

ckb-daemon

Generated by Doxygen 1.8.14

Contents

1	Todo List	1
2	Data Structure Index	3
2.1	Data Structures	3
3	File Index	5
3.1	File List	5
4	Data Structure Documentation	7
4.1	devcmd.__unnamed__ Struct Reference	7
4.1.1	Detailed Description	8
4.1.2	Field Documentation	8
4.1.2.1	active	8
4.1.2.2	allocprofile	8
4.1.2.3	bind	8
4.1.2.4	dpi	9
4.1.2.5	dpisel	9
4.1.2.6	erase	9
4.1.2.7	eraseprofile	9
4.1.2.8	freeprofile	9
4.1.2.9	fwupdate	9
4.1.2.10	get	9
4.1.2.11	hwload	9
4.1.2.12	hwsave	9
4.1.2.13	iauto	9

4.1.2.14	id	10
4.1.2.15	idle	10
4.1.2.16	inotify	10
4.1.2.17	ioff	10
4.1.2.18	ion	10
4.1.2.19	lift	10
4.1.2.20	loadprofile	10
4.1.2.21	macro	10
4.1.2.22	name	10
4.1.2.23	notify	10
4.1.2.24	pollrate	11
4.1.2.25	profileid	11
4.1.2.26	profilename	11
4.1.2.27	rebind	11
4.1.2.28	restart	11
4.1.2.29	rgb	11
4.1.2.30	setmodeindex	11
4.1.2.31	snap	11
4.1.2.32	start	11
4.1.2.33	unbind	11
4.1.2.34	updatedpi	11
4.1.2.35	updateindicators	11
4.1.2.36	updatergb	11

5 File Documentation	13
5.1 command.c File Reference	13
5.1.1 Macro Definition Documentation	14
5.1.1.1 TRY_WITH_RESET	14
5.1.2 Function Documentation	14
5.1.2.1 readcmd()	14
5.1.3 Variable Documentation	18
5.1.3.1 cmd_strings	18
5.2 command.h File Reference	19
5.2.1 Data Structure Documentation	20
5.2.1.1 union devcmd	20
5.2.2 Macro Definition Documentation	21
5.2.2.1 CMD_COUNT	21
5.2.2.2 CMD_DEV_COUNT	21
5.2.3 Typedef Documentation	21
5.2.3.1 cmdhandler	21
5.2.3.2 cmdhandler_io	21
5.2.3.3 cmdhandler_mac	22
5.2.3.4 devcmd	22
5.2.4 Enumeration Type Documentation	22
5.2.4.1 cmd	22
5.2.5 Function Documentation	24
5.2.5.1 readcmd()	24
5.3 device.c File Reference	28
5.3.1 Function Documentation	29
5.3.1.1 _start_dev()	29
5.3.1.2 start_dev()	31
5.3.2 Variable Documentation	31
5.3.2.1 devlistmutex	31
5.3.2.2 devmutex	31

5.3.2.3	hwload_mode	32
5.3.2.4	inputmutex	32
5.3.2.5	keyboard	32
5.4	device.h File Reference	32
5.4.1	Macro Definition Documentation	33
5.4.1.1	ACT_LIGHT	34
5.4.1.2	ACT_LOCK	34
5.4.1.3	ACT_M1	34
5.4.1.4	ACT_M2	34
5.4.1.5	ACT_M3	34
5.4.1.6	ACT_MR_RING	35
5.4.1.7	ACT_NEXT	35
5.4.1.8	ACT_NEXT_NOWRAP	35
5.4.1.9	DEV_MAX	35
5.4.1.10	dmutex	35
5.4.1.11	imutex	36
5.4.1.12	IN_CORSAIR	36
5.4.1.13	IN_HID	36
5.4.1.14	IS_CONNECTED	36
5.4.1.15	setactive	36
5.4.2	Function Documentation	37
5.4.2.1	cmd_active_kb()	37
5.4.2.2	cmd_active_mouse()	37
5.4.2.3	cmd_idle_kb()	38
5.4.2.4	cmd_idle_mouse()	39
5.4.2.5	cmd_pollrate()	39
5.4.2.6	setactive_kb()	40
5.4.2.7	setactive_mouse()	42
5.4.2.8	setmodeindex_nrgb()	43
5.4.2.9	start_dev()	43

5.4.2.10	start_kb_nrgb()	44
5.4.3	Variable Documentation	44
5.4.3.1	devmutex	44
5.4.3.2	inputmutex	44
5.4.3.3	keyboard	45
5.5	device_keyboard.c File Reference	45
5.5.1	Function Documentation	45
5.5.1.1	cmd_active_kb()	46
5.5.1.2	cmd_idle_kb()	46
5.5.1.3	setactive_kb()	47
5.5.1.4	setmodeindex_nrgb()	49
5.5.1.5	start_kb_nrgb()	49
5.6	device_mouse.c File Reference	49
5.6.1	Function Documentation	50
5.6.1.1	cmd_active_mouse()	50
5.6.1.2	cmd_idle_mouse()	51
5.6.1.3	cmd_pollrate()	51
5.6.1.4	setactive_mouse()	52
5.7	device_vtable.c File Reference	53
5.7.1	Function Documentation	54
5.7.1.1	cmd_io_none()	54
5.7.1.2	cmd_macro_none()	54
5.7.1.3	cmd_none()	55
5.7.1.4	int1_int_none()	55
5.7.1.5	int1_void_none()	55
5.7.1.6	loadprofile_none()	55
5.7.2	Variable Documentation	56
5.7.2.1	vtable_keyboard	56
5.7.2.2	vtable_keyboard_nonrgb	56
5.7.2.3	vtable_mouse	56

5.8	devnode.c File Reference	56
5.8.1	Data Structure Documentation	58
5.8.1.1	struct_readlines_ctx	58
5.8.2	Macro Definition Documentation	58
5.8.2.1	MAX_BUFFER	58
5.8.2.2	S_GID_READ	59
5.8.3	Function Documentation	59
5.8.3.1	_mkdevpath()	59
5.8.3.2	_mknotifynode()	61
5.8.3.3	_rmnotifynode()	62
5.8.3.4	_updateconnected()	63
5.8.3.5	mkdevpath()	64
5.8.3.6	mkfwnode()	65
5.8.3.7	mknotifynode()	65
5.8.3.8	readlines()	66
5.8.3.9	readlines_ctx_free()	67
5.8.3.10	readlines_ctx_init()	68
5.8.3.11	rm_recursive()	68
5.8.3.12	rmdevpath()	69
5.8.3.13	rmnotifynode()	70
5.8.3.14	updateconnected()	70
5.8.4	Variable Documentation	71
5.8.4.1	devpath	71
5.8.4.2	gid	71
5.9	devnode.h File Reference	72
5.9.1	Macro Definition Documentation	73
5.9.1.1	S_CUSTOM	73
5.9.1.2	S_CUSTOM_R	73
5.9.1.3	S_READ	73
5.9.1.4	S_READDIR	74

5.9.1.5	S_READWRITE	74
5.9.2	Typedef Documentation	74
5.9.2.1	readlines_ctx	74
5.9.3	Function Documentation	74
5.9.3.1	mkdevpath()	74
5.9.3.2	mkfwnode()	75
5.9.3.3	mknotifynode()	76
5.9.3.4	readlines()	77
5.9.3.5	readlines_ctx_free()	78
5.9.3.6	readlines_ctx_init()	78
5.9.3.7	rmdevpath()	79
5.9.3.8	rmnotifynode()	80
5.9.3.9	updateconnected()	80
5.9.4	Variable Documentation	81
5.9.4.1	devpath	81
5.9.4.2	gid	81
5.10	dpi.c File Reference	82
5.10.1	Function Documentation	82
5.10.1.1	cmd_dpi()	82
5.10.1.2	cmd_dpisel()	83
5.10.1.3	cmd_lift()	84
5.10.1.4	cmd_snap()	84
5.10.1.5	loaddpi()	85
5.10.1.6	printdpi()	86
5.10.1.7	savedpi()	87
5.10.1.8	updatedpi()	87
5.11	dpi.h File Reference	88
5.11.1	Function Documentation	89
5.11.1.1	cmd_dpi()	89
5.11.1.2	cmd_dpisel()	90

5.11.1.3	<code>cmd_lift()</code>	90
5.11.1.4	<code>cmd_snap()</code>	91
5.11.1.5	<code>loaddpi()</code>	91
5.11.1.6	<code>printdpi()</code>	92
5.11.1.7	<code>savedpi()</code>	93
5.11.1.8	<code>updatedpi()</code>	94
5.12	<code>extra_mac.c</code> File Reference	95
5.13	<code>firmware.c</code> File Reference	95
5.13.1	Macro Definition Documentation	96
5.13.1.1	<code>FW_MAXSIZE</code>	96
5.13.1.2	<code>FW_NOFILE</code>	96
5.13.1.3	<code>FW_OK</code>	96
5.13.1.4	<code>FW_USBFAIL</code>	96
5.13.1.5	<code>FW_WRONGDEV</code>	96
5.13.2	Function Documentation	97
5.13.2.1	<code>cmd_fwupdate()</code>	97
5.13.2.2	<code>fwupdate()</code>	98
5.13.2.3	<code>getfwversion()</code>	100
5.14	<code>firmware.h</code> File Reference	101
5.14.1	Function Documentation	101
5.14.1.1	<code>cmd_fwupdate()</code>	101
5.14.1.2	<code>getfwversion()</code>	102
5.15	<code>includes.h</code> File Reference	103
5.15.1	Macro Definition Documentation	105
5.15.1.1	<code>__FILE_NOPATH__</code>	105
5.15.1.2	<code>ckb_err</code>	105
5.15.1.3	<code>ckb_err_fn</code>	105
5.15.1.4	<code>ckb_err_nofile</code>	106
5.15.1.5	<code>ckb_fatal</code>	106
5.15.1.6	<code>ckb_fatal_fn</code>	106

5.15.1.7	ckb_fatal_nofile	106
5.15.1.8	ckb_info	107
5.15.1.9	ckb_info_fn	107
5.15.1.10	ckb_info_nofile	107
5.15.1.11	ckb_s_err	107
5.15.1.12	ckb_s_out	107
5.15.1.13	ckb_warn	108
5.15.1.14	ckb_warn_fn	108
5.15.1.15	ckb_warn_nofile	108
5.15.1.16	INDEX_OF	108
5.15.1.17	timespec_eq	109
5.15.1.18	timespec_ge	109
5.15.1.19	timespec_gt	109
5.15.1.20	timespec_le	109
5.15.1.21	timespec_lt	109
5.15.2	Typedef Documentation	110
5.15.2.1	uchar	110
5.15.2.2	ushort	110
5.15.3	Function Documentation	110
5.15.3.1	timespec_add()	110
5.16	input.c File Reference	110
5.16.1	Macro Definition Documentation	111
5.16.1.1	IS_WHEEL	111
5.16.2	Function Documentation	111
5.16.2.1	_cmd_macro()	112
5.16.2.2	cmd_bind()	113
5.16.2.3	cmd_macro()	114
5.16.2.4	cmd_rebind()	115
5.16.2.5	cmd_unbind()	115
5.16.2.6	freebind()	116

5.16.2.7	initbind()	116
5.16.2.8	inputupdate()	117
5.16.2.9	inputupdate_keys()	118
5.16.2.10	macromask()	120
5.16.2.11	updateindicators_kb()	121
5.17	input.h File Reference	122
5.17.1	Macro Definition Documentation	123
5.17.1.1	IS_MOD	123
5.17.2	Function Documentation	123
5.17.2.1	cmd_bind()	123
5.17.2.2	cmd_macro()	124
5.17.2.3	cmd_rebind()	124
5.17.2.4	cmd_unbind()	125
5.17.2.5	freebind()	125
5.17.2.6	initbind()	126
5.17.2.7	inputupdate()	126
5.17.2.8	os_inputclose()	127
5.17.2.9	os_inputopen()	128
5.17.2.10	os_keypress()	129
5.17.2.11	os_mousemove()	130
5.17.2.12	os_setupindicators()	131
5.17.2.13	updateindicators_kb()	132
5.18	input_linux.c File Reference	133
5.18.1	Function Documentation	133
5.18.1.1	_ledthread()	133
5.18.1.2	isync()	134
5.18.1.3	os_inputclose()	135
5.18.1.4	os_inputopen()	135
5.18.1.5	os_keypress()	137
5.18.1.6	os_mousemove()	138

