service Broadcast System Operand (dig_bcast_system) */
#define CEC_OP_DIG_SERVICE_BCAST_SYSTEM_ARIB_GEN      0x00
#define CEC_OP_DIG_SERVICE_BCAST_SYSTEM_ATSC_GEN     0x01
#define CEC_OP_DIG_SERVICE_BCAST_SYSTEM_DVB_GEN      0x02
#define CEC_OP_DIG_SERVICE_BCAST_SYSTEM_ARIB_BS      0x08
#define CEC_OP_DIG_SERVICE_BCAST_SYSTEM_ARIB_CS      0x09
#define CEC_OP_DIG_SERVICE_BCAST_SYSTEM_ARIB_T       0x0a
#define CEC_OP_DIG_SERVICE_BCAST_SYSTEM_ATSC_CABLE   0x10
#define CEC_OP_DIG_SERVICE_BCAST_SYSTEM_ATSC_SAT     0x11
#define CEC_OP_DIG_SERVICE_BCAST_SYSTEM_ATSC_T       0x12
#define CEC_OP_DIG_SERVICE_BCAST_SYSTEM_DVB_C        0x18
#define CEC_OP_DIG_SERVICE_BCAST_SYSTEM_DVB_S        0x19
#define CEC_OP_DIG_SERVICE_BCAST_SYSTEM_DVB_S2       0x1a
#define CEC_OP_DIG_SERVICE_BCAST_SYSTEM_DVB_T        0x1b
/* Analogue Broadcast Type Operand (ana_bcast_type) */
#define CEC_OP_ANA_BCAST_TYPE_CABLE                   0
#define CEC_OP_ANA_BCAST_TYPE_SATELLITE               1
#define CEC_OP_ANA_BCAST_TYPE_TERRESTRIAL             2
/* Broadcast System Operand (bcast_system) */
#define CEC_OP_BCAST_SYSTEM_PAL_BG                    0x00
#define CEC_OP_BCAST_SYSTEM_SECAM_LQ                  0x01 /* SECAM L' */
#define CEC_OP_BCAST_SYSTEM_PAL_M                     0x02
#define CEC_OP_BCAST_SYSTEM_NTSC_M                    0x03
#define CEC_OP_BCAST_SYSTEM_PAL_I                     0x04
#define CEC_OP_BCAST_SYSTEM_SECAM_DK                  0x05
#define CEC_OP_BCAST_SYSTEM_SECAM_BG                  0x06
#define CEC_OP_BCAST_SYSTEM_SECAM_L                   0x07
#define CEC_OP_BCAST_SYSTEM_PAL_DK                    0x08
#define CEC_OP_BCAST_SYSTEM_OTHER                     0x1f
/* Channel Number Format Operand (channel_number_fmt) */
#define CEC_OP_CHANNEL_NUMBER_FMT_1_PART              0x01
#define CEC_OP_CHANNEL_NUMBER_FMT_2_PART              0x02

#define CEC_MSG_RECORD_STATUS                          0x0a
/* Record Status Operand (rec_status) */
#define CEC_OP_RECORD_STATUS_CUR_SRC                  0x01

```

```
#define CEC_OP_RECORD_STATUS_DIG_SERVICE          0x02
#define CEC_OP_RECORD_STATUS_ANA_SERVICE         0x03
#define CEC_OP_RECORD_STATUS_EXT_INPUT          0x04
#define CEC_OP_RECORD_STATUS_NO_DIG_SERVICE     0x05
#define CEC_OP_RECORD_STATUS_NO_ANA_SERVICE     0x06
#define CEC_OP_RECORD_STATUS_NO_SERVICE        0x07
#define CEC_OP_RECORD_STATUS_INVALID_EXT_PLUG   0x09
#define CEC_OP_RECORD_STATUS_INVALID_EXT_PHYS_ADDR 0x0a
#define CEC_OP_RECORD_STATUS_UNSUP_CA          0x0b
#define CEC_OP_RECORD_STATUS_NO_CA_ENTITLEMENTS 0x0c
#define CEC_OP_RECORD_STATUS_CANT_COPY_SRC     0x0d
#define CEC_OP_RECORD_STATUS_NO_MORE_COPIES    0x0e
#define CEC_OP_RECORD_STATUS_NO_MEDIA          0x10
#define CEC_OP_RECORD_STATUS_PLAYING           0x11
#define CEC_OP_RECORD_STATUS_ALREADY_RECORDING 0x12
#define CEC_OP_RECORD_STATUS_MEDIA_PROT        0x13
#define CEC_OP_RECORD_STATUS_NO_SIGNAL         0x14
#define CEC_OP_RECORD_STATUS_MEDIA_PROBLEM     0x15
#define CEC_OP_RECORD_STATUS_NO_SPACE          0x16
#define CEC_OP_RECORD_STATUS_PARENTAL_LOCK     0x17
#define CEC_OP_RECORD_STATUS_TERMINATED_OK     0x1a
#define CEC_OP_RECORD_STATUS_ALREADY_TERM      0x1b
#define CEC_OP_RECORD_STATUS_OTHER             0x1f

#define CEC_MSG_RECORD_TV_SCREEN                0x0f

/* Timer Programming Feature */
#define CEC_MSG_CLEAR_ANALOGUE_TIMER            0x33
/* Recording Sequence Operand (recording_seq) */
#define CEC_OP_REC_SEQ_SUNDAY                   0x01
#define CEC_OP_REC_SEQ_MONDAY                   0x02
#define CEC_OP_REC_SEQ_TUESDAY                  0x04
#define CEC_OP_REC_SEQ_WEDNESDAY                0x08
#define CEC_OP_REC_SEQ_THURSDAY                0x10
#define CEC_OP_REC_SEQ_FRIDAY                  0x20
#define CEC_OP_REC_SEQ_SATURDAY                 0x40
#define CEC_OP_REC_SEQ_ONCE_ONLY                0x00

#define CEC_MSG_CLEAR_DIGITAL_TIMER             0x99

#define CEC_MSG_CLEAR_EXT_TIMER                 0xa1
/* External Source Specifier Operand (ext_src_spec) */
#define CEC_OP_EXT_SRC_PLUG                     0x04
#define CEC_OP_EXT_SRC_PHYS_ADDR                0x05

#define CEC_MSG_SET_ANALOGUE_TIMER              0x34
#define CEC_MSG_SET_DIGITAL_TIMER              0x97
#define CEC_MSG_SET_EXT_TIMER                   0xa2

#define CEC_MSG_SET_TIMER_PROGRAM_TITLE        0x67
#define CEC_MSG_TIMER_CLEARED_STATUS           0x43
```

```

/* Timer Cleared Status Data Operand (timer_cleared_status) */
#define CEC_OP_TIMER_CLR_STAT_RECORDING          0x00
#define CEC_OP_TIMER_CLR_STAT_NO_MATCHING       0x01
#define CEC_OP_TIMER_CLR_STAT_NO_INFO          0x02
#define CEC_OP_TIMER_CLR_STAT_CLEARED          0x80

#define CEC_MSG_TIMER_STATUS                     0x35
/* Timer Overlap Warning Operand (timer_overlap_warning) */
#define CEC_OP_TIMER_OVERLAP_WARNING_NO_OVERLAP 0
#define CEC_OP_TIMER_OVERLAP_WARNING_OVERLAP   1
/* Media Info Operand (media_info) */
#define CEC_OP_MEDIA_INFO_UNPROT_MEDIA         0
#define CEC_OP_MEDIA_INFO_PROT_MEDIA          1
#define CEC_OP_MEDIA_INFO_NO_MEDIA            2
/* Programmed Indicator Operand (prog_indicator) */
#define CEC_OP_PROG_IND_NOT_PROGRAMMED         0
#define CEC_OP_PROG_IND_PROGRAMMED            1
/* Programmed Info Operand (prog_info) */
#define CEC_OP_PROG_INFO_ENOUGH_SPACE          0x08
#define CEC_OP_PROG_INFO_NOT_ENOUGH_SPACE      0x09
#define CEC_OP_PROG_INFO_MIGHT_NOT_BE_ENOUGH_SPACE 0x0b
#define CEC_OP_PROG_INFO_NONE_AVAILABLE       0x0a
/* Not Programmed Error Info Operand (prog_error) */
#define CEC_OP_PROG_ERROR_NO_FREE_TIMER        0x01
#define CEC_OP_PROG_ERROR_DATE_OUT_OF_RANGE    0x02
#define CEC_OP_PROG_ERROR_REC_SEQ_ERROR        0x03
#define CEC_OP_PROG_ERROR_INV_EXT_PLUG         0x04
#define CEC_OP_PROG_ERROR_INV_EXT_PHYS_ADDR    0x05
#define CEC_OP_PROG_ERROR_CA_UNSUPP            0x06
#define CEC_OP_PROG_ERROR_INSUF_CA_ENTITLEMENTS 0x07
#define CEC_OP_PROG_ERROR_RESOLUTION_UNSUPP    0x08
#define CEC_OP_PROG_ERROR_PARENTAL_LOCK        0x09
#define CEC_OP_PROG_ERROR_CLOCK_FAILURE        0x0a
#define CEC_OP_PROG_ERROR_DUPLICATE            0x0e

/* System Information Feature */
#define CEC_MSG_CEC_VERSION                     0x9e
/* CEC Version Operand (cec_version) */
#define CEC_OP_CEC_VERSION_1_3A                 4
#define CEC_OP_CEC_VERSION_1_4                 5
#define CEC_OP_CEC_VERSION_2_0                 6

#define CEC_MSG_GET_CEC_VERSION                 0x9f
#define CEC_MSG_GIVE_PHYSICAL_ADDR             0x83
#define CEC_MSG_GET_MENU_LANGUAGE             0x91
#define CEC_MSG_REPORT_PHYSICAL_ADDR          0x84
/* Primary Device Type Operand (prim_devtype) */
#define CEC_OP_PRIM_DEVTYPE_TV                 0
#define CEC_OP_PRIM_DEVTYPE_RECORD            1
#define CEC_OP_PRIM_DEVTYPE_TUNER             3
#define CEC_OP_PRIM_DEVTYPE_PLAYBACK          4

```

```

#define CEC_OP_PRIM_DEVTYPE_AUDIOSYSTEM          5
#define CEC_OP_PRIM_DEVTYPE_SWITCH              6
#define CEC_OP_PRIM_DEVTYPE_PROCESSOR          7

#define CEC_MSG_SET_MENU_LANGUAGE              0x32
#define CEC_MSG_REPORT_FEATURES              0xa6    /* HDMI 2.0 */
/* All Device Types Operand (all_device_types) */
#define CEC_OP_ALL_DEVTYPE_TV                  0x80
#define CEC_OP_ALL_DEVTYPE_RECORD             0x40
#define CEC_OP_ALL_DEVTYPE_TUNER             0x20
#define CEC_OP_ALL_DEVTYPE_PLAYBACK          0x10
#define CEC_OP_ALL_DEVTYPE_AUDIOSYSTEM       0x08
#define CEC_OP_ALL_DEVTYPE_SWITCH            0x04
/*
 * And if you wondering what happened to PROCESSOR devices: those should
 * be mapped to a SWITCH.
 */

/* Valid for RC Profile and Device Feature operands */
#define CEC_OP_FEAT_EXT                        0x80    /* Extension_
→bit */
/* RC Profile Operand (rc_profile) */
#define CEC_OP_FEAT_RC_TV_PROFILE_NONE        0x00
#define CEC_OP_FEAT_RC_TV_PROFILE_1          0x02
#define CEC_OP_FEAT_RC_TV_PROFILE_2          0x06
#define CEC_OP_FEAT_RC_TV_PROFILE_3          0x0a
#define CEC_OP_FEAT_RC_TV_PROFILE_4          0x0e
#define CEC_OP_FEAT_RC_SRC_HAS_DEV_ROOT_MENU 0x50
#define CEC_OP_FEAT_RC_SRC_HAS_DEV_SETUP_MENU 0x48
#define CEC_OP_FEAT_RC_SRC_HAS_CONTENTS_MENU 0x44
#define CEC_OP_FEAT_RC_SRC_HAS_MEDIA_TOP_MENU 0x42
#define CEC_OP_FEAT_RC_SRC_HAS_MEDIA_CONTEXT_MENU 0x41
/* Device Feature Operand (dev_features) */
#define CEC_OP_FEAT_DEV_HAS_RECORD_TV_SCREEN  0x40
#define CEC_OP_FEAT_DEV_HAS_SET_OSD_STRING    0x20
#define CEC_OP_FEAT_DEV_HAS_DECK_CONTROL      0x10
#define CEC_OP_FEAT_DEV_HAS_SET_AUDIO_RATE    0x08
#define CEC_OP_FEAT_DEV_SINK_HAS_ARC_TX       0x04
#define CEC_OP_FEAT_DEV_SOURCE_HAS_ARC_RX     0x02

#define CEC_MSG_GIVE_FEATURES                 0xa5    /* HDMI 2.0 */

/* Deck Control Feature */
#define CEC_MSG_DECK_CONTROL                  0x42
/* Deck Control Mode Operand (deck_control_mode) */
#define CEC_OP_DECK_CTL_MODE_SKIP_FWD         1
#define CEC_OP_DECK_CTL_MODE_SKIP_REV        2
#define CEC_OP_DECK_CTL_MODE_STOP            3
#define CEC_OP_DECK_CTL_MODE_EJECT           4

#define CEC_MSG_DECK_STATUS                   0x1b

```

```

/* Deck Info Operand (deck_info) */
#define CEC_OP_DECK_INFO_PLAY                0x11
#define CEC_OP_DECK_INFO_RECORD              0x12
#define CEC_OP_DECK_INFO_PLAY_REV           0x13
#define CEC_OP_DECK_INFO_STILL              0x14
#define CEC_OP_DECK_INFO_SLOW               0x15
#define CEC_OP_DECK_INFO_SLOW_REV           0x16
#define CEC_OP_DECK_INFO_FAST_FWD           0x17
#define CEC_OP_DECK_INFO_FAST_REV           0x18
#define CEC_OP_DECK_INFO_NO_MEDIA           0x19
#define CEC_OP_DECK_INFO_STOP               0x1a
#define CEC_OP_DECK_INFO_SKIP_FWD           0x1b
#define CEC_OP_DECK_INFO_SKIP_REV           0x1c
#define CEC_OP_DECK_INFO_INDEX_SEARCH_FWD   0x1d
#define CEC_OP_DECK_INFO_INDEX_SEARCH_REV   0x1e
#define CEC_OP_DECK_INFO_OTHER              0x1f

#define CEC_MSG_GIVE_DECK_STATUS             0x1a
/* Status Request Operand (status_req) */
#define CEC_OP_STATUS_REQ_ON                 1
#define CEC_OP_STATUS_REQ_OFF                2
#define CEC_OP_STATUS_REQ_ONCE               3

#define CEC_MSG_PLAY                         0x41
/* Play Mode Operand (play_mode) */
#define CEC_OP_PLAY_MODE_PLAY_FWD            0x24
#define CEC_OP_PLAY_MODE_PLAY_REV            0x20
#define CEC_OP_PLAY_MODE_PLAY_STILL          0x25
#define CEC_OP_PLAY_MODE_PLAY_FAST_FWD_MIN   0x05
#define CEC_OP_PLAY_MODE_PLAY_FAST_FWD_MED   0x06
#define CEC_OP_PLAY_MODE_PLAY_FAST_FWD_MAX   0x07
#define CEC_OP_PLAY_MODE_PLAY_FAST_REV_MIN   0x09
#define CEC_OP_PLAY_MODE_PLAY_FAST_REV_MED   0x0a
#define CEC_OP_PLAY_MODE_PLAY_FAST_REV_MAX   0x0b
#define CEC_OP_PLAY_MODE_PLAY_SLOW_FWD_MIN   0x15
#define CEC_OP_PLAY_MODE_PLAY_SLOW_FWD_MED   0x16
#define CEC_OP_PLAY_MODE_PLAY_SLOW_FWD_MAX   0x17
#define CEC_OP_PLAY_MODE_PLAY_SLOW_REV_MIN   0x19
#define CEC_OP_PLAY_MODE_PLAY_SLOW_REV_MED   0x1a
#define CEC_OP_PLAY_MODE_PLAY_SLOW_REV_MAX   0x1b

/* Tuner Control Feature */
#define CEC_MSG_GIVE_TUNER_DEVICE_STATUS     0x08
#define CEC_MSG_SELECT_ANALOGUE_SERVICE     0x92
#define CEC_MSG_SELECT_DIGITAL_SERVICE      0x93
#define CEC_MSG_TUNER_DEVICE_STATUS         0x07
/* Recording Flag Operand (rec_flag) */
#define CEC_OP_REC_FLAG_NOT_USED             0
#define CEC_OP_REC_FLAG_USED                 1
/* Tuner Display Info Operand (tuner_display_info) */
#define CEC_OP_TUNER_DISPLAY_INFO_DIGITAL    0

```

```
#define CEC_OP_TUNER_DISPLAY_INFO_NONE 1
#define CEC_OP_TUNER_DISPLAY_INFO_ANALOGUE 2

#define CEC_MSG_TUNER_STEP_DECREMENT 0x06
#define CEC_MSG_TUNER_STEP_INCREMENT 0x05

/* Vendor Specific Commands Feature */

/*
 * Has also:
 *     CEC_MSG_CEC_VERSION
 *     CEC_MSG_GET_CEC_VERSION
 */
#define CEC_MSG_DEVICE_VENDOR_ID 0x87
#define CEC_MSG_GIVE_DEVICE_VENDOR_ID 0x8c
#define CEC_MSG_VENDOR_COMMAND 0x89
#define CEC_MSG_VENDOR_COMMAND_WITH_ID 0xa0
#define CEC_MSG_VENDOR_REMOTE_BUTTON_DOWN 0x8a
#define CEC_MSG_VENDOR_REMOTE_BUTTON_UP 0x8b

/* OSD Display Feature */
#define CEC_MSG_SET_OSD_STRING 0x64
/* Display Control Operand (disp_ctl) */
#define CEC_OP_DISP_CTL_DEFAULT 0x00
#define CEC_OP_DISP_CTL_UNTIL_CLEARED 0x40
#define CEC_OP_DISP_CTL_CLEAR 0x80

/* Device OSD Transfer Feature */
#define CEC_MSG_GIVE_OSD_NAME 0x46
#define CEC_MSG_SET_OSD_NAME 0x47

/* Device Menu Control Feature */
#define CEC_MSG_MENU_REQUEST 0x8d
/* Menu Request Type Operand (menu_req) */
#define CEC_OP_MENU_REQUEST_ACTIVATE 0x00
#define CEC_OP_MENU_REQUEST_DEACTIVATE 0x01
#define CEC_OP_MENU_REQUEST_QUERY 0x02

#define CEC_MSG_MENU_STATUS 0x8e
/* Menu State Operand (menu_state) */
#define CEC_OP_MENU_STATE_ACTIVATED 0x00
#define CEC_OP_MENU_STATE_DEACTIVATED 0x01

#define CEC_MSG_USER_CONTROL_PRESSED 0x44
/* UI Command Operand (ui_cmd) */
#define CEC_OP_UI_CMD_SELECT 0x00
#define CEC_OP_UI_CMD_UP 0x01
#define CEC_OP_UI_CMD_DOWN 0x02
#define CEC_OP_UI_CMD_LEFT 0x03
#define CEC_OP_UI_CMD_RIGHT 0x04
#define CEC_OP_UI_CMD_RIGHT_UP 0x05
```