5.18.1.7	<code>os_setupindicators()</code>	139
5.18.1.8	<code>uinputopen()</code>	139
5.19	<code>input_mac.c</code> File Reference	140
5.20	<code>input_mac_mouse.c</code> File Reference	141
5.21	<code>keymap.c</code> File Reference	141
5.21.1	Macro Definition Documentation	142
5.21.1.1	<code>BUTTON_HID_COUNT</code>	142
5.21.2	Function Documentation	142
5.21.2.1	<code>corsair_kbcopy()</code>	142
5.21.2.2	<code>corsair_mousecopy()</code>	143
5.21.2.3	<code>hid_kb_translate()</code>	143
5.21.2.4	<code>hid_mouse_translate()</code>	145
5.21.3	Variable Documentation	146
5.21.3.1	<code>keymap</code>	146
5.22	<code>keymap.h</code> File Reference	147
5.22.1	Data Structure Documentation	148
5.22.1.1	<code>struct key</code>	148
5.22.2	Macro Definition Documentation	149
5.22.2.1	<code>BTN_WHEELDOWN</code>	149
5.22.2.2	<code>BTN_WHEELUP</code>	149
5.22.2.3	<code>KEY_BACKSLASH_ISO</code>	149
5.22.2.4	<code>KEY_CORSAIR</code>	149
5.22.2.5	<code>KEY_NONE</code>	150
5.22.2.6	<code>KEY_UNBOUND</code>	150
5.22.2.7	<code>LED_DPI</code>	150
5.22.2.8	<code>LED_MOUSE</code>	150
5.22.2.9	<code>MOUSE_BUTTON_FIRST</code>	150
5.22.2.10	<code>MOUSE_EXTRA_FIRST</code>	151
5.22.2.11	<code>N_BUTTONS_EXTENDED</code>	151
5.22.2.12	<code>N_BUTTONS_HW</code>	151

5.22.2.13	N_KEY_ZONES	151
5.22.2.14	N_KEYBYTES_EXTENDED	151
5.22.2.15	N_KEYBYTES_HW	151
5.22.2.16	N_KEYBYTES_INPUT	152
5.22.2.17	N_KEYS_EXTENDED	152
5.22.2.18	N_KEYS_EXTRA	152
5.22.2.19	N_KEYS_HW	152
5.22.2.20	N_KEYS_INPUT	152
5.22.2.21	N_MOUSE_ZONES	153
5.22.2.22	N_MOUSE_ZONES_EXTENDED	153
5.22.2.23	SCAN_KBD	153
5.22.2.24	SCAN_MOUSE	153
5.22.2.25	SCAN_SILENT	153
5.22.3	Function Documentation	153
5.22.3.1	corsair_kbcopy()	154
5.22.3.2	corsair_mousecopy()	154
5.22.3.3	hid_kb_translate()	155
5.22.3.4	hid_mouse_translate()	157
5.22.4	Variable Documentation	158
5.22.4.1	keymap	158
5.23	keymap_mac.h File Reference	158
5.24	led.c File Reference	159
5.24.1	Function Documentation	159
5.24.1.1	cmd_iauto()	159
5.24.1.2	cmd_inotify()	160
5.24.1.3	cmd_ioff()	161
5.24.1.4	cmd_ion()	161
5.24.1.5	cmd_rgb()	162
5.24.1.6	has_key()	162
5.24.1.7	iselect()	163

5.24.1.8	printrgb()	164
5.25	led.h File Reference	165
5.25.1	Function Documentation	166
5.25.1.1	cmd_iauto()	166
5.25.1.2	cmd_inotify()	167
5.25.1.3	cmd_ioff()	168
5.25.1.4	cmd_ion()	168
5.25.1.5	cmd_rgb()	169
5.25.1.6	loadrgb_kb()	169
5.25.1.7	loadrgb_mouse()	171
5.25.1.8	printrgb()	172
5.25.1.9	savergb_kb()	174
5.25.1.10	savergb_mouse()	175
5.25.1.11	updatergb_kb()	176
5.25.1.12	updatergb_mouse()	177
5.26	led_keyboard.c File Reference	178
5.26.1	Macro Definition Documentation	201
5.26.1.1	BR1	201
5.26.1.2	BR2	202
5.26.1.3	BR4	202
5.26.1.4	O0	202
5.26.1.5	O1	202
5.26.1.6	O2	202
5.26.1.7	O3	202
5.26.1.8	O4	203
5.26.1.9	O5	203
5.26.1.10	O6	203
5.26.1.11	O7	203
5.26.1.12	O8	203
5.26.2	Function Documentation	203

5.26.2.1	loadrgb_kb()	204
5.26.2.2	makergb_512()	206
5.26.2.3	makergb_full()	206
5.26.2.4	ordered8to3()	207
5.26.2.5	quantize8to3()	208
5.26.2.6	rgbcmp()	208
5.26.2.7	savergb_kb()	209
5.26.2.8	updatergb_kb()	210
5.26.3	Variable Documentation	212
5.26.3.1	bit_reverse_table	212
5.27	led_mouse.c File Reference	240
5.27.1	Function Documentation	240
5.27.1.1	isblack()	240
5.27.1.2	loadrgb_mouse()	241
5.27.1.3	rgbcmp()	242
5.27.1.4	savergb_mouse()	243
5.27.1.5	updatergb_mouse()	243
5.28	main.c File Reference	244
5.28.1	Function Documentation	245
5.28.1.1	localecase()	246
5.28.1.2	main()	246
5.28.1.3	quit()	249
5.28.1.4	quitWithLock()	250
5.28.1.5	restart()	251
5.28.1.6	sighandler()	252
5.28.1.7	sighandler2()	253
5.28.1.8	timespec_add()	253
5.28.2	Variable Documentation	254
5.28.2.1	features_mask	254
5.28.2.2	hwload_mode	254

5.28.2.3	main_ac	254
5.28.2.4	main_av	254
5.28.2.5	reset_stop	255
5.29	notify.c File Reference	255
5.29.1	Macro Definition Documentation	256
5.29.1.1	HW_STANDARD	256
5.29.1.2	HWMODE_OR_RETURN	256
5.29.2	Function Documentation	256
5.29.2.1	_cmd_get()	257
5.29.2.2	cmd_get()	259
5.29.2.3	cmd_notify()	260
5.29.2.4	cmd_restart()	261
5.29.2.5	nprintf()	261
5.29.2.6	nprintind()	262
5.29.2.7	nprintkey()	263
5.29.2.8	restart()	264
5.30	notify.h File Reference	265
5.30.1	Function Documentation	266
5.30.1.1	cmd_get()	266
5.30.1.2	cmd_notify()	267
5.30.1.3	cmd_restart()	267
5.30.1.4	nprintf()	268
5.30.1.5	nprintind()	268
5.30.1.6	nprintkey()	269
5.31	os.h File Reference	270
5.31.1	Macro Definition Documentation	271
5.31.1.1	_DEFAULT_SOURCE	271
5.31.1.2	_GNU_SOURCE	271
5.31.1.3	euid_guard_start	271
5.31.1.4	euid_guard_stop	272

5.31.1.5	UINPUT_VERSION	272
5.32	profile.c File Reference	272
5.32.1	Function Documentation	273
5.32.1.1	_freeprofile()	273
5.32.1.2	allocprofile()	274
5.32.1.3	cmd_erase()	275
5.32.1.4	cmd_eraseprofile()	276
5.32.1.5	cmd_id()	276
5.32.1.6	cmd_name()	277
5.32.1.7	cmd_profileid()	278
5.32.1.8	cmd_profilename()	278
5.32.1.9	freemode()	279
5.32.1.10	freeprofile()	280
5.32.1.11	gethwmodename()	280
5.32.1.12	gethwprofilename()	281
5.32.1.13	getid()	282
5.32.1.14	getmodename()	283
5.32.1.15	getprofilename()	283
5.32.1.16	hwtonative()	284
5.32.1.17	initmode()	285
5.32.1.18	loadprofile()	286
5.32.1.19	nativetohw()	286
5.32.1.20	printname()	287
5.32.1.21	setid()	288
5.32.1.22	u16dec()	289
5.32.1.23	u16enc()	289
5.32.1.24	urldecode2()	290
5.32.1.25	urlencode2()	291
5.32.2	Variable Documentation	292
5.32.2.1	utf16to8	292

5.32.2.2	utf8to16	292
5.33	profile.h File Reference	292
5.33.1	Macro Definition Documentation	293
5.33.1.1	hwloadprofile	293
5.33.2	Function Documentation	293
5.33.2.1	allocprofile()	294
5.33.2.2	cmd_erase()	295
5.33.2.3	cmd_eraseprofile()	295
5.33.2.4	cmd_hwload_kb()	296
5.33.2.5	cmd_hwload_mouse()	297
5.33.2.6	cmd_hwsave_kb()	298
5.33.2.7	cmd_hwsave_mouse()	299
5.33.2.8	cmd_id()	300
5.33.2.9	cmd_name()	301
5.33.2.10	cmd_profileid()	302
5.33.2.11	cmd_profilename()	302
5.33.2.12	freeprofile()	303
5.33.2.13	gethwmodename()	304
5.33.2.14	gethwprofilename()	304
5.33.2.15	getid()	305
5.33.2.16	getmodename()	306
5.33.2.17	getprofilename()	307
5.33.2.18	hwtonative()	307
5.33.2.19	loadprofile()	308
5.33.2.20	nativetohw()	309
5.33.2.21	setid()	309
5.34	profile_keyboard.c File Reference	310
5.34.1	Function Documentation	311
5.34.1.1	cmd_hwload_kb()	311
5.34.1.2	cmd_hwsave_kb()	312

5.34.1.3	hwloadmode()	313
5.35	profile_mouse.c File Reference	314
5.35.1	Function Documentation	314
5.35.1.1	cmd_hwload_mouse()	314
5.35.1.2	cmd_hwsave_mouse()	315
5.36	structures.h File Reference	316
5.36.1	Data Structure Documentation	318
5.36.1.1	struct usbid	318
5.36.1.2	struct macroaction	319
5.36.1.3	struct keymacro	319
5.36.1.4	struct binding	320
5.36.1.5	struct dpiset	321
5.36.1.6	struct lighting	322
5.36.1.7	struct usbmode	323
5.36.1.8	struct usbprofile	325
5.36.1.9	struct hwprofile	325
5.36.1.10	struct usbinput	326
5.36.1.11	struct usbdevice	327
5.36.2	Macro Definition Documentation	329
5.36.2.1	CLEAR_KEYBIT	329
5.36.2.2	DPI_COUNT	329
5.36.2.3	FEAT_ADJRATE	329
5.36.2.4	FEAT_ANSI	330
5.36.2.5	FEAT_BIND	330
5.36.2.6	FEAT_COMMON	330
5.36.2.7	FEAT_FWUPDATE	330
5.36.2.8	FEAT_FWVERSION	330
5.36.2.9	FEAT_HWLOAD	331
5.36.2.10	FEAT_ISO	331
5.36.2.11	FEAT_LMASK	331

5.36.2.12 FEAT_MONOCHROME	331
5.36.2.13 FEAT_MOUSEACCEL	331
5.36.2.14 FEAT_NOTIFY	332
5.36.2.15 FEAT_POLLRATE	332
5.36.2.16 FEAT_RGB	332
5.36.2.17 FEAT_STD_NRGB	332
5.36.2.18 FEAT_STD_RGB	332
5.36.2.19 HAS_ANY_FEATURE	333
5.36.2.20 HAS_FEATURES	333
5.36.2.21 HWMODE_K70	333
5.36.2.22 HWMODE_K95	333
5.36.2.23 HWMODE_MAX	333
5.36.2.24 I_CAPS	334
5.36.2.25 I_NUM	334
5.36.2.26 I_SCROLL	334
5.36.2.27 IFACE_MAX	334
5.36.2.28 KB_NAME_LEN	334
5.36.2.29 LIFT_MAX	335
5.36.2.30 LIFT_MIN	335
5.36.2.31 MACRO_MAX	335
5.36.2.32 MD_NAME_LEN	335
5.36.2.33 MODE_COUNT	335
5.36.2.34 MSG_SIZE	336
5.36.2.35 NEEDS_FW_UPDATE	336
5.36.2.36 OUTFIFO_MAX	336
5.36.2.37 PR_NAME_LEN	336
5.36.2.38 SCROLL_ACCELERATED	336
5.36.2.39 SCROLL_MAX	337
5.36.2.40 SCROLL_MIN	337
5.36.2.41 SERIAL_LEN	337

5.36.2.42 SET_KEYBIT	337
5.36.3 Variable Documentation	337
5.36.3.1 vtable_keyboard	337
5.36.3.2 vtable_keyboard_nonrgb	338
5.36.3.3 vtable_mouse	338
5.37 usb.c File Reference	338
5.37.1 Function Documentation	339
5.37.1.1 _resetusb()	339
5.37.1.2 _setupusb()	340
5.37.1.3 _usbrecv()	344
5.37.1.4 _usbsend()	346
5.37.1.5 closeusb()	348
5.37.1.6 devmain()	349
5.37.1.7 get_vtable()	352
5.37.1.8 product_str()	353
5.37.1.9 revertusb()	354
5.37.1.10 setupusb()	355
5.37.1.11 usb_tryreset()	356
5.37.1.12 vendor_str()	357
5.37.2 Variable Documentation	358
5.37.2.1 features_mask	358
5.37.2.2 hwload_mode	358
5.37.2.3 reset_stop	358
5.37.2.4 usbmutex	358
5.38 usb.h File Reference	359
5.38.1 Detailed Description	362
5.38.2 Macro Definition Documentation	363
5.38.2.1 DELAY_LONG	363
5.38.2.2 DELAY_MEDIUM	363
5.38.2.3 DELAY_SHORT	363

5.38.2.4 IS_FULLRANGE	364
5.38.2.5 IS_K65	364
5.38.2.6 IS_K70	364
5.38.2.7 IS_K95	364
5.38.2.8 IS_M65	365
5.38.2.9 IS_MONOCHROME	365
5.38.2.10 IS_MONOCHROME_DEV	365
5.38.2.11 IS_MOUSE	365
5.38.2.12 IS_MOUSE_DEV	366
5.38.2.13 IS_RGB	366
5.38.2.14 IS_RGB_DEV	366
5.38.2.15 IS_SABRE	366
5.38.2.16 IS_SCIMITAR	367
5.38.2.17 IS_STRAFE	367
5.38.2.18 NK95_HWOFF	367
5.38.2.19 NK95_HWON	367
5.38.2.20 NK95_M1	368
5.38.2.21 NK95_M2	368
5.38.2.22 NK95_M3	368
5.38.2.23 nk95cmd	368
5.38.2.24 P_HARPOON	368
5.38.2.25 P_HARPOON_STR	369
5.38.2.26 P_K65	369
5.38.2.27 P_K65_LUX	369
5.38.2.28 P_K65_LUX_STR	369
5.38.2.29 P_K65_NRGB	369
5.38.2.30 P_K65_NRGB_STR	370
5.38.2.31 P_K65_RFIRE	370
5.38.2.32 P_K65_RFIRE_STR	370
5.38.2.33 P_K65_STR	370

5.38.2.34 P_K70	370
5.38.2.35 P_K70_LUX	370
5.38.2.36 P_K70_LUX_NRGB	371
5.38.2.37 P_K70_LUX_NRGB_STR	371
5.38.2.38 P_K70_LUX_STR	371
5.38.2.39 P_K70_NRGB	371
5.38.2.40 P_K70_NRGB_STR	371
5.38.2.41 P_K70_RFIRE	371
5.38.2.42 P_K70_RFIRE_NRGB	372
5.38.2.43 P_K70_RFIRE_NRGB_STR	372
5.38.2.44 P_K70_RFIRE_STR	372
5.38.2.45 P_K70_STR	372
5.38.2.46 P_K95	372
5.38.2.47 P_K95_NRGB	372
5.38.2.48 P_K95_NRGB_STR	373
5.38.2.49 P_K95_PLATINUM	373
5.38.2.50 P_K95_PLATINUM_STR	373
5.38.2.51 P_K95_STR	373
5.38.2.52 P_M65	373
5.38.2.53 P_M65_PRO	373
5.38.2.54 P_M65_PRO_STR	374
5.38.2.55 P_M65_STR	374
5.38.2.56 P_SABRE_L	374
5.38.2.57 P_SABRE_L_STR	374
5.38.2.58 P_SABRE_N	374
5.38.2.59 P_SABRE_N_STR	374
5.38.2.60 P_SABRE_O	375
5.38.2.61 P_SABRE_O2	375
5.38.2.62 P_SABRE_O2_STR	375
5.38.2.63 P_SABRE_O_STR	375

5.38.2.64	P_SCIMITAR	375
5.38.2.65	P_SCIMITAR_PRO	376
5.38.2.66	P_SCIMITAR_PRO_STR	376
5.38.2.67	P_SCIMITAR_STR	376
5.38.2.68	P_STRAFE	376
5.38.2.69	P_STRAFE_NRGB	376
5.38.2.70	P_STRAFE_NRGB_STR	377
5.38.2.71	P_STRAFE_STR	377
5.38.2.72	resetusb	377
5.38.2.73	USB_DELAY_DEFAULT	377
5.38.2.74	usbrecv	377
5.38.2.75	usbsend	378
5.38.2.76	V_CORSAIR	378
5.38.2.77	V_CORSAIR_STR	378
5.38.3	Function Documentation	379
5.38.3.1	_nk95cmd()	379
5.38.3.2	_resetusb()	380
5.38.3.3	_usbrecv()	381
5.38.3.4	_usbsend()	383
5.38.3.5	closeusb()	385
5.38.3.6	os_closeusb()	387
5.38.3.7	os_inputmain()	388
5.38.3.8	os_resetusb()	391
5.38.3.9	os_sendindicators()	393
5.38.3.10	os_setupusb()	394
5.38.3.11	os_usbrecv()	395
5.38.3.12	os_usbsend()	397
5.38.3.13	product_str()	399
5.38.3.14	revertusb()	401
5.38.3.15	setupusb()	402

5.38.3.16	usb_tryreset()	403
5.38.3.17	usbkill()	405
5.38.3.18	usbmain()	405
5.38.3.19	vendor_str()	407
5.39	usb_linux.c File Reference	408
5.39.1	Data Structure Documentation	410
5.39.1.1	struct _model	410
5.39.2	Macro Definition Documentation	410
5.39.2.1	N_MODELS	410
5.39.2.2	TEST_RESET	411
5.39.3	Function Documentation	411
5.39.3.1	_nk95cmd()	411
5.39.3.2	os_closeusb()	412
5.39.3.3	os_inputmain()	413
5.39.3.4	os_resetusb()	416
5.39.3.5	os_sendindicators()	418
5.39.3.6	os_setupusb()	419
5.39.3.7	os_usbrecv()	420
5.39.3.8	os_usbsend()	422
5.39.3.9	strtrim()	424
5.39.3.10	udev_enum()	424
5.39.3.11	usb_add_device()	426
5.39.3.12	usb_rm_device()	427
5.39.3.13	usbadd()	428
5.39.3.14	usbclaim()	430
5.39.3.15	usbkill()	431
5.39.3.16	usbmain()	432
5.39.3.17	usbunclaim()	434
5.39.4	Variable Documentation	435
5.39.4.1	kbsyspath	435
5.39.4.2	models	435
5.39.4.3	udev	436
5.39.4.4	udevthread	436
5.39.4.5	usbthread	436
5.40	usb_mac.c File Reference	436

Chapter 1

Todo List

Global `_usbsend` (usbdevice *kb, const uchar *messages, int count, const char *file, int line)

A lot of different conditions are combined in this code. Don't think, it is good in every combination...

Check whether this is the same in the macOS variant. It is not dramatic, but if errors occur, it can certainly irritate the devices completely if they receive incomplete data streams. Do we have errors with the messages "Wrote YY bytes (expected 64)" in the system logs? If not, we do not need to look any further.

Global `closeusb` (usbdevice *kb)

What is not yet comprehensible is the call to `updateconnected()` BEFORE `os_closeusb()`. Should that be in the other sequence? Or is `updateconnected()` not displaying the connected usb devices, but the representation which uinput devices are loaded? Questions about questions ...

Global `devmain` (usbdevice *kb)

Hope to find the need for dmux usage later.

Should this function be declared as pthread_t* function, because of the definition of pthread_create? But void* works also...

`readcmd()` gets a **line**, not **lines**. Have a look on that later.

Is the condition IS_CONNECTED valid? What functions change the condition for the macro?

Global `get_vtable` (short vendor, short product)

Is the last point really a good decision and always correct?

Global `os_inputmain` (void *context)

This function is a collection of many tasks. It should be divided into several sub-functions for the sake of greater convenience:

Global `os_resetusb` (usbdevice *kb, const char *file, int line)

it seems that no one wants to try the reset again. But I've seen it somewhere...

Global `os_setupusb` (usbdevice *kb)

in these modules a pullrequest is outstanding

Global `os_usbsend` (usbdevice *kb, const uchar *out_msg, int is_rcv, const char *file, int line)

Since the handling of endpoints has already led to problems elsewhere, this implementation is extremely hardware-dependent and critical!

Eg. the new keyboard K95PLATINUMRGB has a version number significantly less than 2.0 - will it run with this implementation?

Global `product_str` (short product)

There are macros defined in `usb.h` to detect all the combinations below. the only difference is the parameter: The macros need the `kb*`, `product_str()` needs the *product ID*

Global [revertusb](#) (usbdevice *kb)

Why is this useful? Are there problems seen with deactivating a device with older fw-version??? Why isn't this an error indicating reason and we return success (0)?

The return value of [nk95cmd\(\)](#) is ignored (but sending the ioctl may produce an error and `_nk95_cmd` will indicate this), instead [revertusb\(\)](#) returns success in any case.

Global [udevthread](#)

These two thread variables seem to be unused: `usbthread`, `udevthread`

Global [udevthread](#)

These two thread variables seem to be unused: `usbthread`, `udevthread`

Global [usb_add_device](#) (struct udev_device *dev)

So why the hell not a transformation between the string and the short presentation? Lets check if the string representation is used elsewhere.

Global [usb_tryreset](#) (usbdevice *kb)

Why does [usb_tryreset\(\)](#) hide the information returned from [resetusb\(\)](#)? Isn't it needed by the callers?

Global [usbmain](#) ()

Why isn't missing of uinput a fatal error?

lae. here the work has to go on...

Global [usbmutex](#)

We should have a look why this mutex is never used.

Chapter 2

Data Structure Index

2.1 Data Structures

Here are the data structures with brief descriptions:

devcmd.__unnamed__	7
--	---

Chapter 3

File Index

3.1 File List

Here is a list of all files with brief descriptions:

command.c	13
command.h	19
device.c	28
device.h	32
device_keyboard.c	45
device_mouse.c	49
device_vtable.c	53
devnode.c	56
devnode.h	72
dpi.c	82
dpi.h	88
extra_mac.c	95
firmware.c	95
firmware.h	101
includes.h	103
input.c	110
input.h	122
input_linux.c	133
input_mac.c	140
input_mac_mouse.c	141
keymap.c	141
keymap.h	147
keymap_mac.h	158
led.c	159
led.h	165
led_keyboard.c	178
led_mouse.c	240
main.c	244
notify.c	255
notify.h	265
os.h	270
profile.c	272
profile.h	292
profile_keyboard.c	310
profile_mouse.c	314

structures.h	316
usb.c	338
usb.h	
Definitions for using USB interface	359
usb_linux.c	408
usb_mac.c	436

Chapter 4

Data Structure Documentation

4.1 devcmd.__unnamed__ Struct Reference

Collaboration diagram for devcmd.__unnamed__:

devcmd.__unnamed__
+ hwload + hwsave + fwupdate + pollrate + active + idle + erase + eraseprofile + name + profilename and 26 more...

Data Fields

- [cmdhandler_io hwload](#)
- [cmdhandler_io hwsave](#)
- [cmdhandler_io fwupdate](#)
- [cmdhandler_io pollrate](#)
- [cmdhandler_io active](#)
- [cmdhandler_io idle](#)
- [cmdhandler erase](#)
- [cmdhandler eraseprofile](#)
- [cmdhandler name](#)

- [cmdhandler profilename](#)
- [cmdhandler id](#)
- [cmdhandler profileid](#)
- [cmdhandler rgb](#)
- [cmdhandler ioff](#)
- [cmdhandler ion](#)
- [cmdhandler iauto](#)
- [cmdhandler bind](#)
- [cmdhandler unbind](#)
- [cmdhandler rebind](#)
- [cmdhandler_mac macro](#)
- [cmdhandler_mac dpi](#)
- [cmdhandler dpisel](#)
- [cmdhandler lift](#)
- [cmdhandler snap](#)
- [cmdhandler notify](#)
- [cmdhandler inotify](#)
- [cmdhandler get](#)
- [cmdhandler restart](#)
- [int\(* start \)\(usbdevice *kb, int makeactive\)](#)
- [void\(* setmodeindex \)\(usbdevice *kb, int index\)](#)
- [void\(* allocprofile \)\(usbdevice *kb\)](#)
- [int\(* loadprofile \)\(usbdevice *kb\)](#)
- [void\(* freeprofile \)\(usbdevice *kb\)](#)
- [int\(* updatergb \)\(usbdevice *kb, int force\)](#)
- [void\(* updateindicators \)\(usbdevice *kb, int force\)](#)
- [int\(* updatedpi \)\(usbdevice *kb, int force\)](#)

4.1.1 Detailed Description

Definition at line 78 of file command.h.

4.1.2 Field Documentation

4.1.2.1 active

4.1.2.2 allocprofile

4.1.2.3 bind

4.1.2.4 dpi

4.1.2.5 dpisel

4.1.2.6 erase

4.1.2.7 eraseprofile

4.1.2.8 freeprofile

4.1.2.9 fwupdate

4.1.2.10 get

4.1.2.11 hwload

4.1.2.12 hwsave

4.1.2.13 iauto

4.1.2.14 id

4.1.2.15 idle

4.1.2.16 inotify

4.1.2.17 ioff

4.1.2.18 ion

4.1.2.19 lift

4.1.2.20 loadprofile

4.1.2.21 macro

4.1.2.22 name

4.1.2.23 notify

4.1.2.24 pollrate

4.1.2.25 profileid

4.1.2.26 profilename

4.1.2.27 rebind

4.1.2.28 restart

4.1.2.29 rgb

4.1.2.30 setmodeindex

4.1.2.31 snap

4.1.2.32 start

4.1.2.33 unbind

4.1.2.34 updatedpi

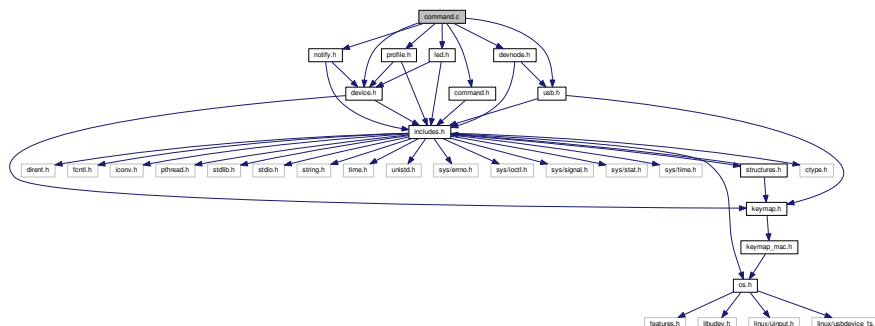
4.1.2.35 updateindicators

4.1.2.36 updatergb

The documentation for this struct was generated from the following files:

File Documentation

```
#include "command.h"
#include "device.h"
#include "devnode.h"
#include "led.h"
#include "notify.h"
#include "profile.h"
#include "usb.h"
Include dependency graph for command.c:
```



- #define TRY_WITH_RESET(action)

- int readcmd (usbdevice *kb, const char *line)

- static const char *const cmd_strings [(CMD_LAST - CMD_FIRST+2) - 1]

5.1.1 Macro Definition Documentation

5.1.1.1 TRY_WITH_RESET

```
#define TRY_WITH_RESET(  
    action )
```

Value:

```
while(action){  
    if(usb_tryreset(kb)){  
        free(word);  
        return 1;  
    }  
}
```

Definition at line 58 of file command.c.