```
#define CEC_OP_UI_CMD_RIGHT_DOWN          0x06
#define CEC_OP_UI_CMD_LEFT_UP            0x07
#define CEC_OP_UI_CMD_LEFT_DOWN          0x08
#define CEC_OP_UI_CMD_DEVICE_ROOT_MENU   0x09
#define CEC_OP_UI_CMD_DEVICE_SETUP_MENU  0x0a
#define CEC_OP_UI_CMD_CONTENTS_MENU      0x0b
#define CEC_OP_UI_CMD_FAVORITE_MENU      0x0c
#define CEC_OP_UI_CMD_BACK               0x0d
#define CEC_OP_UI_CMD_MEDIA_TOP_MENU     0x10
#define CEC_OP_UI_CMD_MEDIA_CONTEXT_SENSITIVE_MENU 0x11
#define CEC_OP_UI_CMD_NUMBER_ENTRY_MODE  0x1d
#define CEC_OP_UI_CMD_NUMBER_11          0x1e
#define CEC_OP_UI_CMD_NUMBER_12          0x1f
#define CEC_OP_UI_CMD_NUMBER_0_OR_NUMBER_10 0x20
#define CEC_OP_UI_CMD_NUMBER_1           0x21
#define CEC_OP_UI_CMD_NUMBER_2           0x22
#define CEC_OP_UI_CMD_NUMBER_3           0x23
#define CEC_OP_UI_CMD_NUMBER_4           0x24
#define CEC_OP_UI_CMD_NUMBER_5           0x25
#define CEC_OP_UI_CMD_NUMBER_6           0x26
#define CEC_OP_UI_CMD_NUMBER_7           0x27
#define CEC_OP_UI_CMD_NUMBER_8           0x28
#define CEC_OP_UI_CMD_NUMBER_9           0x29
#define CEC_OP_UI_CMD_DOT                 0x2a
#define CEC_OP_UI_CMD_ENTER               0x2b
#define CEC_OP_UI_CMD_CLEAR               0x2c
#define CEC_OP_UI_CMD_NEXT_FAVORITE      0x2f
#define CEC_OP_UI_CMD_CHANNEL_UP         0x30
#define CEC_OP_UI_CMD_CHANNEL_DOWN       0x31
#define CEC_OP_UI_CMD_PREVIOUS_CHANNEL   0x32
#define CEC_OP_UI_CMD_SOUND_SELECT       0x33
#define CEC_OP_UI_CMD_INPUT_SELECT       0x34
#define CEC_OP_UI_CMD_DISPLAY_INFORMATION 0x35
#define CEC_OP_UI_CMD_HELP                0x36
#define CEC_OP_UI_CMD_PAGE_UP            0x37
#define CEC_OP_UI_CMD_PAGE_DOWN          0x38
#define CEC_OP_UI_CMD_POWER              0x40
#define CEC_OP_UI_CMD_VOLUME_UP          0x41
#define CEC_OP_UI_CMD_VOLUME_DOWN        0x42
#define CEC_OP_UI_CMD_MUTE                0x43
#define CEC_OP_UI_CMD_PLAY                0x44
#define CEC_OP_UI_CMD_STOP                0x45
#define CEC_OP_UI_CMD_PAUSE               0x46
#define CEC_OP_UI_CMD_RECORD              0x47
#define CEC_OP_UI_CMD_REWIND              0x48
#define CEC_OP_UI_CMD_FAST_FORWARD        0x49
#define CEC_OP_UI_CMD_EJECT               0x4a
#define CEC_OP_UI_CMD_SKIP_FORWARD        0x4b
#define CEC_OP_UI_CMD_SKIP_BACKWARD       0x4c
#define CEC_OP_UI_CMD_STOP_RECORD         0x4d
#define CEC_OP_UI_CMD_PAUSE_RECORD        0x4e
```

```
#define CEC_OP_UI_CMD_ANGLE 0x50
#define CEC_OP_UI_CMD_SUB_PICTURE 0x51
#define CEC_OP_UI_CMD_VIDEO_ON_DEMAND 0x52
#define CEC_OP_UI_CMD_ELECTRONIC_PROGRAM_GUIDE 0x53
#define CEC_OP_UI_CMD_TIMER_PROGRAMMING 0x54
#define CEC_OP_UI_CMD_INITIAL_CONFIGURATION 0x55
#define CEC_OP_UI_CMD_SELECT_BROADCAST_TYPE 0x56
#define CEC_OP_UI_CMD_SELECT_SOUND_PRESENTATION 0x57
#define CEC_OP_UI_CMD_AUDIO_DESCRIPTION 0x58
#define CEC_OP_UI_CMD_INTERNET 0x59
#define CEC_OP_UI_CMD_3D_MODE 0x5a
#define CEC_OP_UI_CMD_PLAY_FUNCTION 0x60
#define CEC_OP_UI_CMD_PAUSE_PLAY_FUNCTION 0x61
#define CEC_OP_UI_CMD_RECORD_FUNCTION 0x62
#define CEC_OP_UI_CMD_PAUSE_RECORD_FUNCTION 0x63
#define CEC_OP_UI_CMD_STOP_FUNCTION 0x64
#define CEC_OP_UI_CMD_MUTE_FUNCTION 0x65
#define CEC_OP_UI_CMD_RESTORE_VOLUME_FUNCTION 0x66
#define CEC_OP_UI_CMD_TUNE_FUNCTION 0x67
#define CEC_OP_UI_CMD_SELECT_MEDIA_FUNCTION 0x68
#define CEC_OP_UI_CMD_SELECT_AV_INPUT_FUNCTION 0x69
#define CEC_OP_UI_CMD_SELECT_AUDIO_INPUT_FUNCTION 0x6a
#define CEC_OP_UI_CMD_POWER_TOGGLE_FUNCTION 0x6b
#define CEC_OP_UI_CMD_POWER_OFF_FUNCTION 0x6c
#define CEC_OP_UI_CMD_POWER_ON_FUNCTION 0x6d
#define CEC_OP_UI_CMD_F1_BLUE 0x71
#define CEC_OP_UI_CMD_F2_RED 0x72
#define CEC_OP_UI_CMD_F3_GREEN 0x73
#define CEC_OP_UI_CMD_F4_YELLOW 0x74
#define CEC_OP_UI_CMD_F5 0x75
#define CEC_OP_UI_CMD_DATA 0x76
/* UI Broadcast Type Operand (ui_bcast_type) */
#define CEC_OP_UI_BCAST_TYPE_TOGGLE_ALL 0x00
#define CEC_OP_UI_BCAST_TYPE_TOGGLE_DIG_ANA 0x01
#define CEC_OP_UI_BCAST_TYPE_ANALOGUE 0x10
#define CEC_OP_UI_BCAST_TYPE_ANALOGUE_T 0x20
#define CEC_OP_UI_BCAST_TYPE_ANALOGUE_CABLE 0x30
#define CEC_OP_UI_BCAST_TYPE_ANALOGUE_SAT 0x40
#define CEC_OP_UI_BCAST_TYPE_DIGITAL 0x50
#define CEC_OP_UI_BCAST_TYPE_DIGITAL_T 0x60
#define CEC_OP_UI_BCAST_TYPE_DIGITAL_CABLE 0x70
#define CEC_OP_UI_BCAST_TYPE_DIGITAL_SAT 0x80
#define CEC_OP_UI_BCAST_TYPE_DIGITAL_COM_SAT 0x90
#define CEC_OP_UI_BCAST_TYPE_DIGITAL_COM_SAT2 0x91
#define CEC_OP_UI_BCAST_TYPE_IP 0xa0
/* UI Sound Presentation Control Operand (ui_snd_pres_ctl) */
#define CEC_OP_UI_SND_PRES_CTL_DUAL_MONO 0x10
#define CEC_OP_UI_SND_PRES_CTL_KARAOKE 0x20
#define CEC_OP_UI_SND_PRES_CTL_DOWNMIX 0x80
#define CEC_OP_UI_SND_PRES_CTL_REVERB 0x90
#define CEC_OP_UI_SND_PRES_CTL_EQUALIZER 0xa0
```

```

#define CEC_OP_UI_SND_PRES_CTL_BASS_UP                0xb1
#define CEC_OP_UI_SND_PRES_CTL_BASS_NEUTRAL          0xb2
#define CEC_OP_UI_SND_PRES_CTL_BASS_DOWN            0xb3
#define CEC_OP_UI_SND_PRES_CTL_TREBLE_UP            0xc1
#define CEC_OP_UI_SND_PRES_CTL_TREBLE_NEUTRAL       0xc2
#define CEC_OP_UI_SND_PRES_CTL_TREBLE_DOWN         0xc3

#define CEC_MSG_USER_CONTROL_RELEASED                0x45

/* Remote Control Passthrough Feature */

/*
 * Has also:
 *     CEC_MSG_USER_CONTROL_PRESSED
 *     CEC_MSG_USER_CONTROL_RELEASED
 */

/* Power Status Feature */
#define CEC_MSG_GIVE_DEVICE_POWER_STATUS             0x8f
#define CEC_MSG_REPORT_POWER_STATUS                 0x90
/* Power Status Operand (pwr_state) */
#define CEC_OP_POWER_STATUS_ON                      0
#define CEC_OP_POWER_STATUS_STANDBY                 1
#define CEC_OP_POWER_STATUS_TO_ON                   2
#define CEC_OP_POWER_STATUS_TO_STANDBY              3

/* General Protocol Messages */
#define CEC_MSG_FEATURE_ABORT                        0x00
/* Abort Reason Operand (reason) */
#define CEC_OP_ABORT_UNRECOGNIZED_OP                0
#define CEC_OP_ABORT_INCORRECT_MODE                 1
#define CEC_OP_ABORT_NO_SOURCE                      2
#define CEC_OP_ABORT_INVALID_OP                     3
#define CEC_OP_ABORT_REFUSED                        4
#define CEC_OP_ABORT_UNDETERMINED                  5

#define CEC_MSG_ABORT                                0xff

/* System Audio Control Feature */

/*
 * Has also:
 *     CEC_MSG_USER_CONTROL_PRESSED
 *     CEC_MSG_USER_CONTROL_RELEASED
 */
#define CEC_MSG_GIVE_AUDIO_STATUS                    0x71
#define CEC_MSG_GIVE_SYSTEM_AUDIO_MODE_STATUS       0x7d
#define CEC_MSG_REPORT_AUDIO_STATUS                 0x7a
/* Audio Mute Status Operand (aud_mute_status) */
#define CEC_OP_AUD_MUTE_STATUS_OFF                  0
#define CEC_OP_AUD_MUTE_STATUS_ON                    1

```

```
#define CEC_MSG_REPORT_SHORT_AUDIO_DESCRIPTOR          0xa3
#define CEC_MSG_REQUEST_SHORT_AUDIO_DESCRIPTOR        0xa4
#define CEC_MSG_SET_SYSTEM_AUDIO_MODE                0x72
/* System Audio Status Operand (sys_aud_status) */
#define CEC_OP_SYS_AUD_STATUS_OFF                    0
#define CEC_OP_SYS_AUD_STATUS_ON                     1

#define CEC_MSG_SYSTEM_AUDIO_MODE_REQUEST            0x70
#define CEC_MSG_SYSTEM_AUDIO_MODE_STATUS            0x7e
/* Audio Format ID Operand (audio_format_id) */
#define CEC_OP_AUD_FMT_ID_CEA861                     0
#define CEC_OP_AUD_FMT_ID_CEA861_CXT                 1

/* Audio Rate Control Feature */
#define CEC_MSG_SET_AUDIO_RATE                        0x9a
/* Audio Rate Operand (audio_rate) */
#define CEC_OP_AUD_RATE_OFF                          0
#define CEC_OP_AUD_RATE_WIDE_STD                     1
#define CEC_OP_AUD_RATE_WIDE_FAST                    2
#define CEC_OP_AUD_RATE_WIDE_SLOW                   3
#define CEC_OP_AUD_RATE_NARROW_STD                  4
#define CEC_OP_AUD_RATE_NARROW_FAST                 5
#define CEC_OP_AUD_RATE_NARROW_SLOW                 6

/* Audio Return Channel Control Feature */
#define CEC_MSG_INITIATE_ARC                         0xc0
#define CEC_MSG_REPORT_ARC_INITIATED                0xc1
#define CEC_MSG_REPORT_ARC_TERMINATED               0xc2
#define CEC_MSG_REQUEST_ARC_INITIATION              0xc3
#define CEC_MSG_REQUEST_ARC_TERMINATION             0xc4
#define CEC_MSG_TERMINATE_ARC                       0xc5

/* Dynamic Audio Lipsync Feature */
/* Only for CEC 2.0 and up */
#define CEC_MSG_REQUEST_CURRENT_LATENCY              0xa7
#define CEC_MSG_REPORT_CURRENT_LATENCY              0xa8
/* Low Latency Mode Operand (low_latency_mode) */
#define CEC_OP_LOW_LATENCY_MODE_OFF                 0
#define CEC_OP_LOW_LATENCY_MODE_ON                  1
/* Audio Output Compensated Operand (audio_out_compensated) */
#define CEC_OP_AUD_OUT_COMPENSATED_NA                0
#define CEC_OP_AUD_OUT_COMPENSATED_DELAY            1
#define CEC_OP_AUD_OUT_COMPENSATED_NO_DELAY         2
#define CEC_OP_AUD_OUT_COMPENSATED_PARTIAL_DELAY     3

/* Capability Discovery and Control Feature */
#define CEC_MSG_CDC_MESSAGE                          0xf8
/* Ethernet-over-HDMI: nobody ever does this... */
#define CEC_MSG_CDC_HEC_INQUIRE_STATE              0x00
#define CEC_MSG_CDC_HEC_REPORT_STATE                0x01
```

```

/* HEC Functionality State Operand (hec_func_state) */
#define CEC_OP_HEC_FUNC_STATE_NOT_SUPPORTED      0
#define CEC_OP_HEC_FUNC_STATE_INACTIVE          1
#define CEC_OP_HEC_FUNC_STATE_ACTIVE            2
#define CEC_OP_HEC_FUNC_STATE_ACTIVATION_FIELD  3
/* Host Functionality State Operand (host_func_state) */
#define CEC_OP_HOST_FUNC_STATE_NOT_SUPPORTED     0
#define CEC_OP_HOST_FUNC_STATE_INACTIVE         1
#define CEC_OP_HOST_FUNC_STATE_ACTIVE           2
/* ENC Functionality State Operand (enc_func_state) */
#define CEC_OP_ENC_FUNC_STATE_EXT_CON_NOT_SUPPORTED 0
#define CEC_OP_ENC_FUNC_STATE_EXT_CON_INACTIVE    1
#define CEC_OP_ENC_FUNC_STATE_EXT_CON_ACTIVE     2
/* CDC Error Code Operand (cdc_errcode) */
#define CEC_OP_CDC_ERROR_CODE_NONE              0
#define CEC_OP_CDC_ERROR_CODE_CAP_UNSUPPORTED    1
#define CEC_OP_CDC_ERROR_CODE_WRONG_STATE       2
#define CEC_OP_CDC_ERROR_CODE_OTHER             3
/* HEC Support Operand (hec_support) */
#define CEC_OP_HEC_SUPPORT_NO                   0
#define CEC_OP_HEC_SUPPORT_YES                  1
/* HEC Activation Operand (hec_activation) */
#define CEC_OP_HEC_ACTIVATION_ON                0
#define CEC_OP_HEC_ACTIVATION_OFF               1

#define CEC_MSG_CDC_HEC_SET_STATE_ADJACENT      0x02
#define CEC_MSG_CDC_HEC_SET_STATE              0x03
/* HEC Set State Operand (hec_set_state) */
#define CEC_OP_HEC_SET_STATE_DEACTIVATE        0
#define CEC_OP_HEC_SET_STATE_ACTIVATE          1

#define CEC_MSG_CDC_HEC_REQUEST_DEACTIVATION    0x04
#define CEC_MSG_CDC_HEC_NOTIFY_ALIVE           0x05
#define CEC_MSG_CDC_HEC_DISCOVER               0x06
/* Hotplug Detect messages */
#define CEC_MSG_CDC_HPD_SET_STATE              0x10
/* HPD State Operand (hpd_state) */
#define CEC_OP_HPD_STATE_CP_EDID_DISABLE       0
#define CEC_OP_HPD_STATE_CP_EDID_ENABLE        1
#define CEC_OP_HPD_STATE_CP_EDID_DISABLE_ENABLE 2
#define CEC_OP_HPD_STATE_EDID_DISABLE          3
#define CEC_OP_HPD_STATE_EDID_ENABLE           4
#define CEC_OP_HPD_STATE_EDID_DISABLE_ENABLE   5
#define CEC_MSG_CDC_HPD_REPORT_STATE           0x11
/* HPD Error Code Operand (hpd_error) */
#define CEC_OP_HPD_ERROR_NONE                  0
#define CEC_OP_HPD_ERROR_INITIATOR_NOT_CAPABLE 1
#define CEC_OP_HPD_ERROR_INITIATOR_WRONG_STATE 2
#define CEC_OP_HPD_ERROR_OTHER                 3
#define CEC_OP_HPD_ERROR_NONE_NO_VIDEO         4

```

```
/* End of Messages */

/* Helper functions to identify the 'special' CEC devices */

static inline int cec_is_2nd_tv(const struct cec_log_addrs *las)
{
    /*
     * It is a second TV if the logical address is 14 or 15 and the
     * primary device type is a TV.
     */
    return las->num_log_addrs &&
           las->log_addr[0] >= CEC_LOG_ADDR_SPECIFIC &&
           las->primary_device_type[0] == CEC_OP_PRIM_DEVTYPE_TV;
}

static inline int cec_is_processor(const struct cec_log_addrs *las)
{
    /*
     * It is a processor if the logical address is 12-15 and the
     * primary device type is a Processor.
     */
    return las->num_log_addrs &&
           las->log_addr[0] >= CEC_LOG_ADDR_BACKUP_1 &&
           las->primary_device_type[0] == CEC_OP_PRIM_DEVTYPE_PROCESSOR;
}

static inline int cec_is_switch(const struct cec_log_addrs *las)
{
    /*
     * It is a switch if the logical address is 15 and the
     * primary device type is a Switch and the CDC-Only flag is not set.
     */
    return las->num_log_addrs == 1 &&
           las->log_addr[0] == CEC_LOG_ADDR_UNREGISTERED &&
           las->primary_device_type[0] == CEC_OP_PRIM_DEVTYPE_SWITCH &&
           !(las->flags & CEC_LOG_ADDRS_FL_CDC_ONLY);
}

static inline int cec_is_cdc_only(const struct cec_log_addrs *las)
{
    /*
     * It is a CDC-only device if the logical address is 15 and the
     * primary device type is a Switch and the CDC-Only flag is set.
     */
    return las->num_log_addrs == 1 &&
           las->log_addr[0] == CEC_LOG_ADDR_UNREGISTERED &&
           las->primary_device_type[0] == CEC_OP_PRIM_DEVTYPE_SWITCH &&
           (las->flags & CEC_LOG_ADDRS_FL_CDC_ONLY);
}