Referenced by readcmd().

5.1.2 Function Documentation

5.1.2.1 readcmd()

```
int readcmd (  
    usbdevice * kb,  
    const char * line )
```

Because length of word is length of line + 1, there should be no problem with buffer overflow.

Definition at line 67 of file command.c.

References ACCEL, ACTIVE, usbdevice::active, BIND, CMD_COUNT, CMD_FIRST, cmd_strings, usbprofile::currentmode, DELAY, usbdevice::delay, DITHER, usbdevice::dither, devcmd::do_cmd, devcmd::do_io, devcmd::do_macro, DPI, DPISEL, ERASE, ERASEPROFILE, FEAT_ANSI, FEAT_BIND, FEAT_ISO, FEAT_LMASK, FEAT_MOUSEACCEL, FEAT_NOTIFY, usbdevice::features, lighting::forceupdate, FPS, FWUPDATE, GET, HAS_FEATURES, HWLOAD, HWSAVE, IAUTO, ID, IDLE, INDEX_OF, INOTIFY, IOFF, ION, IS_FULLRANGE, IS_MOUSE_DEV, keymap, LAYOUT, LIFT, usbmode::light, MACRO, mknotifynode(), MODE, usbprofile::mode, MODE_COUNT, N_KEYS_EXTENDED, NAME, NEEDS_FW_UPDATE, NONE, NOTIFY, NOTIFYOFF, NOTIFYON, OUTFIFO_MAX, POLLRATE, usbdevice::profile, PROFILEID, PROFILENAME, REBIND, RESTART, RGB, rmnotifynode(), SCROLL_ACCELERATED, SCROLL_MAX, SCROLL_MIN, SCROLLSPEED, SNAP, SWITCH, TRY_WITH_RESET, UNBIND, usbdevice::usbdelay, and usbdevice::vtable.

Referenced by devmain().

```

67     {
68         char* word = malloc(strlen(line) + 1);
69         int wordlen;
70         const char* newline = 0;
71         const devcmd* vt = kb->vtable;
72         usbprofile* profile = kb->profile;
73         usbmode* mode = 0;
74         int notifynumber = 0;
75         // Read words from the input
76         cmd command = NONE;
77         while(sscanf(line, "%s%n", word, &wordlen) == 1){
78             line += wordlen;
79             // If we passed a newline, reset the context
80             if(line > newline){
81                 mode = profile->currentmode;
82                 command = NONE;
83                 notifynumber = 0;
84                 newline = strchr(line, '\n');
85                 if(!newline)
86                     newline = line + strlen(line);
87             }
88             // Check for a command word
89             for(int i = 0; i < CMD_COUNT - 1; i++){
90                 if(!strcmp(word, cmd_strings[i])){
91                     command = i + CMD_FIRST;
92 #ifndef OS_MAC
93                     // Layout and mouse acceleration aren't used on Linux; ignore
94                     if(command == LAYOUT || command == ACCEL || command ==
95                        SCROLLSPEED)
96                         command = NONE;
97 #endif
98                     // Most commands require parameters, but a few are actions in and of themselves
99                     if(command != SWITCH
100                        && command != HWLOAD && command != HWSAVE
101                        && command != ACTIVE && command != IDLE
102                        && command != ERASE && command != ERASEPROFILE
103                        && command != RESTART)
104                         goto next_loop;
105                     break;
106                 }
107             }
108             // Set current notification node when given @number
109             int newnotify;
110             if(sscanf(word, "%u", &newnotify) == 1 && newnotify < OUTFIFO_MAX){
111                 notifynumber = newnotify;
112                 continue;
113             }
114             // Reject unrecognized commands. Reject bind or notify related commands if the keyboard doesn't
115             // have the feature enabled.
116             if(command == NONE
117                || ((!HAS_FEATURES(kb, FEAT_BIND) && (command ==
118                BIND || command == UNBIND || command == REBIND || command ==
119                MACRO || command == DELAY))
120                || (!HAS_FEATURES(kb, FEAT_NOTIFY) && command ==
121                NOTIFY))){
122                 next_loop:
123                 continue;
124             }
125             // Reject anything not related to fwupdate if device has a bricked FW
126             if(NEEDS_FW_UPDATE(kb) && command != FWUPDATE && command !=
127                NOTIFYON && command != NOTIFYOFF)
128                 continue;
129             // Specially handled commands - these are available even when keyboard is IDLE
130             switch(command){
131             case NOTIFYON: {
132                 // Notification node on
133                 int notify;
134                 if(sscanf(word, "%u", &notify) == 1)
135                     mknotifynode(kb, notify);
136                 continue;
137             } case NOTIFYOFF: {
138                 // Notification node off
139                 int notify;
140                 if(sscanf(word, "%u", &notify) == 1 && notify != 0) // notify0 can't be removed
141                     rmnotifynode(kb, notify);
142                 continue;
143             } case GET:
144                 // Output data to notification node
145                 vt->get(kb, mode, notifynumber, 0, word);
146                 continue;
147             case LAYOUT:
148                 // OSX: switch ANSI/ISO keyboard layout
149                 if(!strcmp(word, "ansi"))
150                     kb->features = (kb->features & ~FEAT_LMASK) |

```

```

    FEAT_ANSI;
148     else if(!strcmp(word, "iso"))
149         kb->features = (kb->features & ~FEAT_LMASK) |
        FEAT_ISO;
150     continue;
151 #ifdef OS_MAC
152     case ACCEL:
153         // OSX mouse acceleration on/off
154         if(!strcmp(word, "on"))
155             kb->features |= FEAT_MOUSEACCEL;
156         else if(!strcmp(word, "off"))
157             kb->features &= ~FEAT_MOUSEACCEL;
158         continue;
159     case SCROLLSPEED: {
160         int newscroll;
161         if(sscanf(word, "%d", &newscroll) != 1)
162             break;
163         if(newscroll < SCROLL_MIN)
164             newscroll = SCROLL_ACCELERATED;
165         if(newscroll > SCROLL_MAX)
166             newscroll = SCROLL_MAX;
167         kb->scroll_rate = newscroll;
168         continue;
169     }
170 #endif
171     case MODE: {
172         // Select a mode number (1 - 6)
173         int newmode;
174         if(sscanf(word, "%u", &newmode) == 1 && newmode > 0 && newmode <=
        MODE_COUNT)
175             mode = profile->mode + newmode - 1;
176             continue;
177     }
178     case FPS: {
179         // USB command delay (2 - 10ms)
180         uint framerate;
181         if(sscanf(word, "%u", &framerate) == 1 && framerate > 0){
182             // Not all devices require the same number of messages per frame; select delay
        appropriately
183             uint per_frame = IS_MOUSE_DEV(kb) ? 2 : IS_FULLRANGE(kb) ? 14 : 5;
184             uint delay = 1000 / framerate / per_frame;
185             if(delay < 2)
186                 delay = 2;
187             else if(delay > 10)
188                 delay = 10;
189             kb->usbdelay = delay;
190         }
191         continue;
192     }
193     case DITHER: {
194         // 0: No dither, 1: Ordered dither.
195         uint dither;
196         if(sscanf(word, "%u", &dither) == 1 && dither <= 1){
197             kb->dither = dither;
198             profile->currentmode->light.forceupdate = 1;
199             mode->light.forceupdate = 1;
200         }
201         continue;
202     }
203     case DELAY:
204         kb->delay = (!strcmp(word, "on")); // independendant from parameter to handle false
        commands like "delay off"
205         continue;
206     case RESTART: {
207         char mybuffer[] = "no reason specified";
208         if (sscanf(line, "%[^\\n]", word) == -1) {
209             word = mybuffer;
210         }
211         vt->do_cmd[command](kb, mode, notifiynumber, 0, word);
212         continue;
213     }
214
215     default:;
216 }
217
218 // If a keyboard is inactive, it must be activated before receiving any other commands
219 if(!kb->active){
220     if(command == ACTIVE)
221         TRY_WITH_RESET(vt->active(kb, mode, notifiynumber, 0, 0));
222     continue;
223 }
224 // Specially handled commands only available when keyboard is ACTIVE
225 switch(command){
226 case IDLE:
227     TRY_WITH_RESET(vt->idle(kb, mode, notifiynumber, 0, 0));
228     continue;
229 case SWITCH:

```

```

230         if(profile->currentmode != mode){
231             profile->currentmode = mode;
232             // Set mode light for non-RGB K95
233             int index = INDEX_OF(mode, profile->mode);
234             vt->setmodeindex(kb, index);
235         }
236         continue;
237     case HWLOAD: case HWSAVE:{
238         char delay = kb->usbdelay;
239         // Ensure delay of at least 10ms as the device can get overwhelmed otherwise
240         if(delay < 10)
241             kb->usbdelay = 10;
242         // Try to load/save the hardware profile. Reset on failure, disconnect if reset fails.
243         TRY_WITH_RESET(vt->do_io[command](kb, mode, notifynumber, 1, 0));
244         // Re-send the current RGB state as it sometimes gets scrambled
245         TRY_WITH_RESET(vt->updatergb(kb, 1));
246         kb->usbdelay = delay;
247         continue;
248     }
249     case FWUPDATE:
250         // FW update parses a whole word. Unlike hwload/hwsave, there's no try again on failure.
251         if(vt->fwupdate(kb, mode, notifynumber, 0, word)){
252             free(word);
253             return 1;
254         }
255         continue;
256     case POLLRATE: {
257         uint rate;
258         if(sscanf(word, "%u", &rate) == 1 && (rate == 1 || rate == 2 || rate == 4 || rate == 8))
259             TRY_WITH_RESET(vt->pollrate(kb, mode, notifynumber, rate, 0));
260         continue;
261     }
262     case ERASEPROFILE:
263         // Erase the current profile
264         vt->eraseprofile(kb, mode, notifynumber, 0, 0);
265         // Update profile/mode pointers
266         profile = kb->profile;
267         mode = profile->currentmode;
268         continue;
269     case ERASE: case NAME: case IOFF: case ION: case IAUTO: case
INOTIFY: case PROFILENAME: case ID: case PROFILEID: case
DPISEL: case LIFT: case SNAP:
270         // All of the above just parse the whole word
271         vt->do_cmd[command](kb, mode, notifynumber, 0, word);
272         continue;
273     case RGB: {
274         // RGB command has a special response for a single hex constant
275         int r, g, b;
276         if(sscanf(word, "%02x%02x%02x", &r, &g, &b) == 3){
277             // Set all keys
278             for(int i = 0; i < N_KEYS_EXTENDED; i++){
279                 vt->rgb(kb, mode, notifynumber, i, word);
280                 continue;
281             }
282             break;
283         }
284     case MACRO:
285         if(!strcmp(word, "clear")){
286             // Macro has a special clear command
287             vt->macro(kb, mode, notifynumber, 0, 0);
288             continue;
289         }
290         break;
291     default:;
292     }
293     // For anything else, split the parameter at the colon
294     int left = -1;
295     sscanf(word, "%*[^:]%n", &left);
296     if(left <= 0)
297         continue;
298     const char* right = word + left;
299     if(right[0] == ':')
300         right++;
301     // Macros and DPI have a separate left-side handler
302     if(command == MACRO || command == DPI){
303         word[left] = 0;
304         vt->do_macro[command](kb, mode, notifynumber, word, right);
305         continue;
306     }
307     // Scan the left side for key names and run the requested command
308     int position = 0, field = 0;
309     char keyname[11];
310     while(position < left && sscanf(word + position, "%10[^:,%n", keyname, &field) == 1){
311         int keycode;
312         if(!strcmp(keyname, "all")){
313             // Set all keys
314             for(int i = 0; i < N_KEYS_EXTENDED; i++){

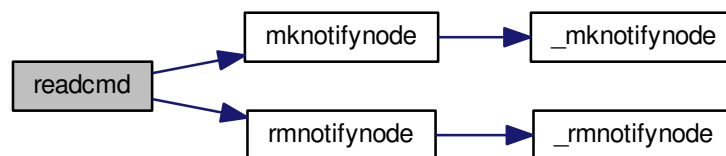
```

```

315         vt->do_cmd[command](kb, mode, notifynumber, i, right);
316     } else if((sscanf(keyname, "%d", &keycode) && keycode >= 0 && keycode <
N_KEYS_EXTENDED)
317         || (sscanf(keyname, "%x", &keycode) && keycode >= 0 && keycode <
N_KEYS_EXTENDED)){
318         // Set a key numerically
319         vt->do_cmd[command](kb, mode, notifynumber, keycode, right);
320     } else {
321         // Find this key in the keymap
322         for(unsigned i = 0; i < N_KEYS_EXTENDED; i++){
323             if(keymap[i].name && !strcmp(keyname, keymap[i].name)){
324                 vt->do_cmd[command](kb, mode, notifynumber, i, right);
325                 break;
326             }
327         }
328     }
329     if(word[position += field] == ',')
330         position++;
331 }
332 }
333
334 // Finish up
335 if(!NEEDS_FW_UPDATE(kb)){
336     TRY_WITH_RESET(vt->updatergb(kb, 0));
337     TRY_WITH_RESET(vt->updatedpi(kb, 0));
338 }
339 free(word);
340 return 0;
341 }

```

Here is the call graph for this function:



Here is the caller graph for this function:



5.1.3 Variable Documentation

5.1.3.1 cmd_strings

```
const char* const cmd_strings[(CMD_LAST - CMD_FIRST+2) - 1] [static]
```

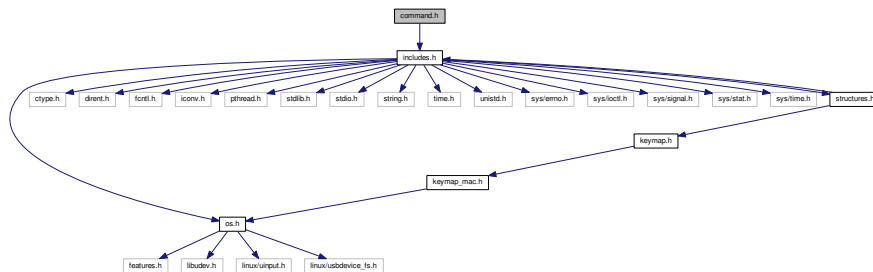
Definition at line 9 of file command.c.

Referenced by readcmd().

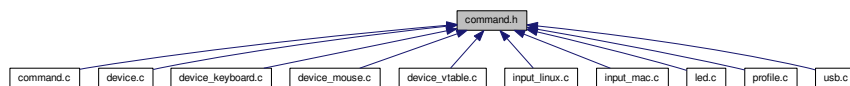
5.2 command.h File Reference

```
#include "includes.h"
```

Include dependency graph for command.h:



This graph shows which files directly or indirectly include this file:



Data Structures

- union [devcmd](#)
- struct [devcmd.__unnamed__](#)

Macros

- `#define` [CMD_COUNT](#) (CMD_LAST - CMD_FIRST + 2)
- `#define` [CMD_DEV_COUNT](#) (CMD_LAST - CMD_VT_FIRST + 1)

Typedefs

- typedef void(* [cmdhandler](#)) ([usbdevice](#) *kb, [usbmode](#) *modeidx, int notifyidx, int keyindex, const char *parameter)
- typedef int(* [cmdhandler_io](#)) ([usbdevice](#) *kb, [usbmode](#) *modeidx, int notifyidx, int keyindex, const char *parameter)
- typedef void(* [cmdhandler_mac](#)) ([usbdevice](#) *kb, [usbmode](#) *modeidx, int notifyidx, const char *keys, const char *assignment)
- typedef union [devcmd](#) [devcmd](#)

Enumerations

```
enum cmd {
    NONE = -11, DELAY = -10, CMD_FIRST = DELAY, MODE = -9,
    SWITCH = -8, LAYOUT = -7, ACCEL = -6, SCROLLSPEED = -5,
    NOTIFYON = -4, NOTIFYOFF = -3, FPS = -2, DITHER = -1,
    HWLOAD = 0, CMD_VT_FIRST = 0, HWSAVE, FWUPDATE,
    POLLRATE, ACTIVE, IDLE, ERASE,
    ERASEPROFILE, NAME, PROFILENAME, ID,
    PROFILEID, RGB, IOFF, ION,
    IAUTO, BIND, UNBIND, REBIND,
    MACRO, DPI, DPISEL, LIFT,
    SNAP, NOTIFY, INOTIFY, GET,
    RESTART, CMD_LAST = RESTART }

```

Functions

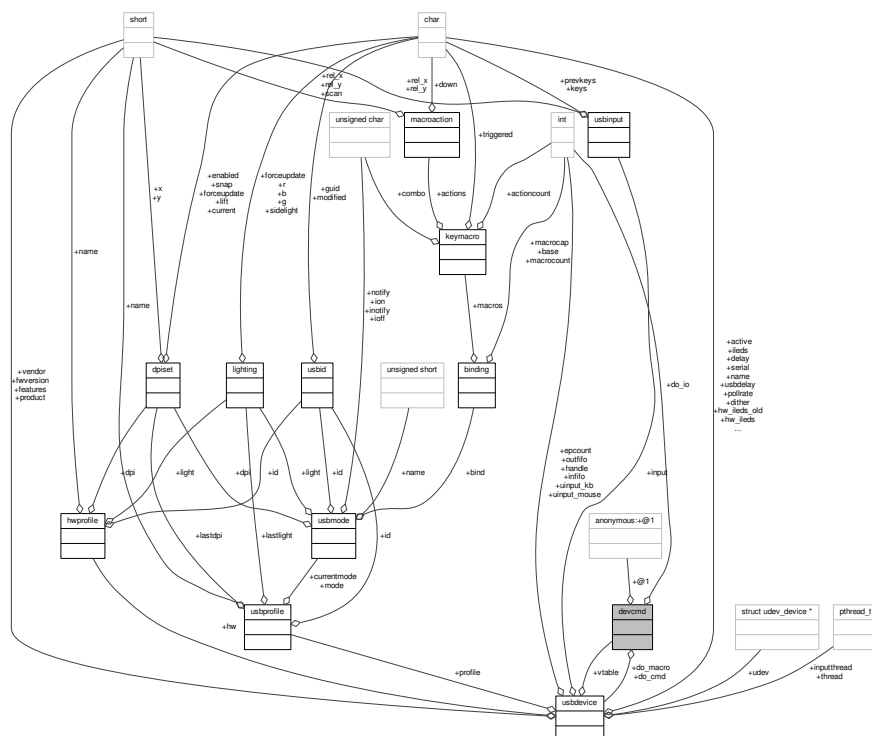
```
int readcmd (usbdevice *kb, const char *line)
```

5.2.1 Data Structure Documentation

5.2.1.1 union devcmd

Definition at line 73 of file command.h.

Collaboration diagram for devcmd:



Data Fields

struct devcmd	__unnamed__	
cmdhandler	do_cmd[(CMD_LAST - CMD_VT_FIRST+1)]	
cmdhandler_io	do_io[(CMD_LAST - CMD_VT_FIRST+1)]	
cmdhandler_mac	do_macro[(CMD_LAST - CMD_VT_FIRST+1)]	

5.2.2 Macro Definition Documentation

5.2.2.1 CMD_COUNT

```
#define CMD_COUNT (CMD_LAST - CMD_FIRST + 2)
```

Definition at line 65 of file [command.h](#).

Referenced by [readcmd\(\)](#).

5.2.2.2 CMD_DEV_COUNT

```
#define CMD_DEV_COUNT (CMD_LAST - CMD_VT_FIRST + 1)
```

Definition at line 66 of file [command.h](#).

5.2.3 Typedef Documentation

5.2.3.1 cmdhandler

```
typedef void(* cmdhandler) (usbdevice *kb, usbmode *modeidx, int notifyidx, int keyindex, const  
char *parameter)
```

Definition at line 70 of file [command.h](#).

5.2.3.2 cmdhandler_io

```
typedef int(* cmdhandler_io) (usbdevice *kb, usbmode *modeidx, int notifyidx, int keyindex,  
const char *parameter)
```

Definition at line 71 of file [command.h](#).

5.2.3.3 cmdhandler_mac

```
typedef void(* cmdhandler_mac) (usbdevice *kb, usbmode *modeidx, int notifyidx, const char *keys, const char *assignment)
```

Definition at line 72 of file command.h.

5.2.3.4 devcmd

```
typedef union devcmd devcmd
```

5.2.4 Enumeration Type Documentation

5.2.4.1 cmd

```
enum cmd
```

Enumerator

NONE	
DELAY	
CMD_FIRST	
MODE	
SWITCH	
LAYOUT	
ACCEL	
SCROLLSPEED	
NOTIFYON	
NOTIFYOFF	
FPS	
DITHER	
HWLOAD	
CMD_VT_FIRST	
HWSAVE	
FWUPDATE	
POLLRATE	
ACTIVE	
IDLE	
ERASE	
ERASEPROFILE	
NAME	
PROFILENAME	
ID	
PROFILEID	
RGB	
IOFF	

Enumerator

ION	
IAUTO	
BIND	
UNBIND	
REBIND	
MACRO	
DPI	
DPISEL	
LIFT	
SNAP	
NOTIFY	
INOTIFY	
GET	
RESTART	
CMD_LAST	

Definition at line 7 of file command.h.

```

7      {
8      // Special - handled by readcmd, no device functions
9      NONE      = -11,
10     DELAY      = -10,    CMD_FIRST = DELAY,
11     MODE       = -9,
12     SWITCH     = -8,
13     LAYOUT     = -7,
14     ACCEL      = -6,
15     SCROLLSPEED = -5,
16     NOTIFYON   = -4,
17     NOTIFYOFF  = -3,
18     FPS        = -2,
19     DITHER     = -1,
20
21     // Hardware data
22     HWLOAD      = 0,    CMD_VT_FIRST = 0,
23     HWSAVE,
24     FWUPDATE,
25     POLLRATE,
26
27     // Software control on/off
28     ACTIVE,
29     IDLE,
30
31     // Profile/mode metadata
32     ERASE,
33     ERASEPROFILE,
34     NAME,
35     PROFILENAME,
36     ID,
37     PROFILEID,
38
39     // LED control
40     RGB,
41     IOFF,
42     ION,
43     IAUTO,
44
45     // Key binding control
46     BIND,
47     UNBIND,
48     REBIND,
49     MACRO,
50
51     // DPI control
52     DPI,
53     DPISEL,
54     LIFT,
55     SNAP,
56
57     // Notifications and output
58     NOTIFY,

```

```

59     INOTIFY,
60     GET,
61     RESTART,
62
63     CMD_LAST = RESTART
64 } cmd;

```

5.2.5 Function Documentation

5.2.5.1 readcmd()

```

int readcmd (
    usbdevice * kb,
    const char * line )

```

Because length of word is length of line + 1, there should be no problem with buffer overflow.

Definition at line 67 of file command.c.

References ACCEL, ACTIVE, usbdevice::active, BIND, CMD_COUNT, CMD_FIRST, cmd_strings, usbprofile::currentmode, DELAY, usbdevice::delay, DITHER, usbdevice::dither, devcmd::do_cmd, devcmd::do_io, devcmd::do_macro, DPI, DPISEL, ERASE, ERASEPROFILE, FEAT_ANSI, FEAT_BIND, FEAT_ISO, FEAT_LMASK, FEAT_MOUSEACCEL, FEAT_NOTIFY, usbdevice::features, lighting::forceupdate, FPS, FWUPDATE, GET, HAS_FEATURES, HWLOAD, HWSAVE, IAUTO, ID, IDLE, INDEX_OF, INOTIFY, IOFF, ION, IS_FULLRANGE, IS_MOUSE_DEV, keymap, LAYOUT, LIFT, usbmode::light, MACRO, mknotifynode(), MODE, usbprofile::mode, MODE_COUNT, N_KEYS_EXTENDED, NAME, NEEDS_FW_UPDATE, NONE, NOTIFY, NOTIFYOFF, NOTIFYON, OUTFIFO_MAX, POLLRATE, usbdevice::profile, PROFILEID, PROFILENAME, REBIND, RESTART, RGB, rmnotifynode(), SCROLL_ACCELERATED, SCROLL_MAX, SCROLL_MIN, SCROLLSPEED, SNAP, SWITCH, TRY_WITH_RESET, UNBIND, usbdevice::usbdelay, and usbdevice::vtable.