#endif
```

3.6.5 Revision and Copyright

Authors:

- Verkuil, Hans <hverkuil-cisco@xs4all.nl>
- Initial version.

Copyright © 2016 : Hans Verkuil

3.6.6 Revision History

revision 1.0.0 / 2016-03-17 (*hv*)

Initial revision

3.7 Generic Error Codes

Table 300: Generic error codes

EAGAIN (aka EWOULDBLOCK)	The ioctl can't be handled because the device is in state where it can't perform it. This could happen for example in case where device is sleeping and ioctl is performed to query statistics. It is also returned when the ioctl would need to wait for an event, but the device was opened in non-blocking mode.
EBADF	The file descriptor is not a valid.
EBUSY	The ioctl can't be handled because the device is busy. This is typically return while device is streaming, and an ioctl tried to change something that would affect the stream, or would require the usage of a hardware resource that was already allocated. The ioctl must not be retried without performing another action to fix the problem first (typically: stop the stream before retrying).
EFAULT	There was a failure while copying data from/to userspace, probably caused by an invalid pointer reference.
EINVAL	One or more of the ioctl parameters are invalid or out of the allowed range. This is a widely used error code. See the individual ioctl requests for specific causes.
ENODEV	Device not found or was removed.
ENOMEM	There's not enough memory to handle the desired operation.
ENOTTY	The ioctl is not supported by the driver, actually meaning that the required functionality is not available, or the file descriptor is not for a media device.
ENOSPC	On USB devices, the stream ioctl's can return this error, meaning that this request would overcommit the usb bandwidth reserved for periodic transfers (up to 80% of the USB bandwidth).
EPERM	Permission denied. Can be returned if the device needs write permission, or some special capabilities is needed (e. g. root)
EIO	I/O error. Typically used when there are problems communicating with a hardware device. This could indicate broken or flaky hardware. It's a 'Something is wrong, I give up!' type of error.
ENXIO	No device corresponding to this device special file exists.

Note:

1. This list is not exhaustive; ioctls may return other error codes. Since errors may have side effects such as a driver reset, applications should abort on unexpected errors, or otherwise assume that the device is in a bad state.
 2. Request-specific error codes are listed in the individual requests descriptions.
-

3.8 Glossary

Note: The goal of this section is to standardize the terms used within the media userspace API documentation. This is Work In Progress.

Bridge Driver A *Device Driver* that implements the main logic to talk with media hardware.

CEC API Consumer Electronics Control API

An API designed to receive and transmit data via an HDMI CEC interface.

See *Part V - Consumer Electronics Control API*.

Device Driver Part of the Linux Kernel that implements support for a hardware component.

Device Node A character device node in the file system used to control and transfer data in and out of a Kernel driver.

Digital TV API Previously known as DVB API

An API designed to control a subset of the *Media Hardware* that implements digital TV (e.g. DVB, ATSC, ISDB, etc).

See *Part II - Digital TV API*.

DSP Digital Signal Processor

A specialized *Microprocessor*, with its architecture optimized for the operational needs of digital signal processing.

FPGA Field-programmable Gate Array

An *IC* circuit designed to be configured by a customer or a designer after manufacturing.

See https://en.wikipedia.org/wiki/Field-programmable_gate_array.

Hardware Component A subset of the *Media Hardware*. For example an *I²C* or *SPI* device, or an *IP Block* inside an *SoC* or *FPGA*.

Hardware Peripheral A group of *hardware components* that together make a larger user-facing functional peripheral. For instance, the *SoC ISP IP Block* and the external camera sensors together make a camera hardware peripheral.

Also known as *Peripheral*.

I²C Inter-Integrated Circuit

A multi-master, multi-slave, packet switched, single-ended, serial computer bus used to control some hardware components like sub-device hardware components.

See <http://www.nxp.com/docs/en/user-guide/UM10204.pdf>.

IC Integrated circuit

A set of electronic circuits on one small flat piece of semiconductor material, normally silicon.

Also known as chip.

IP Block Intellectual property core

In electronic design a semiconductor intellectual property core, is a reusable unit of logic, cell, or integrated circuit layout design that is the intellectual property of one party. IP Blocks may be licensed to another party or can be owned and used by a single party alone.

See https://en.wikipedia.org/wiki/Semiconductor_intellectual_property_core).

ISP Image Signal Processor

A specialized processor that implements a set of algorithms for processing image data. ISPs may implement algorithms for lens shading correction, demosaicing, scaling and pixel format conversion as well as produce statistics for the use of the control algorithms (e.g. automatic exposure, white balance and focus).

Media API A set of userspace APIs used to control the media hardware. It is composed by:

- *CEC API*;
- *Digital TV API*;
- *MC API*;
- *RC API*; and
- *V4L2 API*.

See [Documentation/userspace-api/media/index.rst](#).

MC API Media Controller API

An API designed to expose and control the relationships between multimedia devices and sub-devices.

See *Part IV - Media Controller API*.

MC-centric *V4L2 Hardware* device driver that requires *MC API*.

Such drivers have `V4L2_CAP_I0_MC` `device_caps` field set (see *ioctl VIDIOC_QUERYCAP*).

See *Controlling a hardware peripheral via V4L2* for more details.

Media Hardware Subset of the hardware that is supported by the Linux Media API.

This includes audio and video capture and playback hardware, digital and analog TV, camera sensors, ISPs, remote controllers, codecs, HDMI Consumer Electronics Control, HDMI capture, etc.

Microprocessor Electronic circuitry that carries out the instructions of a computer program by performing the basic arithmetic, logical, control and input/output (I/O) operations specified by the instructions on a single integrated circuit.

Peripheral The same as *Hardware Peripheral*.

RC API Remote Controller API

An API designed to receive and transmit data from remote controllers.

See *Part III - Remote Controller API*.

SMBus A subset of I²C, which defines a stricter usage of the bus.

SPI Serial Peripheral Interface Bus

Synchronous serial communication interface specification used for short distance communication, primarily in embedded systems.

SoC System on a Chip

An integrated circuit that integrates all components of a computer or other electronic systems.

V4L2 API V4L2 userspace API

The userspace API defined in *Part I - Video for Linux API*, which is used to control a V4L2 hardware.

V4L2 Device Node A *Device Node* that is associated to a V4L driver.

The V4L2 device node naming is specified at *V4L2 Device Node Naming*.

V4L2 Hardware Part of the media hardware which is supported by the *V4L2 API*.

V4L2 Sub-device V4L2 hardware components that aren't controlled by a *Bridge Driver*. See *Sub-device Interface*.

Video-node-centric V4L2 device driver that doesn't require a media controller to be used.

Such drivers have the `V4L2_CAP_IO_MC` `device_caps` field unset (see *ioctl VID-IOC_QUERYCAP*).

V4L2 Sub-device API Part of the *V4L2 API* which control *V4L2 sub-devices*, like sensors, HDMI receivers, scalers, deinterlacers.

See *Controlling a hardware peripheral via V4L2* for more details.

3.9 GNU Free Documentation License

3.9.1 0. PREAMBLE

The purpose of this License is to make a manual, textbook, or other written document “free” in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or noncommercially. Secondly, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

This License is a kind of “copyleft”, which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

3.9.2 1. APPLICABILITY AND DEFINITIONS

This License applies to any manual or other work that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. The “Document”, below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as “you”.

A “Modified Version” of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

A “Secondary Section” is a named appendix or a front-matter section of the *Document* that deals exclusively with the relationship of the publishers or authors of the Document to the Document’s overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (For example, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

The “Invariant Sections” are certain *Secondary Sections* whose titles are designated, as being those of Invariant Sections, in the notice that says that the *Document* is released under this License.

The “Cover Texts” are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the *Document* is released under this License.

A “Transparent” copy of the *Document* means a machine-readable copy, represented in a format whose specification is available to the general public, whose contents can be viewed and edited directly and straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup has been designed to thwart or discourage subsequent modification by readers is not Transparent. A copy that is not “Transparent” is called “Opaque”.

Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTeX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML designed for human modification. Opaque formats include PostScript, PDF, proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML produced by some word processors for output purposes only.

The “Title Page” means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, “Title Page” means the text near the most prominent appearance of the work’s title, preceding the beginning of the body of the text.

3.9.3 2. VERBATIM COPYING

You may copy and distribute the *Document* in any medium, either commercially or noncommercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in *section 3*.

You may also lend copies, under the same conditions stated above, and you may publicly display copies.

3.9.4 3. COPYING IN QUANTITY

If you publish printed copies of the *Document* numbering more than 100, and the Document's license notice requires *Cover Texts*, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the *Document* and satisfy these conditions, can be treated as verbatim copying in other respects.

If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

If you publish or distribute *Opaque* copies of the *Document* numbering more than 100, you must either include a machine-readable *Transparent* copy along with each Opaque copy, or state in or with each Opaque copy a publicly-accessible computer-network location containing a complete Transparent copy of the Document, free of added material, which the general network-using public has access to download anonymously at no charge using public-standard network protocols. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

It is requested, but not required, that you contact the authors of the *Document* well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

3.9.5 4. MODIFICATIONS

You may copy and distribute a *Modified Version* of the *Document* under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

- **A.** Use in the *Title Page* (and on the covers, if any) a title distinct from that of the *Document*, and from those of previous versions (which should, if there were any, be listed in the History

section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission.

- **B.** List on the *Title Page*, as authors, one or more persons or entities responsible for authorship of the modifications in the *Modified Version*, together with at least five of the principal authors of the *Document* (all of its principal authors, if it has less than five).
- **C.** State on the *Title Page* the name of the publisher of the *Modified Version*, as the publisher.
- **D.** Preserve all the copyright notices of the *Document*.
- **E.** Add an appropriate copyright notice for your modifications adjacent to the other copyright notices.
- **F.** Include, immediately after the copyright notices, a license notice giving the public permission to use the *Modified Version* under the terms of this License, in the form shown in the Addendum below.
- **G.** Preserve in that license notice the full lists of *Invariant Sections* and required *Cover Texts* given in the *Document's* license notice.
- **H.** Include an unaltered copy of this License.
- **I.** Preserve the section entitled “History”, and its title, and add to it an item stating at least the title, year, new authors, and publisher of the *Modified Version* as given on the *Title Page*. If there is no section entitled “History” in the *Document*, create one stating the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence.
- **J.** Preserve the network location, if any, given in the *Document* for public access to a *Transparent* copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the “History” section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission.
- **K.** In any section entitled “Acknowledgements” or “Dedications”, preserve the section’s title, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein.
- **L.** Preserve all the *Invariant Sections* of the *Document*, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles.
- **M.** Delete any section entitled “Endorsements”. Such a section may not be included in the *Modified Version*.
- **N.** Do not retitle any existing section as “Endorsements” or to conflict in title with any *Invariant Section*.

If the *Modified Version* includes new front-matter sections or appendices that qualify as *Secondary Sections* and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their titles to the list of *Invariant Sections* in the Modified Version’s license notice. These titles must be distinct from any other section titles.

You may add a section entitled “Endorsements”, provided it contains nothing but endorsements of your *Modified Version* by various parties—for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a *Front-Cover Text*, and a passage of up to 25 words as a *Back-Cover Text*, to the end of the list of *Cover Texts* in the *Modified Version*. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the *Document* already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the *Document* do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any *Modified Version*.

3.9.6 5. COMBINING DOCUMENTS

You may combine the *Document* with other documents released under this License, under the terms defined in *section 4* above for modified versions, provided that you include in the combination all of the *Invariant Sections* of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice.

The combined work need only contain one copy of this License, and multiple identical *Invariant Sections* may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

In the combination, you must combine any sections entitled “History” in the various original documents, forming one section entitled “History”; likewise combine any sections entitled “Acknowledgements”, and any sections entitled “Dedications”. You must delete all sections entitled “Endorsements.”

3.9.7 6. COLLECTIONS OF DOCUMENTS

You may make a collection consisting of the *Document* and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.

You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

3.9.8 7. AGGREGATION WITH INDEPENDENT WORKS

A compilation of the *Document* or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, does not as a whole count as a *Modified Version* of the Document, provided no compilation copyright is claimed for the compilation. Such a compilation is called an “aggregate”, and this License does not apply to the other self-contained works thus compiled with the Document, on account of their being thus compiled, if they are not themselves derivative works of the Document. If the *Cover Text* requirement of *section 3* is applicable to these copies of the Document, then if the Document is less than one quarter of the entire aggregate, the Document’s Cover Texts may be placed on

covers that surround only the Document within the aggregate. Otherwise they must appear on covers around the whole aggregate.

3.9.9 8. TRANSLATION

Translation is considered a kind of modification, so you may distribute translations of the *Document* under the terms of *section 4*. Replacing *Invariant Sections* with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License provided that you also include the original English version of this License. In case of a disagreement between the translation and the original English version of this License, the original English version will prevail.

3.9.10 9. TERMINATION

You may not copy, modify, sublicense, or distribute the *Document* except as expressly provided for under this License. Any other attempt to copy, modify, sublicense or distribute the Document is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

3.9.11 10. FUTURE REVISIONS OF THIS LICENSE

The [Free Software Foundation](#) may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. See <http://www.gnu.org/copyleft/>.

Each version of the License is given a distinguishing version number. If the *Document* specifies that a particular numbered version of this License “or any later version” applies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation.

3.9.12 Addendum

To use this License in a document you have written, include a copy of the License in the document and put the following copyright and license notices just after the title page:

Copyright © YEAR YOUR NAME.

Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.1 or any later version published by the Free Software Foundation; with the *Invariant Sections* being LIST THEIR TITLES, with the *Front-Cover Texts* being LIST, and with the *Back-Cover Texts* being LIST. A copy of the license is included in the section entitled “GNU Free Documentation License”.

If you have no *Invariant Sections*, write “with no Invariant Sections” instead of saying which ones are invariant. If you have no *Front-Cover Texts*, write “no Front-Cover Texts” instead of “Front-Cover Texts being LIST”; likewise for *Back-Cover Texts*.

If your document contains nontrivial examples of program code, we recommend releasing these examples in parallel under your choice of free software license, such as the [GNU General Public License](#), to permit their use in free software.

3.10 Video4Linux (V4L) driver-specific documentation

Copyright © 1999-2016 : LinuxTV Developers

This documentation is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation version 2 of the License.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

For more details see the file COPYING in the source distribution of Linux.

3.10.1 MIPI CCS camera sensor driver

The MIPI CCS camera sensor driver is a generic driver for [MIPI CCS](#) compliant camera sensors. It exposes three sub-devices representing the pixel array, the binner and the scaler.

As the capabilities of individual devices vary, the driver exposes interfaces based on the capabilities that exist in hardware.

3.10.1.1 Pixel Array sub-device

The pixel array sub-device represents the camera sensor's pixel matrix, as well as analogue crop functionality present in many compliant devices. The analogue crop is configured using the `V4L2_SEL_TGT_CROP` on the source pad (0) of the entity. The size of the pixel matrix can be obtained by getting the `V4L2_SEL_TGT_NATIVE_SIZE` target.

3.10.1.2 Binner

The binner sub-device represents the binning functionality on the sensor. For that purpose, selection target `V4L2_SEL_TGT_COMPOSE` is supported on the sink pad (0).

Additionally, if a device has no scaler or digital crop functionality, the source pad (1) exposes another digital crop selection rectangle that can only crop at the end of the lines and frames.

3.10.1.3 Scaler

The scaler sub-device represents the digital crop and scaling functionality of the sensor. The V4L2 selection target `V4L2_SEL_TGT_CROP` is used to configure the digital crop on the sink pad (0) when digital crop is supported. Scaling is configured using selection target `V4L2_SEL_TGT_COMPOSE` on the sink pad (0) as well.

Additionally, if the scaler sub-device exists, its source pad (1) exposes another digital crop selection rectangle that can only crop at the end of the lines and frames.

3.10.1.4 Digital and analogue crop

Digital crop functionality is referred to as cropping that effectively works by dropping some data on the floor. Analogue crop, on the other hand, means that the cropped information is never retrieved. In case of camera sensors, the analogue data is never read from the pixel matrix that are outside the configured selection rectangle that designates crop. The difference has an effect in device timing and likely also in power consumption.

3.10.1.5 Private controls

The MIPI CCS driver implements a number of private controls under `V4L2_CID_USER_BASE_CCS` to control the MIPI CCS compliant camera sensors.

Analogue gain model

The CCS defines an analogue gain model where the gain can be calculated using the following formula:

$$\text{gain} = m0 * x + c0 / (m1 * x + c1)$$

Either `m0` or `c0` will be zero. The constants that are device specific, can be obtained from the following controls:

```
V4L2_CID_CCS_ANALOGUE_GAIN_M0      V4L2_CID_CCS_ANALOGUE_GAIN_M1
V4L2_CID_CCS_ANALOGUE_GAIN_C0     V4L2_CID_CCS_ANALOGUE_GAIN_C1
```

The analogue gain (`x` in the formula) is controlled through `V4L2_CID_ANALOGUE_GAIN` in this case.

Alternate analogue gain model

The CCS defines another analogue gain model called alternate analogue gain. In this case, the formula to calculate actual gain consists of linear and exponential parts:

$$\text{gain} = \text{linear} * 2 ^ \text{exponent}$$

The linear and exponent factors can be set using the `V4L2_CID_CCS_ANALOGUE_LINEAR_GAIN` and `V4L2_CID_CCS_ANALOGUE_EXPONENTIAL_GAIN` controls, respectively

Shading correction

The CCS standard supports lens shading correction. The feature can be controlled using `V4L2_CID_CCS_SHADING_CORRECTION`. Additionally, the luminance correction level may be changed using `V4L2_CID_CCS_LUMINANCE_CORRECTION_LEVEL`, where value 0 indicates no correction and 128 indicates correcting the luminance in corners to 10 % less than in the centre.

Shading correction needs to be enabled for luminance correction level to have an effect.

Copyright © 2020 Intel Corporation

3.10.2 The cx2341x driver

3.10.2.1 Non-compressed file format

The cx23416 can produce (and the cx23415 can also read) raw YUV output. The format of a YUV frame is 16x16 linear tiled NV12 (`V4L2_PIX_FMT_NV12_16L16`).

The format is YUV 4:2:0 which uses 1 Y byte per pixel and 1 U and V byte per four pixels.

The data is encoded as two macroblock planes, the first containing the Y values, the second containing UV macroblocks.

The Y plane is divided into blocks of 16x16 pixels from left to right and from top to bottom. Each block is transmitted in turn, line-by-line.

So the first 16 bytes are the first line of the top-left block, the second 16 bytes are the second line of the top-left block, etc. After transmitting this block the first line of the block on the right to the first block is transmitted, etc.

The UV plane is divided into blocks of 16x8 UV values going from left to right, top to bottom. Each block is transmitted in turn, line-by-line.

So the first 16 bytes are the first line of the top-left block and contain 8 UV value pairs (16 bytes in total). The second 16 bytes are the second line of 8 UV pairs of the top-left block, etc. After transmitting this block the first line of the block on the right to the first block is transmitted, etc.

The code below is given as an example on how to convert `V4L2_PIX_FMT_NV12_16L16` to separate Y, U and V planes. This code assumes frames of 720x576 (PAL) pixels.

The width of a frame is always 720 pixels, regardless of the actual specified width.

If the height is not a multiple of 32 lines, then the captured video is missing macroblocks at the end and is unusable. So the height must be a multiple of 32.

Raw format c example

```

#include <stdio.h>
#include <stdlib.h>
#include <string.h>

static unsigned char frame[576*720*3/2];
static unsigned char framey[576*720];
static unsigned char frameu[576*720 / 4];
static unsigned char framev[576*720 / 4];

static void de_macro_y(unsigned char* dst, unsigned char *src, int dstride, int w, int h)
{
    unsigned int y, x, i;

    // descramble Y plane
    // dstride = 720 = w
    // The Y plane is divided into blocks of 16x16 pixels
    // Each block is transmitted in turn, line-by-line.
    for (y = 0; y < h; y += 16) {
        for (x = 0; x < w; x += 16) {
            for (i = 0; i < 16; i++) {
                memcpy(dst + x + (y + i) * dstride, src, 16);
                src += 16;
            }
        }
    }
}

static void de_macro_uv(unsigned char *dstu, unsigned char *dstv, unsigned char *src,
int dstride, int w, int h)
{
    unsigned int y, x, i;

    // descramble U/V plane
    // dstride = 720 / 2 = w
    // The U/V values are interlaced (UVUV...).
    // Again, the UV plane is divided into blocks of 16x16 UV values.
    // Each block is transmitted in turn, line-by-line.
    for (y = 0; y < h; y += 16) {
        for (x = 0; x < w; x += 8) {
            for (i = 0; i < 16; i++) {
                int idx = x + (y + i) * dstride;

                dstu[idx+0] = src[0];  dstv[idx+0] = src[1];
                dstu[idx+1] = src[2];  dstv[idx+1] = src[3];
                dstu[idx+2] = src[4];  dstv[idx+2] = src[5];
                dstu[idx+3] = src[6];  dstv[idx+3] = src[7];
                dstu[idx+4] = src[8];  dstv[idx+4] = src[9];
                dstu[idx+5] = src[10]; dstv[idx+5] = src[11];
                dstu[idx+6] = src[12]; dstv[idx+6] = src[13];
                dstu[idx+7] = src[14]; dstv[idx+7] = src[15];
                src += 16;
            }
        }
    }
}

```

```
}  
  
/*****  
int main(int argc, char **argv)  
{  
FILE *fin;  
int i;  
  
if (argc == 1) fin = stdin;  
else fin = fopen(argv[1], "r");  
  
if (fin == NULL) {  
    fprintf(stderr, "cannot open input\n");  
    exit(-1);  
}  
while (fread(frame, sizeof(frame), 1, fin) == 1) {  
    de_macro_y(framey, frame, 720, 720, 576);  
    de_macro_uv(frameu, framev, frame + 720 * 576, 720 / 2, 720 / 2, 576 / 2);  
    fwrite(framey, sizeof(framey), 1, stdout);  
    fwrite(framev, sizeof(framev), 1, stdout);  
    fwrite(frameu, sizeof(frameu), 1, stdout);  
}  
fclose(fin);  
return 0;  
}
```

3.10.2.2 Format of embedded V4L2_MPEG_STREAM_VBI_FMT_IVTV VBI data

Author: Hans Verkuil <hverkuil@xs4all.nl>

This section describes the V4L2_MPEG_STREAM_VBI_FMT_IVTV format of the VBI data embedded in an MPEG-2 program stream. This format is in part dictated by some hardware limitations of the ivtv driver (the driver for the Conexant cx23415/6 chips), in particular a maximum size for the VBI data. Anything longer is cut off when the MPEG stream is played back through the cx23415.

The advantage of this format is it is very compact and that all VBI data for all lines can be stored while still fitting within the maximum allowed size.

The stream ID of the VBI data is 0xBD. The maximum size of the embedded data is $4 + 43 * 36$, which is 4 bytes for a header and $2 * 18$ VBI lines with a 1 byte header and a 42 bytes payload each. Anything beyond this limit is cut off by the cx23415/6 firmware. Besides the data for the VBI lines we also need 36 bits for a bitmask determining which lines are captured and 4 bytes for a magic cookie, signifying that this data package contains V4L2_MPEG_STREAM_VBI_FMT_IVTV VBI data. If all lines are used, then there is no longer room for the bitmask. To solve this two different magic numbers were introduced:

'itv0': After this magic number two unsigned longs follow. Bits 0-17 of the first unsigned long denote which lines of the first field are captured. Bits 18-31 of the first unsigned long and bits 0-3 of the second unsigned long are used for the second field.

'ITV0': This magic number assumes all VBI lines are captured, i.e. it implicitly implies that the bitmasks are 0xffffffff and 0xf.

After these magic cookies (and the 8 byte bitmask in case of cookie 'itv0') the captured VBI lines start:

For each line the least significant 4 bits of the first byte contain the data type. Possible values are shown in the table below. The payload is in the following 42 bytes.

Here is the list of possible data types:

```
#define IVTV_SLICED_TYPE_TELETEXT    0x1    // Teletext (uses lines 6-22 for PAL)
#define IVTV_SLICED_TYPE_CC          0x4    // Closed Captions (line 21 NTSC)
#define IVTV_SLICED_TYPE_WSS        0x5    // Wide Screen Signal (line 23 PAL)
#define IVTV_SLICED_TYPE_VPS        0x7    // Video Programming System (PAL)
↪(line 16)
```

3.10.3 Hantro video decoder driver

The Hantro video decoder driver implements the following driver-specific controls:

V4L2_CID_HANTRO_HEVC_SLICE_HEADER_SKIP (integer) Specifies to Hantro HEVC video decoder driver the number of data (in bits) to skip in the slice segment header. If non-IDR, the bits to be skipped go from syntax element “pic_output_flag” to before syntax element “slice_temporal_mvp_enabled_flag”. If IDR, the skipped bits are just “pic_output_flag” (separate_colour_plane_flag is not supported).

Note: This control is not yet part of the public kernel API and it is expected to change.

3.10.4 i.MX Video Capture Driver

3.10.4.1 Events

ipuX_csiY

This subdev can generate the following event when enabling the second IDMAC source pad:

- V4L2_EVENT_IMX_FRAME_INTERVAL_ERROR

The user application can subscribe to this event from the ipuX_csiY subdev node. This event is generated by the Frame Interval Monitor (see below for more on the FIM).

3.10.4.2 Controls

Frame Interval Monitor in ipuX_csiY

The adv718x decoders can occasionally send corrupt fields during NTSC/PAL signal re-sync (too little or too many video lines). When this happens, the IPU triggers a mechanism to re-establish vertical sync by adding 1 dummy line every frame, which causes a rolling effect from image to image, and can last a long time before a stable image is recovered. Or sometimes the mechanism doesn't work at all, causing a permanent split image (one frame contains lines from two consecutive captured images).

From experiment it was found that during image rolling, the frame intervals (elapsed time between two EOF's) drop below the nominal value for the current standard, by about one frame time (60 usec), and remain at that value until rolling stops.

While the reason for this observation isn't known (the IPU dummy line mechanism should show an increase in the intervals by 1 line time every frame, not a fixed value), we can use it to detect the corrupt fields using a frame interval monitor. If the FIM detects a bad frame interval, the `ipuX_csiY` subdev will send the event `V4L2_EVENT_IMX_FRAME_INTERVAL_ERROR`. Userland can register with the FIM event notification on the `ipuX_csiY` subdev device node. Userland can issue a streaming restart when this event is received to correct the rolling/split image.

The `ipuX_csiY` subdev includes custom controls to tweak some dials for FIM. If one of these controls is changed during streaming, the FIM will be reset and will continue at the new settings.

- `V4L2_CID_IMX_FIM_ENABLE`

Enable/disable the FIM.

- `V4L2_CID_IMX_FIM_NUM`

How many frame interval measurements to average before comparing against the nominal frame interval reported by the sensor. This can reduce noise caused by interrupt latency.

- `V4L2_CID_IMX_FIM_TOLERANCE_MIN`

If the averaged intervals fall outside nominal by this amount, in microseconds, the `V4L2_EVENT_IMX_FRAME_INTERVAL_ERROR` event is sent.

- `V4L2_CID_IMX_FIM_TOLERANCE_MAX`

If any intervals are higher than this value, those samples are discarded and do not enter into the average. This can be used to discard really high interval errors that might be due to interrupt latency from high system load.

- `V4L2_CID_IMX_FIM_NUM_SKIP`

How many frames to skip after a FIM reset or stream restart before FIM begins to average intervals.

- `V4L2_CID_IMX_FIM_ICAP_CHANNEL / V4L2_CID_IMX_FIM_ICAP_EDGE`

These controls will configure an input capture channel as the method for measuring frame intervals. This is superior to the default method of measuring frame intervals via EOF interrupt, since it is not subject to uncertainty errors introduced by interrupt latency.

Input capture requires hardware support. A VSYNC signal must be routed to one of the i.MX6 input capture channel pads.

`V4L2_CID_IMX_FIM_ICAP_CHANNEL` configures which i.MX6 input capture channel to use. This must be 0 or 1.

`V4L2_CID_IMX_FIM_ICAP_EDGE` configures which signal edge will trigger input capture events. By default the input capture method is disabled with a value of `IRQ_TYPE_NONE`. Set this control to `IRQ_TYPE_EDGE_RISING`, `IRQ_TYPE_EDGE_FALLING`, or `IRQ_TYPE_EDGE_BOTH` to enable input capture, triggered on the given signal edge(s).

When input capture is disabled, frame intervals will be measured via EOF interrupt.

File list

drivers/staging/media/imx/ include/media/imx.h include/linux/imx-media.h

Authors

- Steve Longerbeam <steve_longerbeam@mentor.com>
- Philipp Zabel <kernel@pengutronix.de>
- Russell King <linux@armlinux.org.uk>

Copyright (C) 2012-2017 Mentor Graphics Inc.

3.10.5 Maxim Integrated MAX2175 RF to bits tuner driver

The MAX2175 driver implements the following driver-specific controls:

3.10.5.1 V4L2_CID_MAX2175_I2S_ENABLE

Enable/Disable I2S output of the tuner. This is a private control that can be accessed only using the subdev interface. Refer to Documentation/driver-api/media/v4l2-controls.rst for more details.

(0)	I2S output is disabled.
(1)	I2S output is enabled.

3.10.5.2 V4L2_CID_MAX2175_HSL

The high-side/low-side (HSL) control of the tuner for a given band.

(0)	The LO frequency position is below the desired frequency.
(1)	The LO frequency position is above the desired frequency.

3.10.5.3 V4L2_CID_MAX2175_RX_MODE (menu)

The Rx mode controls a number of preset parameters of the tuner like sample clock (sck), sampling rate etc. These multiple settings are provided under one single label called Rx mode in the datasheet. The list below shows the supported modes with a brief description.

"Europe modes"	
"FM 1.2" (0)	This configures FM band with a sample rate of 0.512 million samples/sec with a 10.24 MHz sck.
"DAB 1.2" (1)	This configures VHF band with a sample rate of 2.048 million samples/sec with a 32.768 MHz sck.
"North America modes"	
"FM 1.0" (0)	This configures FM band with a sample rate of 0.7441875 million samples/sec with a 14.88375 MHz sck.
"DAB 1.2" (1)	This configures FM band with a sample rate of 0.372 million samples/sec with a 7.441875 MHz sck.

3.10.6 Vaio Picturebook Motion Eye Camera Driver

Copyright © 2001-2004 Stelian Pop <stelian@popies.net>

Copyright © 2001-2002 Alcôve <www.alcove.com>

Copyright © 2000 Andrew Tridgell <tridge@samba.org>

3.10.6.1 Private API

The driver supports frame grabbing with the video4linux API, so all video4linux tools (like xawtv) should work with this driver.

Besides the video4linux interface, the driver has a private interface for accessing the Motion Eye extended parameters (camera sharpness, agc, video framerate), the snapshot and the MJPEG capture facilities.

This interface consists of several ioctls (prototypes and structures can be found in include/linux/meye.h):

MEYEIOC_G_PARAMS and MEYEIOC_S_PARAMS Get and set the extended parameters of the motion eye camera. The user should always query the current parameters with MEYEIOC_G_PARAMS, change what he likes and then issue the MEYEIOC_S_PARAMS call (checking for -EINVAL). The extended parameters are described by the meye_params structure.

MEYEIOC_QBUF_CAPT Queue a buffer for capture (the buffers must have been obtained with a VIDIOCGMBUF call and mmap'ed by the application). The argument to MEYEIOC_QBUF_CAPT is the buffer number to queue (or -1 to end capture). The first call to MEYEIOC_QBUF_CAPT starts the streaming capture.

MEYEIOC_SYNC Takes as an argument the buffer number you want to sync. This ioctl blocks until the buffer is filled and ready for the application to use. It returns the buffer size.

MEYEIOC_STILLCAPT and MEYEIOC_STILLJCAPT Takes a snapshot in an uncompressed or compressed jpeg format. This ioctl blocks until the snapshot is done and returns (for jpeg snapshot) the size of the image. The image data is available from the first mmap'ed buffer.

Look at the 'motioneye' application code for an actual example.

3.10.7 OMAP 3 Image Signal Processor (ISP) driver

Copyright © 2010 Nokia Corporation

Copyright © 2009 Texas Instruments, Inc.

Contacts: Laurent Pinchart <laurent.pinchart@ideasonboard.com>, Sakari Ailus <sakari.ailus@iki.fi>, David Cohen <dacohen@gmail.com>

3.10.7.1 Events

The OMAP 3 ISP driver does support the V4L2 event interface on CCDC and statistics (AEWB, AF and histogram) subdevs.

The CCDC subdev produces V4L2_EVENT_FRAME_SYNC type event on HS_VS interrupt which is used to signal frame start. Earlier version of this driver used V4L2_EVENT_OMAP3ISP_HS_VS for this purpose. The event is triggered exactly when the reception of the first line of the frame starts in the CCDC module. The event can be subscribed on the CCDC subdev.

(When using parallel interface one must pay account to correct configuration of the VS signal polarity. This is automatically correct when using the serial receivers.)

Each of the statistics subdevs is able to produce events. An event is generated whenever a statistics buffer can be dequeued by a user space application using the VIDIOC_OMAP3ISP_STAT_REQ IOCTL. The events available are:

- V4L2_EVENT_OMAP3ISP_AEWB
- V4L2_EVENT_OMAP3ISP_AF
- V4L2_EVENT_OMAP3ISP_HIST

The type of the event data is struct omap3isp_stat_event_status for these ioctls. If there is an error calculating the statistics, there will be an event as usual, but no related statistics buffer. In this case omap3isp_stat_event_status.buf_err is set to non-zero.

3.10.7.2 Private IOCTLs

The OMAP 3 ISP driver supports standard V4L2 IOCTLs and controls where possible and practical. Much of the functions provided by the ISP, however, does not fall under the standard IOCTLs — gamma tables and configuration of statistics collection are examples of such.

In general, there is a private ioctl for configuring each of the blocks containing hardware-dependent functions.

The following private IOCTLs are supported:

- VIDIOC_OMAP3ISP_CCDC_CFG
- VIDIOC_OMAP3ISP_PRV_CFG
- VIDIOC_OMAP3ISP_AEWB_CFG
- VIDIOC_OMAP3ISP_HIST_CFG
- VIDIOC_OMAP3ISP_AF_CFG

- VIDIOC_OMAP3ISP_STAT_REQ
- VIDIOC_OMAP3ISP_STAT_EN

The parameter structures used by these ioctls are described in `include/linux/omap3isp.h`. The detailed functions of the ISP itself related to a given ISP block is described in the Technical Reference Manuals (TRMs) — see the end of the document for those.

While it is possible to use the ISP driver without any use of these private IOCTLs it is not possible to obtain optimal image quality this way. The AEWB, AF and histogram modules cannot be used without configuring them using the appropriate private IOCTLs.

3.10.7.3 CCDC and preview block IOCTLs

The VIDIOC_OMAP3ISP_CCDC_CFG and VIDIOC_OMAP3ISP_PRV_CFG IOCTLs are used to configure, enable and disable functions in the CCDC and preview blocks, respectively. Both IOCTLs control several functions in the blocks they control. VIDIOC_OMAP3ISP_CCDC_CFG IOCTL accepts a pointer to struct `omap3isp_ccdc_update_config` as its argument. Similarly VIDIOC_OMAP3ISP_PRV_CFG accepts a pointer to struct `omap3isp_prev_update_config`. The definition of both structures is available in¹.

The update field in the structures tells whether to update the configuration for the specific function and the flag tells whether to enable or disable the function.

The update and flag bit masks accept the following values. Each separate functions in the CCDC and preview blocks is associated with a flag (either disable or enable; part of the flag field in the structure) and a pointer to configuration data for the function.

Valid values for the update and flag fields are listed here for VIDIOC_OMAP3ISP_CCDC_CFG. Values may be or'ed to configure more than one function in the same IOCTL call.

- OMAP3ISP_CCDC_ALAW
- OMAP3ISP_CCDC_LPF
- OMAP3ISP_CCDC_BLCLAMP
- OMAP3ISP_CCDC_BCOMP
- OMAP3ISP_CCDC_FPC
- OMAP3ISP_CCDC_CULL
- OMAP3ISP_CCDC_CONFIG_LSC
- OMAP3ISP_CCDC_TBL_LSC

The corresponding values for the VIDIOC_OMAP3ISP_PRV_CFG are here:

- OMAP3ISP_PREV_LUMAENH
- OMAP3ISP_PREV_INVALIDAW
- OMAP3ISP_PREV_HRZ_MED
- OMAP3ISP_PREV_CFA
- OMAP3ISP_PREV_CHROMA_SUPP
- OMAP3ISP_PREV_WB

¹ `include/linux/omap3isp.h`

- OMAP3ISP_PREV_BLKADJ
- OMAP3ISP_PREV_RGB2RGB
- OMAP3ISP_PREV_COLOR_CONV
- OMAP3ISP_PREV_YC_LIMIT
- OMAP3ISP_PREV_DEFECT_COR
- OMAP3ISP_PREV_GAMMABYPASS
- OMAP3ISP_PREV_DRK_FRM_CAPTURE
- OMAP3ISP_PREV_DRK_FRM_SUBTRACT
- OMAP3ISP_PREV_LENS_SHADING
- OMAP3ISP_PREV_NF
- OMAP3ISP_PREV_GAMMA

The associated configuration pointer for the function may not be NULL when enabling the function. When disabling a function the configuration pointer is ignored.

3.10.7.4 Statistic blocks IOCTLS

The statistics subdevs do offer more dynamic configuration options than the other subdevs. They can be enabled, disabled and reconfigured when the pipeline is in streaming state.

The statistics blocks always get the input image data from the CCDC (as the histogram memory read isn't implemented). The statistics are dequeuable by the user from the statistics subdev nodes using private IOCTLS.

The private IOCTLS offered by the AEWB, AF and histogram subdevs are heavily reflected by the register level interface offered by the ISP hardware. There are aspects that are purely related to the driver implementation and these are discussed next.

3.10.7.5 VIDIOC_OMAP3ISP_STAT_EN

This private IOCTL enables/disables a statistic module. If this request is done before streaming, it will take effect as soon as the pipeline starts to stream. If the pipeline is already streaming, it will take effect as soon as the CCDC becomes idle.

3.10.7.6 VIDIOC_OMAP3ISP_AEWB_CFG, VIDIOC_OMAP3ISP_HIST_CFG and VIDIOC_OMAP3ISP_AF_CFG

Those IOCTLS are used to configure the modules. They require user applications to have an in-depth knowledge of the hardware. Most of the fields explanation can be found on OMAP's TRMs. The two following fields common to all the above configure private IOCTLS require explanation for better understanding as they are not part of the TRM.

omap3isp_[h3a_af/h3a_aewb/hist]_config.buf_size:

The modules handle their buffers internally. The necessary buffer size for the module's data output depends on the requested configuration. Although the driver supports reconfiguration while streaming, it does not support a reconfiguration which requires bigger buffer size than

what is already internally allocated if the module is enabled. It will return `-EBUSY` on this case. In order to avoid such condition, either disable/reconfigure/enable the module or request the necessary buffer size during the first configuration while the module is disabled.

The internal buffer size allocation considers the requested configuration's minimum buffer size and the value set on `buf_size` field. If `buf_size` field is out of [minimum, maximum] buffer size range, it's clamped to fit in there. The driver then selects the biggest value. The corrected `buf_size` value is written back to user application.

`omap3isp_[h3a_af/h3a_ewb/hist]_config.config_counter`:

As the configuration doesn't take effect synchronously to the request, the driver must provide a way to track this information to provide more accurate data. After a configuration is requested, the `config_counter` returned to user space application will be an unique value associated to that request. When user application receives an event for buffer availability or when a new buffer is requested, this `config_counter` is used to match a buffer data and a configuration.

3.10.7.7 VIDIOC_OMAP3ISP_STAT_REQ

Send to user space the oldest data available in the internal buffer queue and discards such buffer afterwards. The field `omap3isp_stat_data.frame_number` matches with the video buffer's `field_count`.

3.10.7.8 References

3.10.8 The Linux USB Video Class (UVC) driver

This file documents some driver-specific aspects of the UVC driver, such as driver-specific ioctls and implementation notes.

Questions and remarks can be sent to the Linux UVC development mailing list at linux-uvc-devel@lists.berlios.de.

3.10.8.1 Extension Unit (XU) support

Introduction

The UVC specification allows for vendor-specific extensions through extension units (XUs). The Linux UVC driver supports extension unit controls (XU controls) through two separate mechanisms:

- through mappings of XU controls to V4L2 controls
- through a driver-specific ioctl interface

The first one allows generic V4L2 applications to use XU controls by mapping certain XU controls onto V4L2 controls, which then show up during ordinary control enumeration.

The second mechanism requires `uvcvideo`-specific knowledge for the application to access XU controls but exposes the entire UVC XU concept to user space for maximum flexibility.

Both mechanisms complement each other and are described in more detail below.

Control mappings

The UVC driver provides an API for user space applications to define so-called control mappings at runtime. These allow for individual XU controls or byte ranges thereof to be mapped to new V4L2 controls. Such controls appear and function exactly like normal V4L2 controls (i.e. the stock controls, such as brightness, contrast, etc.). However, reading or writing of such a V4L2 controls triggers a read or write of the associated XU control.

The ioctl used to create these control mappings is called `UVCIOC_CTRL_MAP`. Previous driver versions (before 0.2.0) required another ioctl to be used beforehand (`UVCIOC_CTRL_ADD`) to pass XU control information to the UVC driver. This is no longer necessary as newer `uvcvideo` versions query the information directly from the device.

For details on the `UVCIOC_CTRL_MAP` ioctl please refer to the section titled “IOCTL reference” below.

3. Driver specific XU control interface

For applications that need to access XU controls directly, e.g. for testing purposes, firmware upload, or accessing binary controls, a second mechanism to access XU controls is provided in the form of a driver-specific ioctl, namely `UVCIOC_CTRL_QUERY`.

A call to this ioctl allows applications to send queries to the UVC driver that directly map to the low-level UVC control requests.

In order to make such a request the UVC unit ID of the control’s extension unit and the control selector need to be known. This information either needs to be hardcoded in the application or queried using other ways such as by parsing the UVC descriptor or, if available, using the media controller API to enumerate a device’s entities.

Unless the control size is already known it is necessary to first make a `UVC_GET_LEN` requests in order to be able to allocate a sufficiently large buffer and set the buffer size to the correct value. Similarly, to find out whether `UVC_GET_CUR` or `UVC_SET_CUR` are valid requests for a given control, a `UVC_GET_INFO` request should be made. The bits 0 (GET supported) and 1 (SET supported) of the resulting byte indicate which requests are valid.

With the addition of the `UVCIOC_CTRL_QUERY` ioctl the `UVCIOC_CTRL_GET` and `UVCIOC_CTRL_SET` ioctls have become obsolete since their functionality is a subset of the former ioctl. For the time being they are still supported but application developers are encouraged to use `UVCIOC_CTRL_QUERY` instead.

For details on the `UVCIOC_CTRL_QUERY` ioctl please refer to the section titled “IOCTL reference” below.

Security

The API doesn’t currently provide a fine-grained access control facility. The `UVCIOC_CTRL_ADD` and `UVCIOC_CTRL_MAP` ioctls require super user permissions.

Suggestions on how to improve this are welcome.

Debugging

In order to debug problems related to XU controls or controls in general it is recommended to enable the `UVC_TRACE_CONTROL` bit in the module parameter 'trace'. This causes extra output to be written into the system log.

IOCTL reference

UVCIOC_CTRL_MAP - Map a UVC control to a V4L2 control

Argument: struct `uvc_xu_control_mapping`

Description:

This ioctl creates a mapping between a UVC control or part of a UVC control and a V4L2 control. Once mappings are defined, userspace applications can access vendor-defined UVC control through the V4L2 control API.

To create a mapping, applications fill the `uvc_xu_control_mapping` structure with information about an existing UVC control defined with `UVCIOC_CTRL_ADD` and a new V4L2 control.

A UVC control can be mapped to several V4L2 controls. For instance, a UVC pan/tilt control could be mapped to separate pan and tilt V4L2 controls. The UVC control is divided into non overlapping fields using the 'size' and 'offset' fields and are then independently mapped to V4L2 control.

For signed integer V4L2 controls the `data_type` field should be set to `UVC_CTRL_DATA_TYPE_SIGNED`. Other values are currently ignored.

Return value:

On success 0 is returned. On error -1 is returned and `errno` is set appropriately.

ENOMEM Not enough memory to perform the operation.

EPERM Insufficient privileges (super user privileges are required).

EINVAL No such UVC control.

E_OVERFLOW The requested offset and size would overflow the UVC control.

EEXIST Mapping already exists.

Data types:

```
* struct uvc_xu_control_mapping
__u32  id                V4L2 control identifier
__u8   name[32]         V4L2 control name
__u8   entity[16]      UVC extension unit GUID
__u8   selector         UVC control selector
__u8   size             V4L2 control size (in bits)
__u8   offset          V4L2 control offset (in bits)
enum v4l2_ctrl_type
      v4l2_type         V4L2 control type
enum uvc_control_data_type
      data_type        UVC control data type
```

```

struct uvc_menu_info
    *menu_info      Array of menu entries (for menu controls only)
__u32  menu_count  Number of menu entries (for menu controls only)

* struct uvc_menu_info

__u32  value       Menu entry value used by the device
__u8   name[32]    Menu entry name

* enum uvc_control_data_type

UVC_CTRL_DATA_TYPE_RAW      Raw control (byte array)
UVC_CTRL_DATA_TYPE_SIGNED  Signed integer
UVC_CTRL_DATA_TYPE_UNSIGNED Unsigned integer
UVC_CTRL_DATA_TYPE_BOOLEAN Boolean
UVC_CTRL_DATA_TYPE_ENUM    Enumeration
UVC_CTRL_DATA_TYPE_BITMASK Bitmask

```

UVCIOC_CTRL_QUERY - Query a UVC XU control

Argument: struct uvc_xu_control_query

Description:

This ioctl queries a UVC XU control identified by its extension unit ID and control selector.

There are a number of different queries available that closely correspond to the low-level control requests described in the UVC specification. These requests are:

UVC_GET_CUR Obtain the current value of the control.

UVC_GET_MIN Obtain the minimum value of the control.

UVC_GET_MAX Obtain the maximum value of the control.

UVC_GET_DEF Obtain the default value of the control.

UVC_GET_RES Query the resolution of the control, i.e. the step size of the allowed control values.

UVC_GET_LEN Query the size of the control in bytes.

UVC_GET_INFO Query the control information bitmap, which indicates whether get/set requests are supported.

UVC_SET_CUR Update the value of the control.

Applications must set the 'size' field to the correct length for the control. Exceptions are the UVC_GET_LEN and UVC_GET_INFO queries, for which the size must be set to 2 and 1, respectively. The 'data' field must point to a valid writable buffer big enough to hold the indicated number of data bytes.

Data is copied directly from the device without any driver-side processing. Applications are responsible for data buffer formatting, including little-endian/big-endian conversion. This is particularly important for the result of the UVC_GET_LEN requests, which is always returned as a little-endian 16-bit integer by the device.

Return value:

On success 0 is returned. On error -1 is returned and `errno` is set appropriately.

ENOENT The device does not support the given control or the specified extension unit could not be found.

ENOBUFFS The specified buffer size is incorrect (too big or too small).

EINVAL An invalid request code was passed.

EBADRQC The given request is not supported by the given control.

EFAULT The data pointer references an inaccessible memory area.

Data types:

```
* struct uvc_xu_control_query
__u8    unit           Extension unit ID
__u8    selector       Control selector
__u8    query          Request code to send to the device
__u16   size           Control data size (in bytes)
__u8    *data          Control value
```

Copyright © 2009-2020 : LinuxTV Developers

Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.1 or any later version published by the Free Software Foundation, with no Invariant Sections. A copy of the license is included in the chapter entitled "GNU Free Documentation License".

Please notice that some documents inside the media userspace API, when explicitly mentioned on its source code, are dual-licensed with GNU Free Documentation License Version 1.1 and with the GNU General Public License:

This documentation is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

For more details see the file COPYING in the source distribution of Linux.

Symbols

__media_device_register (C function), 503
 __media_device_usb_init (C function), 506
 __media_entity_enum_init (C function), 516
 __media_entity_setup_link (C function),
 521
 __media_pipeline_start (C function), 524
 __media_pipeline_stop (C function), 524
 __media_remove_intf_link (C function), 526
 __media_remove_intf_links (C function),
 526
 __v4l2_async_nf_add_subdev (C function),
 394
 __v4l2_ctrl_grab (C function), 327
 __v4l2_ctrl_handler_setup (C function),
 322
 __v4l2_ctrl_modify_range (C function), 327
 __v4l2_ctrl_s_ctrl (C function), 329
 __v4l2_ctrl_s_ctrl_compound (C function),
 331
 __v4l2_ctrl_s_ctrl_int64 (C function), 330
 __v4l2_ctrl_s_ctrl_string (C function),
 331
 __v4l2_device_call_subdevs (C function),
 260
 __v4l2_device_call_subdevs_p (C function),
 259
 __v4l2_device_call_subdevs_until_err (C
 function), 261
 __v4l2_device_call_subdevs_until_err_p
 (C function), 260
 __v4l2_device_register_subdev_nodes (C
 function), 258
 __video_register_device (C function), 250

A

a8293_platform_data (C type), 664
 af9013_platform_data (C type), 664
 analog_demod_info (C type), 458
 analog_demod_ops (C type), 458
 analog_parameters (C type), 455

ascot2e_attach (C function), 666
 ascot2e_config (C type), 665

B

Bridge Driver, **1608**

C

can_reduce_fps (C function), 371
 ccs_pll (C type), 655
 ccs_pll_branch_bk (C type), 654
 ccs_pll_branch_fr (C type), 654
 ccs_pll_branch_limits_bk (C type), 657
 ccs_pll_branch_limits_fr (C type), 656
 ccs_pll_calculate (C function), 658
 ccs_pll_limits (C type), 657
 CEC API, **1608**
 CEC.CEC_ADAP_G_CAPS (C macro), 1553
 CEC.CEC_ADAP_G_CONNECTOR_INFO (C macro),
 1562
 CEC.CEC_ADAP_G_LOG_ADDRS (C macro), 1556
 CEC.CEC_ADAP_G_PHYS_ADDR (C macro), 1561
 CEC.CEC_ADAP_S_LOG_ADDRS (C macro), 1556
 CEC.CEC_ADAP_S_PHYS_ADDR (C macro), 1561
 CEC.cec_caps (C type), 1554
 CEC.cec_connector_info (C type), 1563
 CEC.CEC_DQEVENT (C macro), 1564
 CEC.cec_drm_connector_info (C type), 1563
 CEC.cec_event (C type), 1565
 CEC.cec_event_lost_msgs (C type), 1564
 CEC.cec_event_state_change (C type), 1564
 CEC.CEC_G_MODE (C macro), 1568
 CEC.cec_log_addrs (C type), 1556
 CEC.cec_msg (C type), 1574
 CEC.CEC_RECEIVE (C macro), 1573
 CEC.CEC_S_MODE (C macro), 1568
 CEC.CEC_TRANSMIT (C macro), 1573
 CEC.close (C function), 1551
 CEC.open (C function), 1550
 CEC.poll (C function), 1552
 cec_adap_ops (C type), 534

- cec_notifier_cec_adap_register (C function), 541
- cec_notifier_cec_adap_unregister (C function), 541
- cec_notifier_conn_register (C function), 541
- cec_notifier_conn_unregister (C function), 541
- cec_notifier_parse_hdmi_phandle (C function), 542
- cec_notifier_phys_addr_invalidate (C function), 542
- cec_notifier_set_phys_addr (C function), 542
- cec_notifier_set_phys_addr_from_edid (C function), 542
- cec_pin_allocate_adapter (C function), 540
- cec_pin_changed (C function), 540
- cec_pin_ops (C type), 539
- crc32_write_args (C type), 697
- cxd2820r_attach (C function), 667
- cxd2820r_config (C type), 666
- cxd2820r_platform_data (C type), 666
- D**
- desc_write_args (C type), 696
- Device Driver, **1608**
- Device Node, **1608**
- devm_rc_allocate_device (C function), 493
- devm_rc_register_device (C function), 493
- Digital TV API, **1608**
- dmx_buffer (C type), 1395
- dmx_buffer_flags (C type), 1394
- dmx_demux (C type), 482
- dmx_demux_caps (C type), 482
- dmx_exportbuffer (C type), 1396
- DMX_FE_ENTRY (C function), 482
- dmx_filter (C type), 1393
- dmx_frontend (C type), 481
- dmx_frontend_source (C type), 481
- dmx_input (C type), 1392
- dmx_output (C type), 1392
- dmx_pes_filter_params (C type), 1394
- dmx_requestbuffers (C type), 1395
- dmx_sct_filter_params (C type), 1393
- dmx_section_cb (C type), 480
- dmx_section_feed (C type), 479
- dmx_section_filter (C type), 478
- dmx_stc (C type), 1394
- dmx_ts_cb (C type), 479
- dmx_ts_feed (C type), 477
- dmx_ts_pes (C type), 1392
- dmxdev (C type), 471
- dmxdev_feed (C type), 469
- dmxdev_filter (C type), 470
- dmxdev_state (C type), 469
- dmxdev_type (C type), 469
- drxk_attach (C function), 668
- drxk_config (C type), 667
- DSP, **1608**
- DTV.ca.ca_caps (C type), 1416
- DTV.ca.ca_descr (C type), 1417
- DTV.ca.ca_descr_info (C type), 1416
- DTV.ca.CA_GET_CAP (C macro), 1420
- DTV.ca.CA_GET_DESCR_INFO (C macro), 1422
- DTV.ca.CA_GET_MSG (C macro), 1422
- DTV.ca.CA_GET_SLOT_INFO (C macro), 1421
- DTV.ca.ca_msg (C type), 1417
- DTV.ca.CA_RESET (C macro), 1419
- DTV.ca.CA_SEND_MSG (C macro), 1423
- DTV.ca.CA_SET_DESCR (C macro), 1424
- DTV.ca.ca_slot_info (C type), 1415
- DTV.ca.close (C function), 1419
- DTV.ca.open (C function), 1418
- DTV.dmx.close (C function), 1398
- DTV.dmx.DMX_ADD_PID (C macro), 1409
- DTV.dmx.DMX_DQBUF (C macro), 1414
- DTV.dmx.DMX_EXPBUF (C macro), 1413
- DTV.dmx.DMX_GET_PES_PIDS (C macro), 1408
- DTV.dmx.DMX_GET_STC (C macro), 1407
- DTV.dmx.DMX_QBUF (C macro), 1414
- DTV.dmx.DMX_QUERYBUF (C macro), 1412
- DTV.dmx.DMX_REMOVE_PID (C macro), 1409
- DTV.dmx.DMX_REQBUFS (C macro), 1410
- DTV.dmx.DMX_SET_BUFFER_SIZE (C macro), 1406
- DTV.dmx.DMX_SET_FILTER (C macro), 1404
- DTV.dmx.DMX_SET_PES_FILTER (C macro), 1405
- DTV.dmx.DMX_START (C macro), 1403
- DTV.dmx.DMX_STOP (C macro), 1404
- DTV.dmx.mmap (C function), 1401
- DTV.dmx.munmap (C function), 1402
- DTV.dmx.open (C function), 1397
- DTV.dmx.read (C function), 1398
- DTV.dmx.write (C function), 1400
- DTV.fe.atscmh_rs_code_mode (C type), 1377
- DTV.fe.atscmh_rs_frame_ensemble (C type), 1376
- DTV.fe.atscmh_rs_frame_mode (C type), 1376
- DTV.fe.atscmh_sccc_block_mode (C type), 1376

- DTV.fe.atscmh_sccc_code_mode (C type), 1376
- DTV.fe.close (C function), 1381
- DTV.fe.dtv_fe_stats (C type), 1378
- DTV.fe.dtv_properties (C type), 1379
- DTV.fe.dtv_property (C type), 1378
- DTV.fe.dtv_stats (C type), 1377
- DTV.fe.dvb_diseqc_master_cmd (C type), 1369
- DTV.fe.dvb_diseqc_slave_reply (C type), 1370
- DTV.fe.dvb_frontend_info (C type), 1369
- DTV.fe.fe_caps (C type), 1368
- DTV.fe.fe_code_rate (C type), 1371
- DTV.fe.fe_delivery_system (C type), 1374
- DTV.fe.FE_DISEQC_RECV_SLAVE_REPLY (C macro), 1386
- DTV.fe.FE_DISEQC_RESET_OVERLOAD (C macro), 1385
- DTV.fe.FE_DISEQC_SEND_BURST (C macro), 1387
- DTV.fe.FE_DISEQC_SEND_MASTER_CMD (C macro), 1386
- DTV.fe.FE_DISHNETWORK_SEND_LEGACY_CMD (C macro), 1440
- DTV.fe.FE_ENABLE_HIGH_LNB_VOLTAGE (C macro), 1390
- DTV.fe.FE_GET_EVENT (C macro), 1440
- DTV.fe.FE_GET_FRONTEND (C macro), 1439
- DTV.fe.FE_GET_INFO (C macro), 1382
- DTV.fe.FE_GET_PROPERTY (C macro), 1384
- DTV.fe.fe_guard_interval (C type), 1373
- DTV.fe.fe_hierarchy (C type), 1373
- DTV.fe.fe_interleaving (C type), 1374
- DTV.fe.fe_modulation (C type), 1372
- DTV.fe.fe_pilot (C type), 1374
- DTV.fe.FE_READ_BER (C macro), 1434
- DTV.fe.FE_READ_SIGNAL_STRENGTH (C macro), 1436
- DTV.fe.FE_READ_SNR (C macro), 1435
- DTV.fe.FE_READ_STATUS (C macro), 1383
- DTV.fe.FE_READ_UNCORRECTED_BLOCKS (C macro), 1437
- DTV.fe.fe_rolloff (C type), 1374
- DTV.fe.fe_sec_mini_cmd (C type), 1371
- DTV.fe.fe_sec_tone_mode (C type), 1370
- DTV.fe.fe_sec_voltage (C type), 1370
- DTV.fe.FE_SET_FRONTEND (C macro), 1438
- DTV.fe.FE_SET_FRONTEND_TUNE_MODE (C macro), 1391
- DTV.fe.FE_SET_PROPERTY (C macro), 1384
- DTV.fe.FE_SET_TONE (C macro), 1388
- DTV.fe.FE_SET_VOLTAGE (C macro), 1389
- DTV.fe.fe_spectral_inversion (C type), 1371
- DTV.fe.fe_status (C type), 1371
- DTV.fe.fe_transmit_mode (C type), 1372
- DTV.fe.fe_type (C type), 1431
- DTV.fe.fecap_scale_params (C type), 1377
- DTV.fe.isdbt_layer_interleaving_table (C type), 1352
- DTV.fe.open (C function), 1380
- DTV.net.dvb_net_if (C type), 1428
- DTV.net.NET_ADD_IF (C macro), 1428
- DTV.net.NET_GET_IF (C macro), 1430
- DTV.net.NET_REMOVE_IF (C macro), 1429
- dtv_frontend_properties (C type), 462
- dvb_adapter (C type), 431
- dvb_attach (C function), 437
- dvb_buf_type (C type), 443
- dvb_buffer (C type), 443
- dvb_ca_en50221 (C type), 485
- dvb_ca_en50221_camchange_irq (C function), 486
- dvb_ca_en50221_camready_irq (C function), 486
- dvb_ca_en50221_frda_irq (C function), 486
- dvb_ca_en50221_init (C function), 486
- dvb_ca_en50221_release (C function), 486
- dvb_create_media_graph (C function), 435
- dvb_demux (C type), 474
- dvb_demux_feed (C type), 473
- dvb_demux_filter (C type), 472
- dvb_detach (C function), 437
- dvb_device (C type), 432
- dvb_device_type (C type), 431
- dvb_dmx_filter_type (C type), 472
- dvb_dmx_init (C function), 475
- dvb_dmx_release (C function), 476
- dvb_dmx_state (C type), 472
- dvb_dmx_swfilter (C function), 476
- dvb_dmx_swfilter_204 (C function), 477
- dvb_dmx_swfilter_packets (C function), 476
- dvb_dmx_swfilter_raw (C function), 477
- dvb_dmxdev_init (C function), 472
- dvb_dmxdev_release (C function), 472
- dvb_free_device (C function), 434
- dvb_frontend (C type), 465
- dvb_frontend_detach (C function), 466
- dvb_frontend_event (C type), 1434
- dvb_frontend_internal_info (C type), 459
- dvb_frontend_ops (C type), 460

- dvb_frontend_parameters (C type), 1432
 - dvb_frontend_reinitialise (C function), 467
 - dvb_frontend_resume (C function), 467
 - dvb_frontend_sleep_until (C function), 467
 - dvb_frontend_suspend (C function), 466
 - dvb_frontend_tune_settings (C type), 455
 - dvb_generic_ioctl (C function), 436
 - dvb_generic_open (C function), 435
 - dvb_generic_release (C function), 435
 - dvb_get_media_controller (C function), 435
 - dvb_module_probe (C function), 436
 - dvb_module_release (C function), 437
 - dvb_net (C type), 487
 - dvb_net_init (C function), 487
 - dvb_net_release (C function), 487
 - dvb_ofdm_parameters (C type), 1433
 - dvb_pll_attach (C function), 668
 - dvb_qam_parameters (C type), 1433
 - dvb_qpsk_parameters (C type), 1433
 - dvb_register_adapter (C function), 433
 - dvb_register_device (C function), 434
 - dvb_register_frontend (C function), 466
 - dvb_register_media_controller (C function), 435
 - dvb_remove_device (C function), 434
 - dvb_ringbuffer (C type), 438
 - dvb_ringbuffer_avail (C function), 439
 - dvb_ringbuffer_empty (C function), 439
 - dvb_ringbuffer_flush (C function), 439
 - dvb_ringbuffer_flush_spinlock_wakeup (C function), 440
 - dvb_ringbuffer_free (C function), 439
 - dvb_ringbuffer_init (C function), 439
 - DVB_RINGBUFFER_PEEK (C function), 440
 - dvb_ringbuffer_pkt_dispose (C function), 442
 - dvb_ringbuffer_pkt_next (C function), 442
 - dvb_ringbuffer_pkt_read (C function), 442
 - dvb_ringbuffer_pkt_read_user (C function), 442
 - dvb_ringbuffer_pkt_write (C function), 441
 - dvb_ringbuffer_read (C function), 440
 - dvb_ringbuffer_read_user (C function), 440
 - dvb_ringbuffer_reset (C function), 439
 - DVB_RINGBUFFER_SKIP (C function), 440
 - dvb_ringbuffer_write (C function), 441
 - DVB_RINGBUFFER_WRITE_BYTE (C function), 441
 - dvb_ringbuffer_write_user (C function), 441
 - dvb_tuner_info (C type), 455
 - dvb_tuner_ops (C type), 457
 - dvb_unregister_adapter (C function), 434
 - dvb_unregister_device (C function), 434
 - dvb_unregister_frontend (C function), 466
 - dvb_usercopy (C function), 436
 - dvb_vb2_ctx (C type), 444
 - dvb_vb2_dqbuf (C function), 447
 - dvb_vb2_expbuf (C function), 446
 - dvb_vb2_fill_buffer (C function), 445
 - dvb_vb2_init (C function), 444
 - dvb_vb2_is_streaming (C function), 445
 - dvb_vb2_mmap (C function), 447
 - dvb_vb2_poll (C function), 445
 - dvb_vb2_qbuf (C function), 446
 - dvb_vb2_querybuf (C function), 446
 - dvb_vb2_release (C function), 445
 - dvb_vb2_reqbufs (C function), 446
 - dvb_vb2_states (C type), 443
 - dvb_vb2_stream_off (C function), 446
 - dvb_vb2_stream_on (C function), 445
 - dvb_vsb_parameters (C type), 1433
 - dvbfe_algo (C type), 456
 - dvbfe_search (C type), 456
- ## F
- fe_bandwidth (C type), 1432
 - FPGA, **1608**
- ## G
- gobj_to_entity (C function), 517
 - gobj_to_intf (C function), 518
 - gobj_to_link (C function), 518
 - gobj_to_pad (C function), 517
- ## H
- Hardware Component, **1608**
 - Hardware Peripheral, **1608**
 - header_write_args (C type), 697
 - helene_attach (C function), 669
 - helene_attach_s (C function), 669
 - helene_config (C type), 668
 - horus3a_attach (C function), 670
 - horus3a_config (C type), 669
- ## I
- I²C, **1608**
 - IC, **1609**
 - intf_to_devnode (C function), 518
 - intlog10 (C function), 430
 - intlog2 (C function), 430
 - IP Block, **1609**

- ipu3_uapi_4a_config (C type), 874
- ipu3_uapi_acc_param (C type), 915
- ipu3_uapi_ae_ccm (C type), 869
- ipu3_uapi_ae_config (C type), 870
- ipu3_uapi_ae_grid_config (C type), 868
- ipu3_uapi_ae_raw_buffer (C type), 867
- ipu3_uapi_ae_raw_buffer_aligned (C type), 868
- ipu3_uapi_ae_weight_elem (C type), 869
- ipu3_uapi_af_config_s (C type), 873
- ipu3_uapi_af_filter_config (C type), 870
- ipu3_uapi_af_raw_buffer (C type), 873
- ipu3_uapi_anr_config (C type), 915
- ipu3_uapi_anr_stitch_config (C type), 914
- ipu3_uapi_anr_stitch_pyramid (C type), 914
- ipu3_uapi_anr_transform_config (C type), 913
- ipu3_uapi_awb_config (C type), 867
- ipu3_uapi_awb_config_s (C type), 867
- ipu3_uapi_awb_fr_config_s (C type), 874
- ipu3_uapi_awb_fr_raw_buffer (C type), 873
- ipu3_uapi_awb_raw_buffer (C type), 866
- ipu3_uapi_bnr_static_config (C type), 881
- ipu3_uapi_bnr_static_config_bp_ctrl_config (C type), 879
- ipu3_uapi_bnr_static_config_dn_detect_ctrl_config (C type), 879
- ipu3_uapi_bnr_static_config_green_disparity (C type), 881
- ipu3_uapi_bnr_static_config_lut_config (C type), 878
- ipu3_uapi_bnr_static_config_opt_center_config (C type), 878
- ipu3_uapi_bnr_static_config_opt_center_sqrcntrl_config (C type), 880
- ipu3_uapi_bnr_static_config_thr_coeffs_config (C type), 877
- ipu3_uapi_bnr_static_config_thr_ctrl_shd_config (C type), 878
- ipu3_uapi_bnr_static_config_wb_gains_config (C type), 876
- ipu3_uapi_bnr_static_config_wb_gains_thr_config (C type), 877
- ipu3_uapi_bubble_info (C type), 874
- ipu3_uapi_ccm_mat_config (C type), 883
- ipu3_uapi_cds_params (C type), 886
- ipu3_uapi_csc_mat_config (C type), 885
- ipu3_uapi_cu_cfg0 (C type), 900
- ipu3_uapi_cu_cfg1 (C type), 901
- ipu3_uapi_dm_config (C type), 882
- ipu3_uapi_far_w (C type), 896
- ipu3_uapi_ff_status (C type), 875
- ipu3_uapi_flags (C type), 920
- ipu3_uapi_gamma_config (C type), 885
- ipu3_uapi_gamma_corr_ctrl (C type), 884
- ipu3_uapi_gamma_corr_lut (C type), 885
- ipu3_uapi_grid_config (C type), 866
- ipu3_uapi_iefd_cux2 (C type), 890
- ipu3_uapi_iefd_cux2_1 (C type), 892
- ipu3_uapi_iefd_cux4 (C type), 892
- ipu3_uapi_iefd_cux6_ed (C type), 890
- ipu3_uapi_iefd_cux6_rad (C type), 893
- ipu3_uapi_isp_lin_vmem_params (C type), 916
- ipu3_uapi_isp_tnr3_params (C type), 917
- ipu3_uapi_isp_tnr3_vmem_params (C type), 917
- ipu3_uapi_isp_xnr3_params (C type), 919
- ipu3_uapi_isp_xnr3_vmem_params (C type), 918
- ipu3_uapi_obgrid_param (C type), 919
- ipu3_uapi_params (C type), 922
- ipu3_uapi_rad_far_w (C type), 900
- ipu3_uapi_radial_cfg (C type), 899
- ipu3_uapi_radial_reset_x2 (C type), 899
- ipu3_uapi_radial_reset_xy (C type), 898
- ipu3_uapi_radial_reset_y2 (C type), 899
- ipu3_uapi_sharp_cfg (C type), 896
- ipu3_uapi_shd_black_level_config (C type), 888
- ipu3_uapi_shd_config (C type), 889
- ipu3_uapi_shd_config_static (C type), 888
- ipu3_uapi_shd_general_config (C type), 888
- ipu3_uapi_shd_grid_config (C type), 887
- ipu3_uapi_shd_lut (C type), 889
- ipu3_uapi_stats_3a (C type), 876
- ipu3_uapi_unsharp_cfg (C type), 897
- ipu3_uapi_unsharp_coef0 (C type), 897
- ipu3_uapi_unsharp_coef1 (C type), 898
- ipu3_uapi_vss_lut_x (C type), 901
- ipu3_uapi_vss_lut_y (C type), 902
- ipu3_uapi_xnr3_alpha_params (C type), 918
- ipu3_uapi_xnr3_blending_params (C type), 919
- ipu3_uapi_xnr3_coring_params (C type), 918
- ipu3_uapi_yuvp1_chnr_config (C type), 906
- ipu3_uapi_yuvp1_chnr_coring_config (C type), 904
- ipu3_uapi_yuvp1_chnr_enable_config (C type), 904
- ipu3_uapi_yuvp1_chnr_iir_fir_config (C type), 905

- ipu3_uapi_yuvp1_chnr_sense_gain_config (C type), 905
- ipu3_uapi_yuvp1_iefd_cfg_units (C type), 894
- ipu3_uapi_yuvp1_iefd_config (C type), 902
- ipu3_uapi_yuvp1_iefd_config_s (C type), 895
- ipu3_uapi_yuvp1_iefd_control (C type), 895
- ipu3_uapi_yuvp1_iefd_rad_cfg (C type), 901
- ipu3_uapi_yuvp1_iefd_shrp_cfg (C type), 897
- ipu3_uapi_yuvp1_iefd_unshrp_cfg (C type), 898
- ipu3_uapi_yuvp1_iefd_vssnlm_cfg (C type), 902
- ipu3_uapi_yuvp1_y_ee_nr_clip_config (C type), 908
- ipu3_uapi_yuvp1_y_ee_nr_config (C type), 910
- ipu3_uapi_yuvp1_y_ee_nr_diag_config (C type), 909
- ipu3_uapi_yuvp1_y_ee_nr_fc_coring_config (C type), 909
- ipu3_uapi_yuvp1_y_ee_nr_frng_config (C type), 908
- ipu3_uapi_yuvp1_y_ee_nr_gain_config (C type), 907
- ipu3_uapi_yuvp1_y_ee_nr_lpf_config (C type), 906
- ipu3_uapi_yuvp1_y_ee_nr_sense_config (C type), 906
- ipu3_uapi_yuvp1_yds_config (C type), 903
- ipu3_uapi_yuvp2_tcc_gain_pawl_lut_static_config (C type), 912
- ipu3_uapi_yuvp2_tcc_gen_control_static_config (C type), 910
- ipu3_uapi_yuvp2_tcc_inv_y_lut_static_config (C type), 912
- ipu3_uapi_yuvp2_tcc_macc_elem_static_config (C type), 911
- ipu3_uapi_yuvp2_tcc_macc_table_static_config (C type), 912
- ipu3_uapi_yuvp2_tcc_r_sqr_lut_static_config (C type), 912
- ipu3_uapi_yuvp2_tcc_static_config (C type), 912
- is_media_entity_v4l2_subdev (C function), 516
- is_media_entity_v4l2_video_device (C function), 515
- ISP, 1609
- ix2505v_attach (C function), 670
- ix2505v_config (C type), 670
- ## L
- lirc_fh (C type), 489
- ## M
- m88ds3103_attach (C function), 673
- m88ds3103_clock_out (C type), 671
- m88ds3103_config (C type), 672
- m88ds3103_platform_data (C type), 671
- m88ds3103_ts_mode (C type), 671
- mb86a20s_attach (C function), 673
- mb86a20s_config (C type), 673
- MC API, 1609
- MC.close (C function), 1522
- MC.lirc_scancode (C type), 1489
- MC.media_device_info (C type), 1524
- MC.media_entity_desc (C type), 1529
- MC.MEDIA_IOC_DEVICE_INFO (C macro), 1524
- MC.MEDIA_IOC_ENUM_ENTITIES (C macro), 1529
- MC.MEDIA_IOC_ENUM_LINKS (C macro), 1531
- MC.MEDIA_IOC_G_TOPOLOGY (C macro), 1526
- MC.MEDIA_IOC_REQUEST_ALLOC (C macro), 1534
- MC.MEDIA_IOC_SETUP_LINK (C macro), 1533
- MC.media_link_desc (C type), 1532
- MC.media_links_enum (C type), 1531
- MC.media_pad_desc (C type), 1532
- MC.MEDIA_REQUEST_IOC_QUEUE (C macro), 1537