Referenced by devmain().

```

67     {
68         char* word = malloc(strlen(line) + 1);
69         int wordlen;
70         const char* newline = 0;
71         const devcmd* vt = kb->vtable;
72         usbprofile* profile = kb->profile;
73         usbmode* mode = 0;
74         int notifynumber = 0;
75         // Read words from the input
76         cmd command = NONE;
77         while(sscanf(line, "%s%n", word, &wordlen) == 1){
78             line += wordlen;
79             // If we passed a newline, reset the context
80             if(line > newline){
81                 mode = profile->currentmode;
82                 command = NONE;
83                 notifynumber = 0;
84                 newline = strchr(line, '\n');
85                 if(!newline)
86                     newline = line + strlen(line);
87             }
88             // Check for a command word
89             for(int i = 0; i < CMD_COUNT - 1; i++){
90                 if(!strcmp(word, cmd_strings[i])){
91                     command = i + CMD_FIRST;
92 #ifndef OS_MAC
93                     // Layout and mouse acceleration aren't used on Linux; ignore
94                     if(command == LAYOUT || command == ACCEL || command ==
SCROLLSPEED)
95                         command = NONE;
96 #endif

```

```

97         // Most commands require parameters, but a few are actions in and of themselves
98         if(command != SWITCH
99             && command != HWLOAD && command != HWSAVE
100             && command != ACTIVE && command != IDLE
101             && command != ERASE && command != ERASEPROFILE
102             && command != RESTART)
103             goto next_loop;
104         break;
105     }
106 }
107
108 // Set current notification node when given @number
109 int newnotify;
110 if(sscanf(word, "%u", &newnotify) == 1 && newnotify < OUTFIFO_MAX){
111     notifynumber = newnotify;
112     continue;
113 }
114
115 // Reject unrecognized commands. Reject bind or notify related commands if the keyboard doesn't
116 // have the feature enabled.
117 if(command == NONE
118     || (!HAS_FEATURES(kb, FEAT_BIND) && (command ==
119     BIND || command == UNBIND || command == REBIND || command ==
120     MACRO || command == DELAY))
121     || (!HAS_FEATURES(kb, FEAT_NOTIFY) && command ==
122     NOTIFY)){
123     next_loop:
124     continue;
125 }
126 // Reject anything not related to fwupdate if device has a bricked FW
127 if(NEEDS_FW_UPDATE(kb) && command != FWUPDATE && command !=
128 NOTIFYON && command != NOTIFYOFF)
129     continue;
130
131 // Specially handled commands - these are available even when keyboard is IDLE
132 switch(command){
133 case NOTIFYON: {
134     // Notification node on
135     int notify;
136     if(sscanf(word, "%u", &notify) == 1)
137         mknotifynode(kb, notify);
138     continue;
139 } case NOTIFYOFF: {
140     // Notification node off
141     int notify;
142     if(sscanf(word, "%u", &notify) == 1 && notify != 0) // notify0 can't be removed
143         rmnotifynode(kb, notify);
144     continue;
145 } case GET:
146     // Output data to notification node
147     vt->get(kb, mode, notifynumber, 0, word);
148     continue;
149 case LAYOUT:
150     // OSX: switch ANSI/ISO keyboard layout
151     if(!strcmp(word, "ansi"))
152         kb->features = (kb->features & ~FEAT_LMASK) |
153         FEAT_ANSI;
154     else if(!strcmp(word, "iso"))
155         kb->features = (kb->features & ~FEAT_LMASK) |
156         FEAT_ISO;
157     continue;
158 #ifdef OS_MAC
159 case ACCEL:
160     // OSX mouse acceleration on/off
161     if(!strcmp(word, "on"))
162         kb->features |= FEAT_MOUSEACCEL;
163     else if(!strcmp(word, "off"))
164         kb->features &= ~FEAT_MOUSEACCEL;
165     continue;
166 case SCROLLSPEED:
167     int newscroll;
168     if(sscanf(word, "%d", &newscroll) != 1)
169         break;
170     if(newscroll < SCROLL_MIN)
171         newscroll = SCROLL_ACCELERATED;
172     if(newscroll > SCROLL_MAX)
173         newscroll = SCROLL_MAX;
174     kb->scroll_rate = newscroll;
175     continue;
176 }
177 #endif
178 case MODE: {
179     // Select a mode number (1 - 6)
180     int newmode;
181     if(sscanf(word, "%u", &newmode) == 1 && newmode > 0 && newmode <=
182     MODE_COUNT)
183         mode = profile->mode + newmode - 1;
184 }

```

```

176         continue;
177     }
178     case FPS: {
179         // USB command delay (2 - 10ms)
180         uint framerate;
181         if(sscanf(word, "%u", &framerate) == 1 && framerate > 0){
182             // Not all devices require the same number of messages per frame; select delay
183             appropriately
184             uint per_frame = IS_MOUSE_DEV(kb) ? 2 : IS_FULLRANGE(kb) ? 14 : 5;
185             uint delay = 1000 / framerate / per_frame;
186             if(delay < 2)
187                 delay = 2;
188             else if(delay > 10)
189                 delay = 10;
190             kb->usbdelay = delay;
191         }
192         continue;
193     }
194     case DITHER: {
195         // 0: No dither, 1: Ordered dither.
196         uint dither;
197         if(sscanf(word, "%u", &dither) == 1 && dither <= 1){
198             kb->dither = dither;
199             profile->currentmode->light.forceupdate = 1;
200             mode->light.forceupdate = 1;
201         }
202         continue;
203     }
204     case DELAY:
205         kb->delay = (!strcmp(word, "on")); // independendant from parameter to handle false
206         commands like "delay off"
207         continue;
208     case RESTART: {
209         char mybuffer[] = "no reason specified";
210         if (sscanf(line, "%[^\n]", word) == -1) {
211             word = mybuffer;
212         }
213         vt->do_cmd[command](kb, mode, notifiynumber, 0, word);
214         continue;
215     }
216     default;;
217 }
218 // If a keyboard is inactive, it must be activated before receiving any other commands
219 if(!kb->active){
220     if(command == ACTIVE)
221         TRY_WITH_RESET(vt->active(kb, mode, notifiynumber, 0, 0));
222     continue;
223 }
224 // Specially handled commands only available when keyboard is ACTIVE
225 switch(command){
226     case IDLE:
227         TRY_WITH_RESET(vt->idle(kb, mode, notifiynumber, 0, 0));
228         continue;
229     case SWITCH:
230         if(profile->currentmode != mode){
231             profile->currentmode = mode;
232             // Set mode light for non-RGB K95
233             int index = INDEX_OF(mode, profile->mode);
234             vt->setmodeindex(kb, index);
235         }
236         continue;
237     case HWLOAD: case HWSAVE:{
238         char delay = kb->usbdelay;
239         // Ensure delay of at least 10ms as the device can get overwhelmed otherwise
240         if(delay < 10)
241             kb->usbdelay = 10;
242         // Try to load/save the hardware profile. Reset on failure, disconnect if reset fails.
243         TRY_WITH_RESET(vt->do_io[command](kb, mode, notifiynumber, 1, 0));
244         // Re-send the current RGB state as it sometimes gets scrambled
245         TRY_WITH_RESET(vt->updatergb(kb, 1));
246         kb->usbdelay = delay;
247         continue;
248     }
249     case FWUPDATE:
250         // FW update parses a whole word. Unlike hwload/hwsave, there's no try again on failure.
251         if(vt->fwupdate(kb, mode, notifiynumber, 0, word)){
252             free(word);
253             return 1;
254         }
255         continue;
256     case POLLRATE: {
257         uint rate;
258         if(sscanf(word, "%u", &rate) == 1 && (rate == 1 || rate == 2 || rate == 4 || rate == 8))
259             TRY_WITH_RESET(vt->pollrate(kb, mode, notifiynumber, rate, 0));
260         continue;

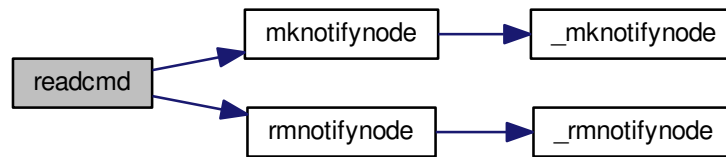
```

```

261     }
262     case ERASEPROFILE:
263         // Erase the current profile
264         vt->eraseprofile(kb, mode, notifynumber, 0, 0);
265         // Update profile/mode pointers
266         profile = kb->profile;
267         mode = profile->currentmode;
268         continue;
269     case ERASE: case NAME: case IOFF: case ION: case IAUTO: case
INOTIFY: case PROFILENAME: case ID: case PROFILEID: case
DPISEL: case LIFT: case SNAP:
270         // All of the above just parse the whole word
271         vt->do_cmd[command](kb, mode, notifynumber, 0, word);
272         continue;
273     case RGB: {
274         // RGB command has a special response for a single hex constant
275         int r, g, b;
276         if(sscanf(word, "%02x%02x%02x", &r, &g, &b) == 3){
277             // Set all keys
278             for(int i = 0; i < N_KEYS_EXTENDED; i++)
279                 vt->rgb(kb, mode, notifynumber, i, word);
280             continue;
281         }
282         break;
283     }
284     case MACRO:
285         if(!strcmp(word, "clear")){
286             // Macro has a special clear command
287             vt->macro(kb, mode, notifynumber, 0, 0);
288             continue;
289         }
290         break;
291     default:;
292     }
293     // For anything else, split the parameter at the colon
294     int left = -1;
295     sscanf(word, "%*[^:]%n", &left);
296     if(left <= 0)
297         continue;
298     const char* right = word + left;
299     if(right[0] == ':')
300         right++;
301     // Macros and DPI have a separate left-side handler
302     if(command == MACRO || command == DPI){
303         word[left] = 0;
304         vt->do_macro[command](kb, mode, notifynumber, word, right);
305         continue;
306     }
307     // Scan the left side for key names and run the requested command
308     int position = 0, field = 0;
309     char keyname[11];
310     while(position < left && sscanf(word + position, "%10[^:,%n", keyname, &field) == 1){
311         int keycode;
312         if(!strcmp(keyname, "all")){
313             // Set all keys
314             for(int i = 0; i < N_KEYS_EXTENDED; i++)
315                 vt->do_cmd[command](kb, mode, notifynumber, i, right);
316         } else if((sscanf(keyname, "%d", &keycode) && keycode >= 0 && keycode <
N_KEYS_EXTENDED)
317             || (sscanf(keyname, "%x", &keycode) && keycode >= 0 && keycode <
N_KEYS_EXTENDED)){
318             // Set a key numerically
319             vt->do_cmd[command](kb, mode, notifynumber, keycode, right);
320         } else {
321             // Find this key in the keymap
322             for(unsigned i = 0; i < N_KEYS_EXTENDED; i++){
323                 if(keymap[i].name && !strcmp(keyname, keymap[i].name)){
324                     vt->do_cmd[command](kb, mode, notifynumber, i, right);
325                     break;
326                 }
327             }
328         }
329         if(word[position += field] == ',')
330             position++;
331     }
332 }
333
334 // Finish up
335 if(!NEEDS_FW_UPDATE(kb)){
336     TRY_WITH_RESET(vt->updatergb(kb, 0));
337     TRY_WITH_RESET(vt->updatedpi(kb, 0));
338 }
339 free(word);
340 return 0;
341 }

```

Here is the call graph for this function:



Here is the caller graph for this function:



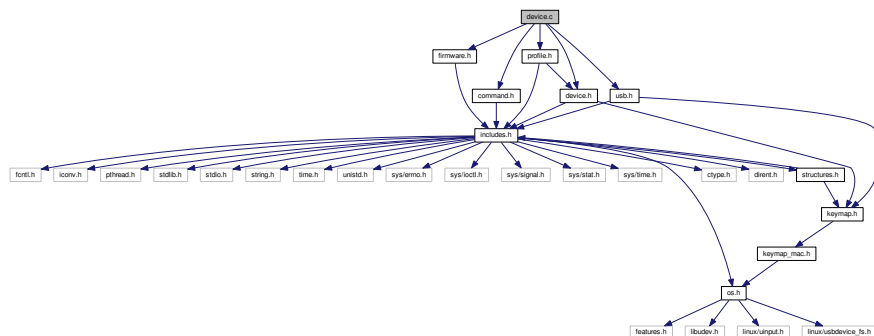
5.3 device.c File Reference

```

#include "command.h"
#include "device.h"
#include "firmware.h"
#include "profile.h"
#include "usb.h"

```

Include dependency graph for device.c:



Functions

- `int _start_dev (usbdevice *kb, int makeactive)`
- `int start_dev (usbdevice *kb, int makeactive)`

Variables

- int `hwload_mode` = 1
- usbdevice `keyboard` [9]
hwload_mode = 1 means read hardware once. should be enough
- pthread_mutex_t `devlistmutex` = PTHREAD_MUTEX_INITIALIZER
remember all usb devices. Needed for `closeusb()`.
- pthread_mutex_t `devmutex` [9] = { [0 ... 9 -1] = PTHREAD_MUTEX_INITIALIZER }
- pthread_mutex_t `inputmutex` [9] = { [0 ... 9 -1] = PTHREAD_MUTEX_INITIALIZER }

5.3.1 Function Documentation

5.3.1.1 `_start_dev()`

```
int _start_dev (
    usbdevice * kb,
    int makeactive )
```

`_start_dev` get fw-info and pollrate; if available, install new firmware; get all hardware profiles

Parameters

<code>kb</code>	the normal kb pointer to the usbdevice. Is also valid for mice.
<code>makeactive</code>	if set to 1, activate the device via <code>setactive()</code>

Returns

0 if success, other else

- This hacker code is tricky in mutiple aspects. What it means is:
 if `hwload_mode == 0`: just set pollrate to 0 and clear features in the bottom lines of the if-block.
 if `hwload_mode == 1`: if the device has FEAT_HWLOAD active, call `getfwversion()`. If it returns true, there was an error while detecting fw-version. Put error message, reset FEAT_HWLOAD and finalize as above.
 if `hwload_mode == 2`: if the device has FEAT_HWLOAD active, call `getfwversion()`. If it returns true, there was an error while detecting fw-version. Put error message and return directly from function with error.
 Why do not you just write it down?
- Now check if device needs a firmware update. If so, set it up and leave the function without error.
- Device needs a firmware update. Finish setting up but don't do anything.
- Load profile from device if the hw-pointer is not set yet and hw-loading is possible and allowed.
 return error if mode == 2 (load always) and loading got an error. Else reset HWLOAD feature, because hwload must be 1.