- MC.MEDIA_REQUEST_IOC_REINIT (C macro), 1539
- MC.media_v2_entity (C type), 1527
- MC.media_v2_interface (C type), 1527
- MC.media_v2_intf_devnode (C type), 1528
- MC.media_v2_link (C type), 1528
- MC.media_v2_pad (C type), 1528
- MC.media_v2_topology (C type), 1526
- MC.open (C function), 1521
- MC.poll (C function), 1536
- MC.rc_proto (C type), 1490
- MC.request.close (C function), 1535
- MC-centric, 1609
- Media API, 1609
- Media Hardware, 1609
- media_create_intf_link (C function), 525
- media_create_pad_link (C function), 519
- media_create_pad_links (C function), 520
- media_device (C type), 501
- media_device_cleanup (C function), 503

- [media_device_delete \(C function\), 532](#)
[media_device_init \(C function\), 503](#)
[media_device_ops \(C type\), 500](#)
[media_device_pci_init \(C function\), 506](#)
[media_device_register \(C function\), 504](#)
[media_device_register_entity \(C function\), 504](#)
[media_device_register_entity_notify \(C function\), 505](#)
[media_device_unregister \(C function\), 504](#)
[media_device_unregister_entity \(C function\), 505](#)
[media_device_unregister_entity_notify \(C function\), 505](#)
[media_device_usb_allocate \(C function\), 532](#)
[media_device_usb_init \(C function\), 506](#)
[media_devnode \(C type\), 507](#)
[media_devnode_create \(C function\), 525](#)
[media_devnode_data \(C function\), 508](#)
[media_devnode_is_registered \(C function\), 509](#)
[media_devnode_register \(C function\), 508](#)
[media_devnode_remove \(C function\), 525](#)
[media_devnode_unregister \(C function\), 508](#)
[media_devnode_unregister_prepare \(C function\), 508](#)
[media_entity \(C type\), 513](#)
[media_entity_call \(C function\), 527](#)
[media_entity_cleanup \(C function\), 519](#)
[media_entity_enum \(C type\), 509](#)
[media_entity_enum_cleanup \(C function\), 516](#)
[media_entity_enum_clear \(C function\), 516](#)
[media_entity_enum_empty \(C function\), 517](#)
[media_entity_enum_init \(C function\), 502](#)
[media_entity_enum_intersects \(C function\), 517](#)
[media_entity_enum_set \(C function\), 516](#)
[media_entity_enum_test \(C function\), 516](#)
[media_entity_enum_test_and_set \(C function\), 517](#)
[media_entity_enum_zero \(C function\), 516](#)
[media_entity_find_link \(C function\), 522](#)
[media_entity_get_fwnode_pad \(C function\), 522](#)
[media_entity_id \(C function\), 515](#)
[media_entity_notify \(C type\), 500](#)
[media_entity_operations \(C type\), 512](#)
[media_entity_pads_init \(C function\), 518](#)
[media_entity_remote_pad \(C function\), 522](#)
[media_entity_remove_links \(C function\), 521](#)
[media_entity_setup_link \(C function\), 521](#)
[media_entity_to_v4l2_subdev \(C function\), 290](#)
[media_entity_to_video_device \(C function\), 250](#)
[media_entity_type \(C type\), 512](#)
[media_file_operations \(C type\), 506](#)
[media_get_pad_index \(C function\), 519](#)
[media_gobj \(C type\), 509](#)
[media_gobj_create \(C function\), 518](#)
[media_gobj_destroy \(C function\), 518](#)
[media_gobj_gen_id \(C function\), 515](#)
[media_gobj_type \(C type\), 509](#)
[media_graph \(C type\), 510](#)
[media_graph_walk_cleanup \(C function\), 523](#)
[media_graph_walk_init \(C function\), 523](#)
[media_graph_walk_next \(C function\), 523](#)
[media_graph_walk_start \(C function\), 523](#)
[media_id \(C function\), 515](#)
[media_interface \(C type\), 514](#)
[media_intf_devnode \(C type\), 515](#)
[media_link \(C type\), 510](#)
[media_pad \(C type\), 512](#)
[media_pad_signal_type \(C type\), 511](#)
[media_pipeline \(C type\), 510](#)
[media_pipeline_start \(C function\), 524](#)
[media_pipeline_stop \(C function\), 524](#)
[media_remove_intf_link \(C function\), 526](#)
[media_remove_intf_links \(C function\), 526](#)
[media_request \(C type\), 527](#)
[media_request_alloc \(C function\), 529](#)
[media_request_get \(C function\), 529](#)
[media_request_get_by_fd \(C function\), 529](#)
[media_request_lock_for_access \(C function\), 528](#)
[media_request_lock_for_update \(C function\), 528](#)
[media_request_object \(C type\), 530](#)
[media_request_object_bind \(C function\), 531](#)
[media_request_object_complete \(C function\), 532](#)
[media_request_object_find \(C function\), 531](#)
[media_request_object_get \(C function\), 530](#)
[media_request_object_init \(C function\), 531](#)
[media_request_object_ops \(C type\), 529](#)
[media_request_object_put \(C function\), 530](#)

- media_request_object_unbind (C function), 532
- media_request_put (C function), 529
- media_request_state (C type), 527
- media_request_unlock_for_access (C function), 528
- media_request_unlock_for_update (C function), 528
- media_type (C function), 515
- Microprocessor, **1609**
- mn88472_config (C type), 674
- ## N
- null_packet_write_args (C type), 706
- ## P
- param_type (C type), 409
- parse_endpoint_func (C type), 405
- pcr_write_args (C type), 706
- Peripheral, **1609**
- pes_header_write_args (C type), 693
- pes_ts_header_write_args (C type), 694
- pes_write_args (C type), 694
- psi_write_args (C type), 695
- ## R
- RC API, **1609**
- RC.LIRC_GET_FEATURES (C macro), 1494
- RC.LIRC_GET_MAX_TIMEOUT (C macro), 1499
- RC.LIRC_GET_MIN_TIMEOUT (C macro), 1499
- RC.LIRC_GET_REC_MODE (C macro), 1497
- RC.LIRC_GET_REC_RESOLUTION (C macro), 1498
- RC.LIRC_GET_REC_TIMEOUT (C macro), 1500
- RC.LIRC_GET_SEND_MODE (C macro), 1496
- RC.LIRC_SET_MEASURE_CARRIER_MODE (C macro), 1505
- RC.LIRC_SET_REC_CARRIER (C macro), 1501
- RC.LIRC_SET_REC_CARRIER_RANGE (C macro), 1502
- RC.LIRC_SET_REC_MODE (C macro), 1497
- RC.LIRC_SET_REC_TIMEOUT (C macro), 1500
- RC.LIRC_SET_REC_TIMEOUT_REPORTS (C macro), 1504
- RC.LIRC_SET_SEND_CARRIER (C macro), 1502
- RC.LIRC_SET_SEND_DUTY_CYCLE (C macro), 1498
- RC.LIRC_SET_SEND_MODE (C macro), 1496
- RC.LIRC_SET_TRANSMITTER_MASK (C macro), 1503
- RC.LIRC_SET_WIDEBAND_RECEIVER (C macro), 1505
- RC.read (C function), 1491
- RC.write (C function), 1492
- rc_allocate_device (C function), 493
- rc_dev (C type), 490
- rc_driver_type (C type), 489
- rc_filter_type (C type), 489
- rc_free_device (C function), 493
- rc_map (C type), 494
- rc_map_get (C function), 495
- rc_map_list (C type), 494
- rc_map_register (C function), 495
- rc_map_table (C type), 494
- rc_map_unregister (C function), 495
- rc_register_device (C function), 493
- rc_scancode_filter (C type), 489
- rc_unregister_device (C function), 494
- rkisp1_cif_isp_ae_stat (C type), 938
- rkisp1_cif_isp_aec_config (C type), 932
- rkisp1_cif_isp_af_meas_val (C type), 938
- rkisp1_cif_isp_af_stat (C type), 939
- rkisp1_cif_isp_afc_config (C type), 932
- rkisp1_cif_isp_awb_gain_config (C type), 929
- rkisp1_cif_isp_awb_meas (C type), 937
- rkisp1_cif_isp_awb_meas_config (C type), 929
- rkisp1_cif_isp_awb_stat (C type), 937
- rkisp1_cif_isp_bdm_config (C type), 931
- rkisp1_cif_isp_bls_config (C type), 924
- rkisp1_cif_isp_bls_fixed_val (C type), 924
- rkisp1_cif_isp_bls_meas_val (C type), 938
- rkisp1_cif_isp_cproc_config (C type), 928
- rkisp1_cif_isp_ctk_config (C type), 931
- rkisp1_cif_isp_dpcc_config (C type), 925
- rkisp1_cif_isp_dpcc_methods_config (C type), 925
- rkisp1_cif_isp_dpf_config (C type), 935
- rkisp1_cif_isp_dpf_gflt (C type), 934
- rkisp1_cif_isp_dpf_gain (C type), 934
- rkisp1_cif_isp_dpf_gain_usage (C type), 933
- rkisp1_cif_isp_dpf_nll (C type), 933
- rkisp1_cif_isp_dpf_nll_scale_mode (C type), 933
- rkisp1_cif_isp_dpf_rb_filtersize (C type), 933
- rkisp1_cif_isp_dpf_rbflt (C type), 934
- rkisp1_cif_isp_dpf_strength_config (C type), 935
- rkisp1_cif_isp_exp_ctrl_autostop (C type), 923

- rkisp1_cif_isp_exp_meas_mode (C type), 923
rkisp1_cif_ispflt_config (C type), 930
rkisp1_cif_isp_gamma_corr_curve (C type), 926
rkisp1_cif_isp_gamma_curve_x_axis_pnts (C type), 926
rkisp1_cif_isp_goc_config (C type), 931
rkisp1_cif_isp_hist_stat (C type), 939
rkisp1_cif_isp_hst_config (C type), 932
rkisp1_cif_isp_ie_config (C type), 928
rkisp1_cif_isp_isp_meas_cfg (C type), 936
rkisp1_cif_isp_isp_other_cfg (C type), 935
rkisp1_cif_isp_lsc_config (C type), 927
rkisp1_cif_isp_sdg_config (C type), 927
rkisp1_cif_isp_stat (C type), 940
rkisp1_cif_isp_version (C type), 923
rkisp1_cif_isp_window (C type), 924
rkisp1_params_cfg (C type), 937
rkisp1_stat_buffer (C type), 940
rtl2830_platform_data (C type), 674
rtl2832_platform_data (C type), 675
rtl2832_sdr_platform_data (C type), 675
- ## S
- SMBus, 1610
SoC, 1610
SPI, 1610
stb6000_attach (C function), 676
- ## T
- tda10071_platform_data (C type), 676
tda826x_attach (C function), 677
to_v4l2_subdev_fh (C function), 290
to_video_device (C function), 250
ts_filter_type (C type), 477
tuner_mode (C type), 408
tuner_params (C type), 410
tuner_range (C type), 409
tuner_setup (C type), 408
tunertype (C type), 412
tveeprom (C type), 428
tveeprom_audio_processor (C type), 428
tveeprom_hauppage_analog (C function), 429
tveeprom_read (C function), 429
- ## V
- V4L.close (C function), 1095
V4L.mmap (C function), 1246
V4L.munmap (C function), 1247
V4L.open (C function), 1248
V4L.poll (C function), 1249
V4L.read (C function), 1251
V4L.select (C function), 1252
V4L.v4l2_audio (C type), 1147
V4L.v4l2_audioout (C type), 1148
V4L.v4l2_bt_timings (C type), 1153
V4L.v4l2_bt_timings_cap (C type), 1114
V4L.v4l2_buf_type (C type), 976
V4L.v4l2_buffer (C type), 972
V4L.v4l2_capability (C type), 1209
V4L.v4l2_captureparm (C type), 1185
V4L.v4l2_clip (C type), 989
V4L.v4l2_close (C function), 1065
V4L.v4l2_control (C type), 1151
V4L.v4l2_create_buffers (C type), 1097
V4L.v4l2_crop (C type), 1150
V4L.v4l2_cropcap (C type), 1099
V4L.v4l2_ctrl_hdr10_cll_info (C type), 806
V4L.v4l2_ctrl_hdr10_mastering_display (C type), 806
V4L.v4l2_ctrl_hevc_decode_params (C type), 779
V4L.v4l2_ctrl_hevc_pps (C type), 774
V4L.v4l2_ctrl_hevc_scaling_matrix (C type), 776
V4L.v4l2_ctrl_hevc_slice_params (C type), 775
V4L.v4l2_ctrl_hevc_sps (C type), 774
V4L.v4l2_ctrl_type (C type), 1217
V4L.v4l2_dbg_chip_info (C type), 1102
V4L.v4l2_dbg_match (C type), 1104
V4L.v4l2_dbg_register (C type), 1104
V4L.v4l2_decoder_cmd (C type), 1106
V4L.v4l2_dup (C function), 1065
V4L.v4l2_dv_timings (C type), 1154
V4L.v4l2_dv_timings_cap (C type), 1114
V4L.v4l2_edid (C type), 1158
V4L.v4l2_enc_idx (C type), 1159
V4L.v4l2_enc_idx_entry (C type), 1160
V4L.v4l2_encoder_cmd (C type), 1116
V4L.v4l2_enum_dv_timings (C type), 1121
V4L.v4l2_event (C type), 1109
V4L.v4l2_event_ctrl (C type), 1111
V4L.v4l2_event_frame_sync (C type), 1112
V4L.v4l2_event_motion_det (C type), 1112
V4L.v4l2_event_src_change (C type), 1112
V4L.v4l2_event_subscription (C type), 1244
V4L.v4l2_event_vsync (C type), 1111
V4L.v4l2_exportbuffer (C type), 1145
V4L.v4l2_ext_control (C type), 1162
V4L.v4l2_ext_controls (C type), 1164
V4L.v4l2_fd_open (C function), 1065

- V4L.v4l2_fmtdesc (C type), 1122
- V4L.v4l2_format (C type), 1174
- V4L.v4l2_fract (C type), 1140
- V4L.v4l2_framebuffer (C type), 1168
- V4L.v4l2_frequency (C type), 1176
- V4L.v4l2_frequency_band (C type), 1132
- V4L.v4l2_frmival_stepwise (C type), 1130
- V4L.v4l2_frmivalenum (C type), 1130
- V4L.v4l2_frmivaltypes (C type), 1131
- V4L.v4l2_frmsize_discrete (C type), 1128
- V4L.v4l2_frmsize_stepwise (C type), 1128
- V4L.v4l2_frmsizeenum (C type), 1128
- V4L.v4l2_frmsizetypes (C type), 1128
- V4L.v4l2_get_control (C function), 1065
- V4L.v4l2_hevc_dpb_entry (C type), 777
- V4L.v4l2_hevc_pred_weight_table (C type), 777
- V4L.v4l2_hw_freq_seek (C type), 1227
- V4L.v4l2_input (C type), 1134
- V4L.v4l2_ioctl (C function), 1065
- V4L.v4l2_jpegcompression (C type), 1179
- V4L.v4l2_mbus_framefmt (C type), 1052
- V4L.v4l2_meta_format (C type), 1063
- V4L.v4l2_modulator (C type), 1181
- V4L.v4l2_mpeg_vbi_fmt_ivtv (C type), 1037
- V4L.v4l2_mpeg_vbi_ITV0 (C type), 1037
- V4L.v4l2_mpeg_vbi_itv0 (C type), 1037
- V4L.v4l2_mpeg_vbi_itv0_line (C type), 1038
- V4L.v4l2_mpeg_video_hevc_decode_mode (C type), 778
- V4L.v4l2_mpeg_video_hevc_start_code (C type), 778
- V4L.v4l2_open (C function), 1065
- V4L.v4l2_output (C type), 1137
- V4L.v4l2_outputparm (C type), 1185
- V4L.v4l2_plane (C type), 975
- V4L.v4l2_priority (C type), 1188
- V4L.v4l2_rds_data (C type), 1041
- V4L.v4l2_read (C function), 1065
- V4L.v4l2_rect (C type), 990
- V4L.v4l2_requestbuffers (C type), 1224
- V4L.v4l2_sdr_format (C type), 1042
- V4L.v4l2_selection (C type), 1191
- V4L.v4l2_set_control (C function), 1065
- V4L.v4l2_sliced_vbi_cap (C type), 1193
- V4L.v4l2_sliced_vbi_data (C type), 1035
- V4L.v4l2_sliced_vbi_format (C type), 1033
- V4L.v4l2_standard (C type), 1140
- V4L.v4l2_streamparm (C type), 1184
- V4L.v4l2_subdev_capability (C type), 1242
- V4L.v4l2_subdev_crop (C type), 1236
- V4L.v4l2_subdev_format (C type), 1237
- V4L.v4l2_subdev_frame_interval (C type), 1239
- V4L.v4l2_subdev_frame_interval_enum (C type), 1230
- V4L.v4l2_subdev_frame_size_enum (C type), 1232
- V4L.v4l2_subdev_mbus_code_enum (C type), 1233
- V4L.v4l2_subdev_selection (C type), 1241
- V4L.v4l2_timecode (C type), 980
- V4L.v4l2_tuner (C type), 1196
- V4L.v4l2_tuner_type (C type), 1198
- V4L.v4l2_vbi_format (C type), 1029
- V4L.v4l2_window (C type), 988
- V4L.VIDIOC_CREATE_BUFS (C macro), 1096
- V4L.VIDIOC_CROPCAP (C macro), 1099
- V4L.VIDIOC_DBG_G_CHIP_INFO (C macro), 1101
- V4L.VIDIOC_DBG_G_REGISTER (C macro), 1103
- V4L.VIDIOC_DBG_S_REGISTER (C macro), 1103
- V4L.VIDIOC_DECODER_CMD (C macro), 1105
- V4L.VIDIOC_DQBUF (C macro), 1205
- V4L.VIDIOC_DQEVENT (C macro), 1108
- V4L.VIDIOC_DV_TIMINGS_CAP (C macro), 1113
- V4L.VIDIOC_ENCODER_CMD (C macro), 1116
- V4L.VIDIOC_ENUM_DV_TIMINGS (C macro), 1120
- V4L.VIDIOC_ENUM_FMT (C macro), 1121
- V4L.VIDIOC_ENUM_FRAMEINTERVALS (C macro), 1129
- V4L.VIDIOC_ENUM_FRAMESIZES (C macro), 1127
- V4L.VIDIOC_ENUM_FREQ_BANDS (C macro), 1132
- V4L.VIDIOC_ENUMAUDIO (C macro), 1118
- V4L.VIDIOC_ENUMAUDOUT (C macro), 1119
- V4L.VIDIOC_ENUMINPUT (C macro), 1134
- V4L.VIDIOC_ENUMOUTPUT (C macro), 1137
- V4L.VIDIOC_ENUMSTD (C macro), 1139
- V4L.VIDIOC_EXPBUF (C macro), 1144
- V4L.VIDIOC_G_AUDIO (C macro), 1146
- V4L.VIDIOC_G_AUDOUT (C macro), 1148
- V4L.VIDIOC_G_CROP (C macro), 1149
- V4L.VIDIOC_G_CTRL (C macro), 1151
- V4L.VIDIOC_G_DV_TIMINGS (C macro), 1152
- V4L.VIDIOC_G_EDID (C macro), 1157
- V4L.VIDIOC_G_ENC_INDEX (C macro), 1159
- V4L.VIDIOC_G_EXT_CTRL (C macro), 1161
- V4L.VIDIOC_G_FBUF (C macro), 1167
- V4L.VIDIOC_G_FMT (C macro), 1173

- V4L.VIDIOC_G_FREQUENCY (C macro), 1176
- V4L.VIDIOC_G_INPUT (C macro), 1178
- V4L.VIDIOC_G_JPEGCOMP (C macro), 1179
- V4L.VIDIOC_G_MODULATOR (C macro), 1180
- V4L.VIDIOC_G_OUTPUT (C macro), 1183
- V4L.VIDIOC_G_PARM (C macro), 1184
- V4L.VIDIOC_G_PRIORITY (C macro), 1188
- V4L.VIDIOC_G_SELECTION (C macro), 1189
- V4L.VIDIOC_G_SLICED_VBI_CAP (C macro), 1192
- V4L.VIDIOC_G_STD (C macro), 1194
- V4L.VIDIOC_G_TUNER (C macro), 1196
- V4L.VIDIOC_LOG_STATUS (C macro), 1202
- V4L.VIDIOC_OVERLAY (C macro), 1203
- V4L.VIDIOC_PREPARE_BUF (C macro), 1204
- V4L.VIDIOC_QBUF (C macro), 1205
- V4L.VIDIOC_QUERY_DV_TIMINGS (C macro), 1221
- V4L.VIDIOC_QUERY_EXT_CTRL (C macro), 1213
- V4L.VIDIOC_QUERYBUF (C macro), 1207
- V4L.VIDIOC_QUERYCAP (C macro), 1209
- V4L.VIDIOC_QUERYMENU (C macro), 1213
- V4L.VIDIOC_QUERYSTD (C macro), 1223
- V4L.VIDIOC_REQBUFS (C macro), 1224
- V4L.VIDIOC_S_AUDIO (C macro), 1146
- V4L.VIDIOC_S_AUDOUT (C macro), 1148
- V4L.VIDIOC_S_CROP (C macro), 1149
- V4L.VIDIOC_S_CTRL (C macro), 1151
- V4L.VIDIOC_S_DV_TIMINGS (C macro), 1152
- V4L.VIDIOC_S_EDID (C macro), 1157
- V4L.VIDIOC_S_EXT_CTRL (C macro), 1161
- V4L.VIDIOC_S_FBUF (C macro), 1167
- V4L.VIDIOC_S_FMT (C macro), 1173
- V4L.VIDIOC_S_FREQUENCY (C macro), 1176
- V4L.VIDIOC_S_HW_FREQ_SEEK (C macro), 1226
- V4L.VIDIOC_S_INPUT (C macro), 1178
- V4L.VIDIOC_S_JPEGCOMP (C macro), 1179
- V4L.VIDIOC_S_MODULATOR (C macro), 1180
- V4L.VIDIOC_S_OUTPUT (C macro), 1183
- V4L.VIDIOC_S_PARM (C macro), 1184
- V4L.VIDIOC_S_PRIORITY (C macro), 1188
- V4L.VIDIOC_S_SELECTION (C macro), 1189
- V4L.VIDIOC_S_STD (C macro), 1194
- V4L.VIDIOC_S_TUNER (C macro), 1196
- V4L.VIDIOC_STREAMOFF (C macro), 1228
- V4L.VIDIOC_STREAMON (C macro), 1228
- V4L.VIDIOC_SUBDEV_DV_TIMINGS_CAP (C macro), 1113
- V4L.VIDIOC_SUBDEV_ENUM_DV_TIMINGS (C macro), 1120
- V4L.VIDIOC_SUBDEV_ENUM_FRAME_INTERVAL (C macro), 1230
- V4L.VIDIOC_SUBDEV_ENUM_FRAME_SIZE (C macro), 1231
- V4L.VIDIOC_SUBDEV_ENUM_MBUS_CODE (C macro), 1233
- V4L.VIDIOC_SUBDEV_ENUMSTD (C macro), 1139
- V4L.VIDIOC_SUBDEV_G_CROP (C macro), 1235
- V4L.VIDIOC_SUBDEV_G_DV_TIMINGS (C macro), 1152
- V4L.VIDIOC_SUBDEV_G_EDID (C macro), 1157
- V4L.VIDIOC_SUBDEV_G_FMT (C macro), 1237
- V4L.VIDIOC_SUBDEV_G_FRAME_INTERVAL (C macro), 1239
- V4L.VIDIOC_SUBDEV_G_SELECTION (C macro), 1240
- V4L.VIDIOC_SUBDEV_G_STD (C macro), 1194
- V4L.VIDIOC_SUBDEV_QUERY_DV_TIMINGS (C macro), 1221
- V4L.VIDIOC_SUBDEV_QUERYCAP (C macro), 1242
- V4L.VIDIOC_SUBDEV_QUERYSTD (C macro), 1223
- V4L.VIDIOC_SUBDEV_S_CROP (C macro), 1235
- V4L.VIDIOC_SUBDEV_S_DV_TIMINGS (C macro), 1152
- V4L.VIDIOC_SUBDEV_S_EDID (C macro), 1157
- V4L.VIDIOC_SUBDEV_S_FMT (C macro), 1237
- V4L.VIDIOC_SUBDEV_S_FRAME_INTERVAL (C macro), 1239
- V4L.VIDIOC_SUBDEV_S_SELECTION (C macro), 1240
- V4L.VIDIOC_SUBDEV_S_STD (C macro), 1194
- V4L.VIDIOC_SUBSCRIBE_EVENT (C macro), 1244
- V4L.VIDIOC_TRY_DECODER_CMD (C macro), 1105
- V4L.VIDIOC_TRY_ENCODER_CMD (C macro), 1116
- V4L.VIDIOC_TRY_EXT_CTRL (C macro), 1161
- V4L.VIDIOC_TRY_FMT (C macro), 1173
- V4L.VIDIOC_UNSUBSCRIBE_EVENT (C macro), 1244
- V4L.write (C function), 1254
- V4L2 API, **1610**
- V4L2 Device Node, **1610**
- V4L2 Hardware, **1610**
- V4L2 Sub-device, **1610**
- V4L2 Sub-device API, **1610**
- v4l2_area (C type), 747
- v4l2_async_debug_init (C function), 394
- v4l2_async_match_type (C type), 392

- `v4l2_async_nf_add_fwnode` (C function), 394
`v4l2_async_nf_add_fwnode_remote` (C function), 395
`v4l2_async_nf_add_i2c` (C function), 395
`v4l2_async_nf_cleanup` (C function), 396
`v4l2_async_nf_init` (C function), 394
`v4l2_async_nf_parse_fwnode_endpoints` (C function), 405
`v4l2_async_nf_register` (C function), 395
`v4l2_async_nf_unregister` (C function), 396
`v4l2_async_notifier` (C type), 393
`v4l2_async_notifier_operations` (C type), 393
`v4l2_async_register_subdev` (C function), 396
`v4l2_async_register_subdev_sensor` (C function), 396
`v4l2_async_subdev` (C type), 392
`v4l2_async_subdev_nf_register` (C function), 395
`v4l2_async_unregister_subdev` (C function), 396
`v4l2_calc_aspect_ratio` (C function), 371
`v4l2_calc_timeperframe` (C function), 368
`v4l2_check_dv_timings_fnc` (C type), 368
`v4l2_colorspace` (C type), 949
`v4l2_compat_ioctl32` (C function), 427
`v4l2_connector_link` (C type), 399
`v4l2_connector_type` (C type), 399
`v4l2_create_fwnode_links` (C function), 376
`v4l2_create_fwnode_links_to_pad` (C function), 376
`v4l2_ctrl` (C type), 315
`v4l2_ctrl_activate` (C function), 327
`v4l2_ctrl_add_handler` (C function), 325
`v4l2_ctrl_auto_cluster` (C function), 326
`v4l2_ctrl_cluster` (C function), 326
`v4l2_ctrl_config` (C type), 319
`v4l2_ctrl_fill` (C function), 320
`v4l2_ctrl_filter` (C type), 325
`v4l2_ctrl_find` (C function), 326
`v4l2_ctrl_fwht_params` (C type), 785
`v4l2_ctrl_g_ctrl` (C function), 329
`v4l2_ctrl_g_ctrl_int64` (C function), 330
`v4l2_ctrl_get_int_menu` (C function), 329
`v4l2_ctrl_get_menu` (C function), 329
`v4l2_ctrl_get_name` (C function), 329
`v4l2_ctrl_grab` (C function), 327
`v4l2_ctrl_h264_decode_params` (C type), 783
`v4l2_ctrl_h264_pps` (C type), 781
`v4l2_ctrl_h264_pred_weights` (C type), 782
`v4l2_ctrl_h264_scaling_matrix` (C type), 781
`v4l2_ctrl_h264_slice_params` (C type), 782
`v4l2_ctrl_h264_sps` (C type), 780
`v4l2_ctrl_handler` (C type), 318
`v4l2_ctrl_handler_free` (C function), 322
`v4l2_ctrl_handler_init` (C function), 321
`v4l2_ctrl_handler_init_class` (C function), 321
`v4l2_ctrl_handler_log_status` (C function), 322
`v4l2_ctrl_handler_setup` (C function), 322
`v4l2_ctrl_lock` (C function), 322
`v4l2_ctrl_log_status` (C function), 332
`v4l2_ctrl_merge` (C function), 332
`v4l2_ctrl_modify_range` (C function), 328
`v4l2_ctrl_mpeg2_picture` (C type), 790
`v4l2_ctrl_mpeg2_quantisation` (C type), 791
`v4l2_ctrl_mpeg2_sequence` (C type), 790
`v4l2_ctrl_new_custom` (C function), 323
`v4l2_ctrl_new_fwnode_properties` (C function), 336
`v4l2_ctrl_new_int_menu` (C function), 325
`v4l2_ctrl_new_std` (C function), 323
`v4l2_ctrl_new_std_compound` (C function), 324
`v4l2_ctrl_new_std_menu` (C function), 323
`v4l2_ctrl_new_std_menu_items` (C function), 324
`v4l2_ctrl_notify` (C function), 328
`v4l2_ctrl_notify_fnc` (C type), 314
`v4l2_ctrl_ops` (C type), 314
`v4l2_ctrl_poll` (C function), 332
`v4l2_ctrl_ptr` (C type), 312
`v4l2_ctrl_ptr_create` (C function), 313
`v4l2_ctrl_query_fill` (C function), 412
`v4l2_ctrl_radio_filter` (C function), 326
`v4l2_ctrl_ref` (C type), 317
`v4l2_ctrl_replace` (C function), 332
`v4l2_ctrl_request_complete` (C function), 333
`v4l2_ctrl_request_hdl_ctrl_find` (C function), 334
`v4l2_ctrl_request_hdl_find` (C function), 333
`v4l2_ctrl_request_hdl_put` (C function), 334
`v4l2_ctrl_request_setup` (C function), 333
`v4l2_ctrl_s_ctrl` (C function), 330
`v4l2_ctrl_s_ctrl_compound` (C function), 331

- v4l2_ctrl_s_ctrl_int64 (C function), 330
 v4l2_ctrl_s_ctrl_string (C function), 331
 v4l2_ctrl_subdev_log_status (C function), 336
 v4l2_ctrl_subdev_subscribe_event (C function), 336
 v4l2_ctrl_subscribe_event (C function), 332
 v4l2_ctrl_to_v4l2_flash (C function), 373
 v4l2_ctrl_type_ops (C type), 314
 v4l2_ctrl_unlock (C function), 322
 v4l2_ctrl_vp8_frame (C type), 786
 v4l2_ctrl_vp9_compressed_hdr (C type), 791
 v4l2_ctrl_vp9_frame (C type), 792
 v4l2_decode_vbi_line (C type), 276
 v4l2_detect_cvt (C function), 370
 v4l2_detect_gtf (C function), 370
 v4l2_device (C type), 256
 v4l2_device_call_all (C function), 261
 v4l2_device_call_until_err (C function), 262
 v4l2_device_disconnect (C function), 258
 v4l2_device_for_each_subdev (C function), 259
 v4l2_device_get (C function), 256
 v4l2_device_has_op (C function), 263
 v4l2_device_mask_call_all (C function), 262
 v4l2_device_mask_call_until_err (C function), 262
 v4l2_device_mask_has_op (C function), 263
 v4l2_device_put (C function), 257
 v4l2_device_register (C function), 257
 v4l2_device_register_ro_subdev_nodes (C function), 259
 v4l2_device_register_subdev (C function), 258
 v4l2_device_register_subdev_nodes (C function), 259
 v4l2_device_set_name (C function), 257
 v4l2_device_supports_requests (C function), 259
 v4l2_device_unregister (C function), 258
 v4l2_device_unregister_subdev (C function), 258
 v4l2_disable_ioctl (C function), 252
 v4l2_dv_timings_aspect_ratio (C function), 371
 v4l2_enum_dv_timings_cap (C function), 368
 v4l2_event_dequeue (C function), 298
 v4l2_event_pending (C function), 299
 v4l2_event_queue (C function), 298
 v4l2_event_queue_fh (C function), 298
 v4l2_event_subdev_unsubscribe (C function), 299
 v4l2_event_subscribe (C function), 299
 v4l2_event_unsubscribe (C function), 299
 v4l2_event_unsubscribe_all (C function), 299
 v4l2_event_wake_all (C function), 298
 v4l2_fh (C type), 266
 v4l2_fh_add (C function), 266
 v4l2_fh_del (C function), 267
 v4l2_fh_exit (C function), 267
 v4l2_fh_init (C function), 266
 v4l2_fh_is_singular (C function), 268
 v4l2_fh_is_singular_file (C function), 268
 v4l2_fh_open (C function), 267
 v4l2_fh_release (C function), 267
 v4l2_field (C type), 981
 v4l2_file_operations (C type), 247
 v4l2_fill_mbus_format (C function), 378
 v4l2_fill_mbus_format_mplane (C function), 379
 v4l2_fill_pix_format (C function), 378
 v4l2_fill_pix_format_mplane (C function), 379
 v4l2_find_dv_timings_cap (C function), 369
 v4l2_find_dv_timings_cea861_vic (C function), 369
 v4l2_find_nearest_size (C function), 416
 v4l2_flash (C type), 373
 v4l2_flash_config (C type), 373
 v4l2_flash_ctrl_data (C type), 372
 v4l2_flash_indicator_init (C function), 374
 v4l2_flash_init (C function), 374
 v4l2_flash_ops (C type), 372
 v4l2_flash_release (C function), 374
 v4l2_format_info (C type), 417
 v4l2_fwnode_bus_mipi_csi1 (C type), 397
 v4l2_fwnode_bus_mipi_csi2 (C type), 397
 v4l2_fwnode_bus_parallel (C type), 397
 v4l2_fwnode_bus_type (C type), 400
 v4l2_fwnode_connector (C type), 400
 v4l2_fwnode_connector_add_link (C function), 404
 v4l2_fwnode_connector_analog (C type), 400
 v4l2_fwnode_connector_free (C function), 403
 v4l2_fwnode_connector_parse (C function), 403

v4l2_fwnode_device_parse (C function), 404
v4l2_fwnode_device_properties (C type), 399
v4l2_fwnode_endpoint (C type), 398
v4l2_fwnode_endpoint_alloc_parse (C function), 402
v4l2_fwnode_endpoint_free (C function), 401
v4l2_fwnode_endpoint_parse (C function), 401
v4l2_fwnode_link (C type), 399
v4l2_fwnode_orientation (C type), 398
v4l2_fwnode_parse_link (C function), 402
V4L2_FWNODE_PROPERTY_UNSET (C function), 398
v4l2_fwnode_put_link (C function), 403
v4l2_g_ctrl (C function), 335
v4l2_g_ext_ctrls (C function), 335
v4l2_g_parm_cap (C function), 416
v4l2_get_link_freq (C function), 418
v4l2_get_subdev_hostdata (C function), 292
v4l2_get_subdevdata (C function), 291
v4l2_h264_dpb_entry (C type), 784
v4l2_h264_reference (C type), 783
v4l2_h264_weight_factors (C type), 782
v4l2_hdmi_colorimetry (C type), 372
v4l2_hsv_encoding (C type), 951
v4l2_i2c_new_subdev (C function), 413
v4l2_i2c_new_subdev_board (C function), 413
v4l2_i2c_subdev_addr (C function), 414
v4l2_i2c_subdev_init (C function), 414
v4l2_i2c_subdev_set_name (C function), 414
v4l2_i2c_subdev_unregister (C function), 414
v4l2_i2c_tuner_addrs (C function), 414
v4l2_i2c_tuner_type (C type), 413
v4l2_ioctl_ops (C type), 418
v4l2_kevent (C type), 296
v4l2_kioctl (C type), 427
v4l2_m2m_buf_copy_metadata (C function), 391
v4l2_m2m_buf_done_and_job_finish (C function), 382
v4l2_m2m_buf_queue (C function), 388
v4l2_m2m_buf_remove (C function), 390
v4l2_m2m_buf_remove_by_buf (C function), 391
v4l2_m2m_buffer (C type), 381
v4l2_m2m_clear_state (C function), 383
v4l2_m2m_create_bufs (C function), 385
v4l2_m2m_ctx (C type), 380
v4l2_m2m_ctx_init (C function), 388
v4l2_m2m_ctx_release (C function), 388
v4l2_m2m_decoder_cmd (C function), 387
v4l2_m2m_dqbuf (C function), 385
v4l2_m2m_dst_buf_is_last (C function), 383
v4l2_m2m_dst_buf_remove (C function), 391
v4l2_m2m_dst_buf_remove_by_buf (C function), 391
v4l2_m2m_encoder_cmd (C function), 386
v4l2_m2m_expbuf (C function), 385
v4l2_m2m_for_each_dst_buf (C function), 390
v4l2_m2m_for_each_dst_buf_safe (C function), 390
v4l2_m2m_for_each_src_buf (C function), 390
v4l2_m2m_for_each_src_buf_safe (C function), 390
v4l2_m2m_get_curr_priv (C function), 381
v4l2_m2m_get_dst_vq (C function), 390
v4l2_m2m_get_src_vq (C function), 390
v4l2_m2m_get_vq (C function), 381
v4l2_m2m_has_stopped (C function), 383
v4l2_m2m_init (C function), 387
v4l2_m2m_is_last_draining_src_buf (C function), 383
v4l2_m2m_job_finish (C function), 382
v4l2_m2m_last_buf (C function), 389
v4l2_m2m_last_buffer_done (C function), 384
v4l2_m2m_last_dst_buf (C function), 389
v4l2_m2m_last_src_buf (C function), 389
v4l2_m2m_mark_stopped (C function), 383
v4l2_m2m_mmap (C function), 387
v4l2_m2m_next_buf (C function), 389
v4l2_m2m_next_dst_buf (C function), 389
v4l2_m2m_next_src_buf (C function), 389
v4l2_m2m_num_dst_bufs_ready (C function), 389
v4l2_m2m_num_src_bufs_ready (C function), 389
v4l2_m2m_ops (C type), 379
v4l2_m2m_poll (C function), 387
v4l2_m2m_prepare_buf (C function), 385
v4l2_m2m_qbuf (C function), 385
v4l2_m2m_querybuf (C function), 384
v4l2_m2m_queue_ctx (C type), 380
v4l2_m2m_release (C function), 388
v4l2_m2m_reqbufs (C function), 384
v4l2_m2m_resume (C function), 384

- [v4l2_m2m_src_buf_remove \(C function\)](#), 390
[v4l2_m2m_src_buf_remove_by_buf \(C function\)](#), 391
[v4l2_m2m_streamoff \(C function\)](#), 386
[v4l2_m2m_streamon \(C function\)](#), 386
[v4l2_m2m_suspend \(C function\)](#), 384
[v4l2_m2m_try_schedule \(C function\)](#), 381
[v4l2_m2m_update_start_streaming_state \(C function\)](#), 386
[v4l2_m2m_update_stop_streaming_state \(C function\)](#), 386
[v4l2_match_dv_timings \(C function\)](#), 369
[v4l2_mbus_config \(C type\)](#), 378
[v4l2_mbus_frame_desc \(C type\)](#), 280
[v4l2_mbus_frame_desc_entry \(C type\)](#), 280
[v4l2_mbus_frame_desc_flags \(C type\)](#), 280
[v4l2_mbus_type \(C type\)](#), 378
[v4l2_mc_create_media_graph \(C function\)](#), 375
[v4l2_norm_to_name \(C function\)](#), 426
[v4l2_pipeline_link_notify \(C function\)](#), 377
[v4l2_pipeline_pm_get \(C function\)](#), 376
[v4l2_pipeline_pm_put \(C function\)](#), 377
[v4l2_pix_format \(C type\)](#), 820
[v4l2_pix_format_mplane \(C type\)](#), 825
[v4l2_pixel_encoding \(C type\)](#), 417
[v4l2_plane_pix_format \(C type\)](#), 825
[v4l2_print_dv_timings \(C function\)](#), 370
[v4l2_prio_change \(C function\)](#), 246
[v4l2_prio_check \(C function\)](#), 247
[v4l2_prio_close \(C function\)](#), 247
[v4l2_prio_init \(C function\)](#), 246
[v4l2_prio_max \(C function\)](#), 247
[v4l2_prio_open \(C function\)](#), 246
[v4l2_prio_state \(C type\)](#), 246
[v4l2_quantization \(C type\)](#), 951
[v4l2_query_ext_ctrl \(C function\)](#), 334
[v4l2_queryctrl \(C function\)](#), 334
[v4l2_querymenu \(C function\)](#), 334
[v4l2_rect_enclosed \(C function\)](#), 408
[v4l2_rect_equal \(C function\)](#), 407
[v4l2_rect_intersect \(C function\)](#), 407
[v4l2_rect_map_inside \(C function\)](#), 406
[v4l2_rect_overlap \(C function\)](#), 407
[v4l2_rect_same_position \(C function\)](#), 407
[v4l2_rect_same_size \(C function\)](#), 406
[v4l2_rect_scale \(C function\)](#), 407
[v4l2_rect_set_max_size \(C function\)](#), 406
[v4l2_rect_set_min_size \(C function\)](#), 406
[v4l2_rect_set_size_to \(C function\)](#), 406
[v4l2_s_ctrl \(C function\)](#), 335
[v4l2_s_ext_ctrls \(C function\)](#), 336
[v4l2_s_parm_cap \(C function\)](#), 417
[v4l2_set_subdev_hostdata \(C function\)](#), 291
[v4l2_set_subdevdata \(C function\)](#), 291
[v4l2_spi_new_subdev \(C function\)](#), 415
[v4l2_spi_subdev_init \(C function\)](#), 415
[v4l2_spi_subdev_unregister \(C function\)](#), 415
[v4l2_src_change_event_subdev_subscribe \(C function\)](#), 300
[v4l2_src_change_event_subscribe \(C function\)](#), 300
[v4l2_stateless_h264_decode_mode \(C type\)](#), 785
[v4l2_stateless_h264_start_code \(C type\)](#), 785
[v4l2_subdev \(C type\)](#), 288
[v4l2_subdev_alloc_state \(C function\)](#), 293
[v4l2_subdev_audio_ops \(C type\)](#), 279
[v4l2_subdev_call \(C function\)](#), 293
[v4l2_subdev_core_ops \(C type\)](#), 277
[v4l2_subdev_fh \(C type\)](#), 290
[v4l2_subdev_free_state \(C function\)](#), 293
[v4l2_subdev_get_fwnode_pad_1_to_1 \(C function\)](#), 292
[v4l2_subdev_get_try_compose \(C function\)](#), 291
[v4l2_subdev_get_try_crop \(C function\)](#), 291
[v4l2_subdev_get_try_format \(C function\)](#), 291
[v4l2_subdev_has_op \(C function\)](#), 293
[v4l2_subdev_init \(C function\)](#), 293
[v4l2_subdev_internal_ops \(C type\)](#), 287
[v4l2_subdev_io_pin_bits \(C type\)](#), 276
[v4l2_subdev_io_pin_config \(C type\)](#), 276
[v4l2_subdev_ir_mode \(C type\)](#), 283
[v4l2_subdev_ir_ops \(C type\)](#), 284
[v4l2_subdev_ir_parameters \(C type\)](#), 283
[v4l2_subdev_link_validate \(C function\)](#), 292
[v4l2_subdev_link_validate_default \(C function\)](#), 292
[v4l2_subdev_notify \(C function\)](#), 259
[v4l2_subdev_notify_event \(C function\)](#), 293
[v4l2_subdev_ops \(C type\)](#), 287
[v4l2_subdev_pad_config \(C type\)](#), 285
[v4l2_subdev_pad_ops \(C type\)](#), 285
[v4l2_subdev_platform_data \(C type\)](#), 288
[v4l2_subdev_pre_streamon_flags \(C type\)](#), 280

- v4l2_subdev_sensor_ops (C type), 283
- v4l2_subdev_state (C type), 285
- v4l2_subdev_to_v4l2_flash (C function), 373
- v4l2_subdev_tuner_ops (C type), 278
- v4l2_subdev_vbi_ops (C type), 282
- v4l2_subdev_video_ops (C type), 280
- v4l2_subscribed_event (C type), 297
- v4l2_subscribed_event_ops (C type), 297
- v4l2_try_ext_ctrls (C function), 335
- v4l2_valid_dv_timings (C function), 368
- v4l2_video_device_flags (C type), 245
- v4l2_video_std_construct (C function), 426
- v4l2_video_std_frame_period (C function), 426
- v4l2_vp8_entropy (C type), 789
- v4l2_vp8_entropy_coder_state (C type), 788
- v4l2_vp8_loop_filter (C type), 789
- v4l2_vp8_quantization (C type), 789
- v4l2_vp8_segment (C type), 788
- v4l2_vp9_loop_filter (C type), 796
- v4l2_vp9_quantization (C type), 796
- v4l2_vp9_segmentation (C type), 795
- v4l2_xfer_func (C type), 950
- v4l2_ycbcr_encoding (C type), 950
- v4l_bound_align_image (C function), 415
- v4l_disable_media_source (C function), 375
- v4l_enable_media_source (C function), 375
- v4l_printk_ioctl (C function), 427
- v4l_video_std_enumstd (C function), 427
- vb2_buf_ops (C type), 348
- vb2_buffer (C type), 346
- vb2_buffer_done (C function), 351
- vb2_buffer_in_use (C function), 361
- vb2_buffer_state (C type), 346
- vb2_clear_last_buffer_dequeued (C function), 361
- vb2_core_create_bufs (C function), 353
- vb2_core_dqbuf (C function), 355
- vb2_core_expbuf (C function), 356
- vb2_core_poll (C function), 358
- vb2_core_prepare_buf (C function), 354
- vb2_core_qbuf (C function), 354
- vb2_core_querybuf (C function), 352
- vb2_core_queue_init (C function), 356
- vb2_core_queue_release (C function), 357
- vb2_core_reqbufs (C function), 352
- vb2_core_streamoff (C function), 356
- vb2_core_streamon (C function), 355
- vb2_create_bufs (C function), 363
- vb2_discard_done (C function), 352
- vb2_dqbuf (C function), 364
- vb2_expbuf (C function), 364
- vb2_fileio_is_active (C function), 360
- vb2_find_timestamp (C function), 362
- vb2_get_buffer (C function), 361
- vb2_get_drv_priv (C function), 360
- vb2_get_plane_payload (C function), 360
- vb2_get_unmapped_area (C function), 357
- vb2_io_modes (C type), 345
- vb2_is_busy (C function), 360
- vb2_is_streaming (C function), 359
- vb2_mem_ops (C type), 343
- vb2_memory (C type), 343
- vb2_mmap (C function), 357
- vb2_ops (C type), 347
- vb2_ops_wait_finish (C function), 367
- vb2_ops_wait_prepare (C function), 367
- vb2_plane (C type), 345
- vb2_plane_cookie (C function), 351
- vb2_plane_size (C function), 360
- vb2_plane_vaddr (C function), 351
- vb2_poll (C function), 366
- vb2_prepare_buf (C function), 363
- vb2_qbuf (C function), 363
- vb2_queue (C type), 349
- vb2_queue_allows_cache_hints (C function), 351
- vb2_queue_change_type (C function), 366
- vb2_queue_error (C function), 357
- vb2_queue_init (C function), 365
- vb2_queue_init_name (C function), 366
- vb2_queue_release (C function), 366
- vb2_read (C function), 358
- vb2_reqbufs (C function), 363
- vb2_request_buffer_cnt (C function), 362
- vb2_request_object_is_buffer (C function), 361
- vb2_set_plane_payload (C function), 360
- vb2_start_streaming_called (C function), 361
- vb2_streamoff (C function), 365
- vb2_streamon (C function), 365
- vb2_thread_fnc (C type), 359
- vb2_thread_start (C function), 359
- vb2_thread_stop (C function), 359
- vb2_v4l2_buffer (C type), 362
- vb2_verify_memory_type (C function), 361
- vb2_video_unregister_device (C function), 367
- vb2_vmarea_handler (C type), 367
- vb2_wait_for_all_buffers (C function), 352

- vb2_write (C function), 358
 vdev_to_v4l2_subdev (C function), 290
 vfl_devnode_direction (C type), 245
 vfl_devnode_type (C type), 245
 video_devdata (C function), 253
 video_device (C type), 248
 video_device_alloc (C function), 252
 video_device_node_name (C function), 253
 video_device_release (C function), 252
 video_device_release_empty (C function), 252
 video_drvdata (C function), 253
 video_get_drvdata (C function), 253
 video_ioctl2 (C function), 428
 video_is_registered (C function), 253
 video_register_device (C function), 251
 video_register_device_no_warn (C function), 251
 video_set_drvdata (C function), 253
 video_unregister_device (C function), 251
 video_usercopy (C function), 427
 Video-node-centric, **1610**
 vidtv_channel (C type), 686
 vidtv_channel_si_init (C function), 687
 vidtv_channels_init (C function), 687
 vidtv_demod_cnr_to_qual_s (C type), 687
 vidtv_demod_config (C type), 688
 vidtv_demod_state (C type), 688
 vidtv_dvb (C type), 686
 vidtv_encoder (C type), 689
 vidtv_memcpy (C function), 708
 vidtv_memset (C function), 708
 vidtv_mux (C type), 691
 vidtv_mux_init_args (C type), 692
 vidtv_mux_pid_ctx (C type), 691
 vidtv_mux_si (C type), 690
 vidtv_mux_timing (C type), 690
 vidtv_pes_write_into (C function), 695
 vidtv_pmt_desc_assign (C function), 698
 vidtv_psi_desc_assign (C function), 698
 vidtv_psi_eit_event_assign (C function), 705
 vidtv_psi_eit_table_update_sec_len (C function), 705
 vidtv_psi_eit_write_args (C type), 704
 vidtv_psi_eit_write_into (C function), 704
 vidtv_psi_find_pmt_sec (C function), 702
 vidtv_psi_nit_write_args (C type), 703
 vidtv_psi_nit_write_into (C function), 704
 vidtv_psi_pat_program_assign (C function), 698
 vidtv_psi_pat_table_update_sec_len (C function), 699
 vidtv_psi_pat_write_args (C type), 700
 vidtv_psi_pat_write_into (C function), 700
 vidtv_psi_pmt_create_sec_for_each_pat_entry (C function), 699
 vidtv_psi_pmt_get_pid (C function), 699
 vidtv_psi_pmt_stream_assign (C function), 699
 vidtv_psi_pmt_table_update_sec_len (C function), 700
 vidtv_psi_pmt_write_args (C type), 701
 vidtv_psi_pmt_write_into (C function), 702
 vidtv_psi_sdt_service_assign (C function), 697
 vidtv_psi_sdt_table_update_sec_len (C function), 700
 vidtv_psi_sdt_write_args (C type), 701
 vidtv_psi_sdt_write_into (C function), 701
 vidtv_psi_table_nit (C type), 703
 vidtv_psi_table_transport (C type), 702
 vidtv_s302m_ctx (C type), 705
 vidtv_s302m_encoder_init_args (C type), 706
 vidtv_sdt_desc_assign (C function), 698
 vidtv_ts_null_write_into (C function), 707
 vidtv_ts_pcr_write_into (C function), 707
 vidtv_tuner_config (C type), 707
 vidtv_tuner_dev (C type), 709
 vidtv_tuner_hardware_state (C type), 709
 vimc_get_source_entity (C function), 641
 vimc_stream (C type), 640
 vimc_streamer_pipeline_init (C function), 641
 vimc_streamer_pipeline_terminate (C function), 641
 vimc_streamer_s_stream (C function), 642
 vimc_streamer_thread (C function), 641
- ## Z
- zd1301_demod_get_dvb_frontend (C function), 677
 zd1301_demod_get_i2c_adapter (C function), 677
 zd1301_demod_platform_data (C type), 677
 zl10036_attach (C function), 678