That is real Horror code.

Definition at line 22 of file device.c.

References `usbdevice::active`, `ckb_info`, `ckb_warn`, `FEAT_ADJRATE`, `FEAT_FWUPDATE`, `FEAT_FWVERSION`, `FEAT_HWLOAD`, `FEAT_POLLRATE`, `FEAT_RGB`, `usbdevice::features`, `usbdevice::fwversion`, `getfwversion()`, `HAS_FEATURES`, `usbdevice::hw`, `hwload_mode`, `hwloadprofile`, `NEEDS_FW_UPDATE`, `usbdevice::pollrate`, and `setactive`.

Referenced by `start_dev()`.

```

22                                     {
23     // Get the firmware version from the device
24     if(kb->pollrate == 0){
32         if(!hwload_mode || (HAS_FEATURES(kb, FEAT_HWLOAD) &&
getfwversion(kb))){
33             if(hwload_mode == 2)
34                 // hwload=always. Report setup failure.
35                 return -1;
36             else if(hwload_mode){
37                 // hwload=once. Log failure, prevent trying again, and continue.
38                 ckb_warn("Unable to load firmware version/poll rate\n");
39                 kb->features &= ~FEAT_HWLOAD;
40             }
41             kb->pollrate = 0;
42             kb->features &= ~(FEAT_POLLRATE | FEAT_ADJRATE);
43             if(kb->fwversion == 0)
44                 kb->features &= ~(FEAT_FWVERSION |
FEAT_FWUPDATE);
45         }
46     }
51     if(NEEDS_FW_UPDATE(kb)){
53         ckb_info("Device needs a firmware update. Please issue a fwupdate command.\n");
54         kb->features = FEAT_RGB | FEAT_FWVERSION |
FEAT_FWUPDATE;
55         kb->active = 1;
56         return 0;
57     }
63     if(!kb->hw && hwload_mode && HAS_FEATURES(kb,
FEAT_HWLOAD)){
64         if(hwloadprofile(kb, 1)){
65             if(hwload_mode == 2)
66                 return -1;
67             ckb_warn("Unable to load hardware profile\n");
68             kb->features &= ~FEAT_HWLOAD;
69         }
70     }
71     // Active software mode if requested
72     if(makeactive)
73         return setactive(kb, 1);
74     return 0;
75 }

```

Here is the call graph for this function:



Here is the caller graph for this function:



5.3.1.2 start_dev()

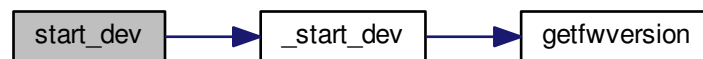
```
int start_dev (
    usbdevice * kb,
    int makeactive )
```

Definition at line 77 of file device.c.

References `_start_dev()`, `USB_DELAY_DEFAULT`, and `usbdevice::usbdelay`.

```
77 {
78     // Force USB interval to 10ms during initial setup phase; return to nominal 5ms after setup completes.
79     kb->usbdelay = 10;
80     int res = _start_dev(kb, makeactive);
81     kb->usbdelay = USB_DELAY_DEFAULT;
82     return res;
83 }
```

Here is the call graph for this function:



5.3.2 Variable Documentation

5.3.2.1 devlistmutex

```
pthread_mutex_t devlistmutex = PTHREAD_MUTEX_INITIALIZER
```

Definition at line 11 of file device.c.

5.3.2.2 devmutex

```
pthread_mutex_t devmutex[9] = { [0 ... 9 -1] = PTHREAD_MUTEX_INITIALIZER }
```

Definition at line 12 of file device.c.

Referenced by `_updateconnected()`, `quitWithLock()`, and `usb_rm_device()`.

5.3.2.3 hwload_mode

```
int hwload_mode = 1
```

Definition at line 7 of file device.c.

Referenced by `_start_dev()`, `_usbrecv()`, `_usbsend()`, `main()`, and `usb_tryreset()`.

5.3.2.4 inputmutex

```
pthread_mutex_t inputmutex[9] = { [0 ... 9 -1] = PTHREAD_MUTEX_INITIALIZER }
```

Definition at line 13 of file device.c.

5.3.2.5 keyboard

```
usbdevice keyboard[9]
```

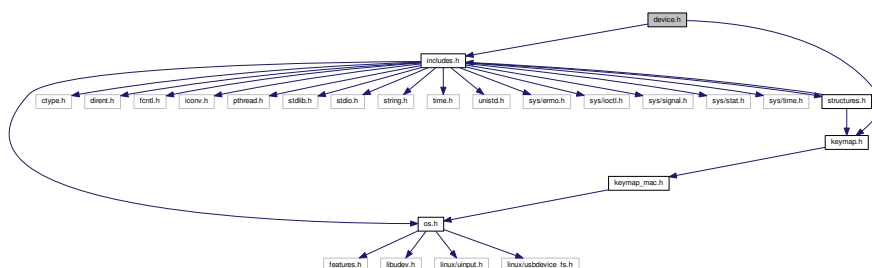
Definition at line 10 of file device.c.

Referenced by `_mkdevpath()`, `_mknotifynode()`, `_rmnotifynode()`, `_setupusb()`, `_updateconnected()`, `closeusb()`, `main()`, `mkfwnode()`, `os_closeusb()`, `os_inputmain()`, `os_inputopen()`, `os_setupusb()`, `quitWithLock()`, `rmdevpath()`, `usb_rm_device()`, and `usbadd()`.

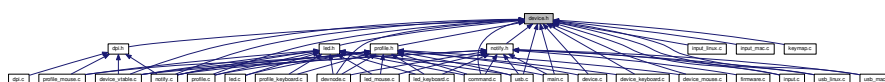
5.4 device.h File Reference

```
#include "includes.h"
#include "keymap.h"
```

Include dependency graph for device.h:



This graph shows which files directly or indirectly include this file:



Macros

- `#define DEV_MAX 9`
- `#define IS_CONNECTED(kb) ((kb) && (kb)->handle && (kb)->uinput_kb && (kb)->uinput_mouse)`
- `#define dmutex(kb) (devmutex + INDEX_OF(kb, keyboard))`
- `#define imutex(kb) (inputmutex + INDEX_OF(kb, keyboard))`
- `#define setactive(kb, makeactive) ((makeactive) ? (kb)->vtable->active((kb), 0, 0, 0, 0) : (kb)->vtable->idle((kb), 0, 0, 0, 0))`
*setactive() calls via the corresponding kb->vtable either the active() or the idle() function.
active() is called if the parameter makeactive is true, idle if it is false.
What function is called effectively is device dependent. Have a look at [device_vtable.c](#) for more information.*
- `#define IN_HID 0x80`
- `#define IN_CORSAIR 0x40`
- `#define ACT_LIGHT 1`
- `#define ACT_NEXT 3`
- `#define ACT_NEXT_NOWRAP 5`
- `#define ACT_LOCK 8`
- `#define ACT_MR_RING 9`
- `#define ACT_M1 10`
- `#define ACT_M2 11`
- `#define ACT_M3 12`

Functions

- `int start_dev (usbdevice *kb, int makeactive)`
- `int start_kb_nrgb (usbdevice *kb, int makeactive)`
- `int setactive_kb (usbdevice *kb, int active)`
- `int setactive_mouse (usbdevice *kb, int active)`
- `int cmd_active_kb (usbdevice *kb, usbmode *dummy1, int dummy2, int dummy3, const char *dummy4)`
- `int cmd_active_mouse (usbdevice *kb, usbmode *dummy1, int dummy2, int dummy3, const char *dummy4)`
- `int cmd_idle_kb (usbdevice *kb, usbmode *dummy1, int dummy2, int dummy3, const char *dummy4)`
- `int cmd_idle_mouse (usbdevice *kb, usbmode *dummy1, int dummy2, int dummy3, const char *dummy4)`
- `int cmd_pollrate (usbdevice *kb, usbmode *dummy1, int dummy2, int rate, const char *dummy3)`
- `void setmodeindex_nrgb (usbdevice *kb, int index)`

Variables

- `usbdevice keyboard [9]`
hwload_mode = 1 means read hardware once. should be enough
- `pthread_mutex_t devmutex [9]`
- `pthread_mutex_t inputmutex [9]`

5.4.1 Macro Definition Documentation

5.4.1.1 ACT_LIGHT

```
#define ACT_LIGHT 1
```

Definition at line 60 of file device.h.

Referenced by `setactive_kb()`.

5.4.1.2 ACT_LOCK

```
#define ACT_LOCK 8
```

Definition at line 63 of file device.h.

Referenced by `setactive_kb()`.

5.4.1.3 ACT_M1

```
#define ACT_M1 10
```

Definition at line 65 of file device.h.

Referenced by `setactive_kb()`.

5.4.1.4 ACT_M2

```
#define ACT_M2 11
```

Definition at line 66 of file device.h.

Referenced by `setactive_kb()`.

5.4.1.5 ACT_M3

```
#define ACT_M3 12
```

Definition at line 67 of file device.h.

Referenced by `setactive_kb()`.

5.4.1.6 ACT_MR_RING

```
#define ACT_MR_RING 9
```

Definition at line 64 of file device.h.

Referenced by `setactive_kb()`.

5.4.1.7 ACT_NEXT

```
#define ACT_NEXT 3
```

Definition at line 61 of file device.h.

5.4.1.8 ACT_NEXT_NOWRAP

```
#define ACT_NEXT_NOWRAP 5
```

Definition at line 62 of file device.h.

5.4.1.9 DEV_MAX

```
#define DEV_MAX 9
```

Definition at line 8 of file device.h.

Referenced by `_updateconnected()`, `quitWithLock()`, `usb_rm_device()`, and `usbadd()`.

5.4.1.10 dmutex

```
#define dmutex(  
    kb ) (devmutex + INDEX_OF(kb, keyboard))
```

Definition at line 18 of file device.h.

Referenced by `_ledthread()`, `_setupusb()`, `closeusb()`, `devmain()`, and `usbadd()`.

5.4.1.11 imutex

```
#define imutex(  
    kb ) (inputmutex + INDEX_OF(kb, keyboard))
```

Definition at line 22 of file device.h.

Referenced by `_setupusb()`, `closeusb()`, `cmd_bind()`, `cmd_erase()`, `cmd_eraseprofile()`, `cmd_get()`, `cmd_macro()`, `cmd_notify()`, `cmd_rebind()`, `cmd_unbind()`, `os_inputmain()`, `setactive_kb()`, `setactive_mouse()`, and `setupusb()`.

5.4.1.12 IN_CORSAIR

```
#define IN_CORSAIR 0x40
```

Definition at line 57 of file device.h.

Referenced by `setactive_kb()`, and `setactive_mouse()`.

5.4.1.13 IN_HID

```
#define IN_HID 0x80
```

Definition at line 56 of file device.h.

Referenced by `setactive_kb()`, and `setactive_mouse()`.

5.4.1.14 IS_CONNECTED

```
#define IS_CONNECTED(  
    kb ) ((kb) && (kb)->handle && (kb)->uinput_kb && (kb)->uinput_mouse)
```

Definition at line 12 of file device.h.

Referenced by `_updateconnected()`, `devmain()`, `quitWithLock()`, and `usbadd()`.

5.4.1.15 setactive

```
#define setactive(  
    kb,  
    makeactive ) ((makeactive) ? (kb)->vtable->active((kb), 0, 0, 0, 0) : (kb)->vtable->idle((kb),  
0, 0, 0, 0))
```

Definition at line 36 of file device.h.

Referenced by `_start_dev()`, and `revertusb()`.

5.4.2 Function Documentation

5.4.2.1 cmd_active_kb()

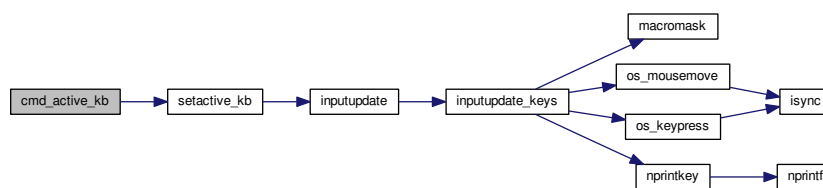
```
int cmd_active_kb (
    usbdevice * kb,
    usbmode * dummy1,
    int dummy2,
    int dummy3,
    const char * dummy4 )
```

Definition at line 112 of file device_keyboard.c.

References [setactive_kb\(\)](#).

```
112
113     return setactive_kb(kb, 1);
114 }
```

Here is the call graph for this function:



5.4.2.2 cmd_active_mouse()

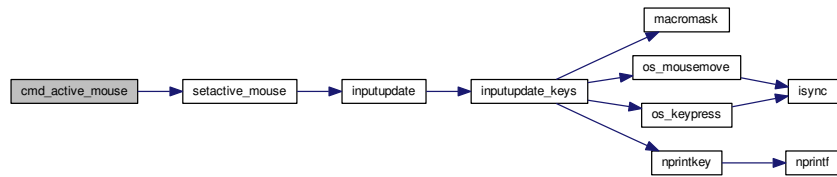
```
int cmd_active_mouse (
    usbdevice * kb,
    usbmode * dummy1,
    int dummy2,
    int dummy3,
    const char * dummy4 )
```

Definition at line 44 of file device_mouse.c.

References [setactive_mouse\(\)](#).

```
44
45     return setactive_mouse(kb, 1);
46 }
```

Here is the call graph for this function:



5.4.2.3 cmd_idle_kb()

```

int cmd_idle_kb (
    usbdevice * kb,
    usbmode * dummy1,
    int dummy2,
    int dummy3,
    const char * dummy4 )

```

Definition at line 116 of file device_keyboard.c.

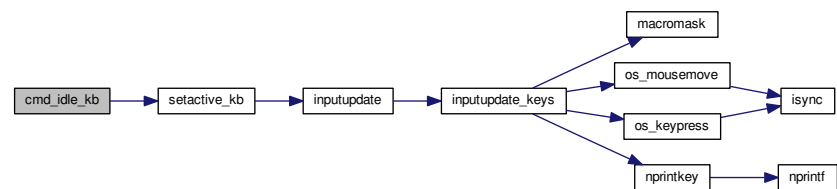
References setactive_kb().

```

116
117     return setactive_kb(kb, 0);
118 }

```

Here is the call graph for this function:



5.4.2.4 cmd_idle_mouse()

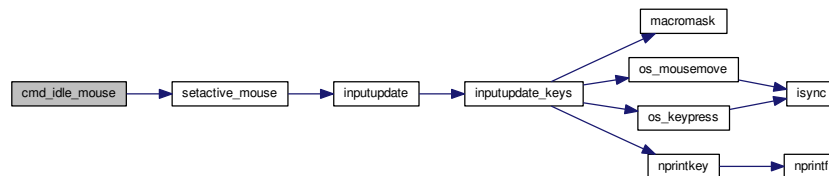
```
int cmd_idle_mouse (
    usbdevice * kb,
    usbmode * dummy1,
    int dummy2,
    int dummy3,
    const char * dummy4 )
```

Definition at line 48 of file device_mouse.c.

References [setactive_mouse\(\)](#).

```
48
49     return setactive_mouse(kb, 0);
50 }
```

Here is the call graph for this function:



5.4.2.5 cmd_pollrate()

```
int cmd_pollrate (
    usbdevice * kb,
    usbmode * dummy1,
    int dummy2,
    int rate,
    const char * dummy3 )
```

Definition at line 52 of file device_mouse.c.

References [MSG_SIZE](#), [usbdevice::pollrate](#), and [usbdevice::send](#).

```
52
53     uchar msg[MSG_SIZE] = {
54         0x07, 0x0a, 0, 0, (uchar)rate
55     };
56     if(!usbdevice::send(kb, msg, 1))
57         return -1;
58     // Device should disconnect+reconnect, but update the poll rate field in case it doesn't
59     kb->pollrate = rate;
60     return 0;
61 }
```

5.4.2.6 setactive_kb()

```
int setactive_kb (
    usbdevice * kb,
    int active )
```

Definition at line 18 of file device_keyboard.c.

References ACT_LIGHT, ACT_LOCK, ACT_M1, ACT_M2, ACT_M3, ACT_MR_RING, usbdevice::active, DELAY_MEDIUM, lighting::forceupdate, imutex, IN_CORSAIR, IN_HID, usbdevice::input, inputupdate(), keymap, usbinput::keys, usbprofile::lastlight, MSG_SIZE, N_KEYS_HW, NEEDS_FW_UPDATE, usbdevice::profile, usb_send, and usbdevice::vtable.

Referenced by cmd_active_kb(), and cmd_idle_kb().

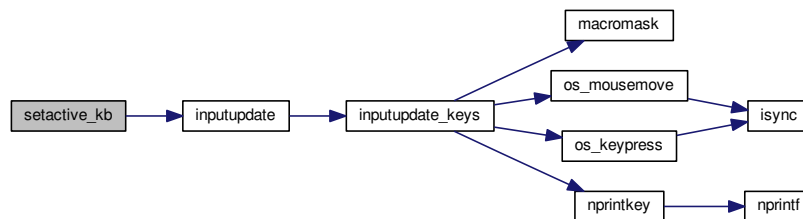
```
18                                     {
19     if (NEEDS_FW_UPDATE(kb))
20         return 0;
21
22     pthread_mutex_lock(&imutex(kb));
23     kb->active = !active;
24     kb->profile->lastlight.forceupdate = 1;
25     // Clear input
26     memset(&kb->input.keys, 0, sizeof(kb->input.keys));
27     inputupdate(kb);
28     pthread_mutex_unlock(&imutex(kb));
29
30     uchar msg[3][MSG_SIZE] = {
31         { 0x07, 0x04, 0 },           // Disables or enables HW control for top row
32         { 0x07, 0x40, 0 },           // Selects key input
33         { 0x07, 0x05, 2, 0, 0x03, 0x00 } // Commits key input selection
34     };
35     if(active){
36         // Put the M-keys (K95) as well as the Brightness/Lock keys into software-controlled mode.
37         msg[0][2] = 2;
38         if(!usb_send(kb, msg[0], 1))
39             return -1;
40         DELAY_MEDIUM(kb);
41         // Set input mode on the keys. They must be grouped into packets of 60 bytes (+ 4 bytes header)
42         // Keys are referenced in byte pairs, with the first byte representing the key and the second byte
43         // representing the mode.
44         for(int key = 0; key < N_KEYS_HW; ){
45             int pair;
46             for(pair = 0; pair < 30 && key < N_KEYS_HW; pair++, key++){
47                 // Select both standard and Corsair input. The standard input will be ignored except in
48                 BIOS mode.
49                 uchar action = IN_HID | IN_CORSAIR;
50                 // Additionally, make MR activate the MR ring (this is disabled for now, may be back later)
51                 //if(keymap[key].name && !strcmp(keymap[key].name, "mr"))
52                 //    action |= ACT_MR_RING;
53                 msg[1][4 + pair * 2] = key;
54                 msg[1][5 + pair * 2] = action;
55             }
56             // Byte 2 = pair count (usually 30, less on final message)
57             msg[1][2] = pair;
58             if(!usb_send(kb, msg[1], 1))
59                 return -1;
60         }
61         // Commit new input settings
62         if(!usb_send(kb, msg[2], 1))
63             return -1;
64         DELAY_MEDIUM(kb);
65     } else {
66         // Set the M-keys back into hardware mode, restore hardware RGB profile. It has to be sent twice
67         for some reason.
68         msg[0][2] = 1;
69         if(!usb_send(kb, msg[0], 1))
70             return -1;
71         DELAY_MEDIUM(kb);
72         if(!usb_send(kb, msg[0], 1))
73             return -1;
74         DELAY_MEDIUM(kb);
75
76 #ifdef OS_LINUX
77     // On OSX the default key mappings are fine. On Linux, the G keys will freeze the keyboard. Set the
78     keyboard entirely to HID input.
79     for(int key = 0; key < N_KEYS_HW; ){
80         int pair;
81         for(pair = 0; pair < 30 && key < N_KEYS_HW; pair++, key++){
```

```

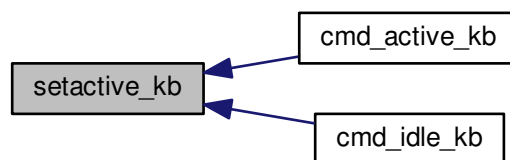
77         uchar action = IN_HID;
78         // Enable hardware actions
79         if(keymap[key].name){
80             if(!strcmp(keymap[key].name, "mr"))
81                 action = ACT_MR_RING;
82             else if(!strcmp(keymap[key].name, "m1"))
83                 action = ACT_M1;
84             else if(!strcmp(keymap[key].name, "m2"))
85                 action = ACT_M2;
86             else if(!strcmp(keymap[key].name, "m3"))
87                 action = ACT_M3;
88             else if(!strcmp(keymap[key].name, "light"))
89                 action = ACT_LIGHT;
90             else if(!strcmp(keymap[key].name, "lock"))
91                 action = ACT_LOCK;
92         }
93         msg[1][4 + pair * 2] = key;
94         msg[1][5 + pair * 2] = action;
95     }
96     // Byte 2 = pair count (usually 30, less on final message)
97     msg[1][2] = pair;
98     if(!usbSend(kb, msg[1], 1))
99         return -1;
100 }
101 // Commit new input settings
102 if(!usbSend(kb, msg[2], 1))
103     return -1;
104 DELAY_MEDIUM(kb);
105 #endif
106 }
107 // Update indicator LEDs if the profile contains settings for them
108 kb->vtable->updateindicators(kb, 0);
109 return 0;
110 }

```

Here is the call graph for this function:



Here is the caller graph for this function:



5.4.2.7 setactive_mouse()

```
int setactive_mouse (
    usbdevice * kb,
    int active )
```

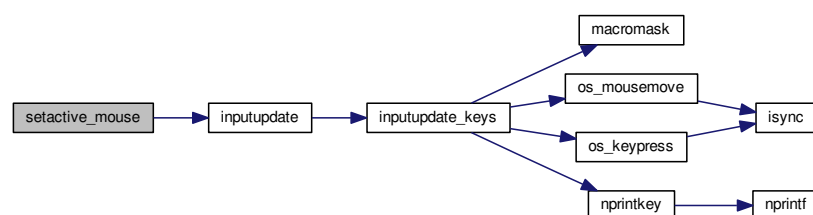
Definition at line 9 of file device_mouse.c.

References `usbdevice::active`, `lighting::forceupdate`, `imutex`, `IN_CORSAIR`, `IN_HID`, `usbdevice::input`, `inputupdate()`, `usbinput::keys`, `usbprofile::lastlight`, `MSG_SIZE`, `NEEDS_FW_UPDATE`, `usbdevice::profile`, and `usbsend`.

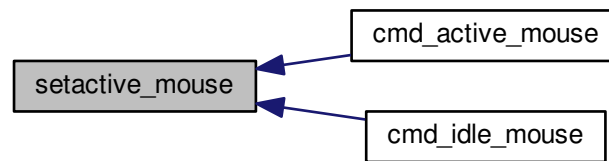
Referenced by `cmd_active_mouse()`, and `cmd_idle_mouse()`.

```
9                                     {
10     if(NEEDS_FW_UPDATE(kb))
11         return 0;
12     const int keycount = 20;
13     uchar msg[2][MSG_SIZE] = {
14         { 0x07, 0x04, 0 },           // Disables or enables HW control for DPI and Sniper button
15         { 0x07, 0x40, keycount, 0 }, // Select button input (similar to the packet sent to
16         keyboards, but lacks a commit packet)
17     };
18     if(active)
19         // Put the mouse into SW mode
20         msg[0][2] = 2;
21     else
22         // Restore HW mode
23         msg[0][2] = 1;
24     pthread_mutex_lock(imutex(kb));
25     kb->active = !active;
26     kb->profile->lastlight.forceupdate = 1;
27     // Clear input
28     memset(&kb->input.keys, 0, sizeof(kb->input.keys));
29     inputupdate(kb);
30     pthread_mutex_unlock(imutex(kb));
31     if(!usbsend(kb, msg[0], 1))
32         return -1;
33     if(active){
34         // Set up key input
35         if(!usbsend(kb, msg[1], 1))
36             return -1;
37         for(int i = 0; i < keycount; i++){
38             msg[1][i * 2 + 4] = i + 1;
39             msg[1][i * 2 + 5] = (i < 6 ? IN_HID : IN_CORSAIR);
40         }
41     }
42     return 0;
43 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



5.4.2.8 setmodeindex_nrgb()

```
void setmodeindex_nrgb (
    usbdevice * kb,
    int index )
```

Definition at line 120 of file `device_keyboard.c`.

References `NK95_M1`, `NK95_M2`, `NK95_M3`, and `nk95cmd`.

```
120                                     {
121     switch(index % 3){
122     case 0:
123         nk95cmd(kb, NK95_M1);
124         break;
125     case 1:
126         nk95cmd(kb, NK95_M2);
127         break;
128     case 2:
129         nk95cmd(kb, NK95_M3);
130         break;
131     }
132 }
```

5.4.2.9 start_dev()

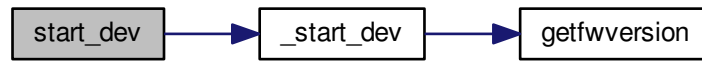
```
int start_dev (
    usbdevice * kb,
    int makeactive )
```

Definition at line 77 of file `device.c`.

References `_start_dev()`, `USB_DELAY_DEFAULT`, and `usbdevice::usbdelay`.

```
77                                     {
78     // Force USB interval to 10ms during initial setup phase; return to nominal 5ms after setup completes.
79     kb->usbdelay = 10;
80     int res = _start_dev(kb, makeactive);
81     kb->usbdelay = USB_DELAY_DEFAULT;
82     return res;
83 }
```

Here is the call graph for this function:



5.4.2.10 start_kb_nrgb()

```
int start_kb_nrgb (
    usbdevice * kb,
    int makeactive )
```

Definition at line 9 of file device_keyboard.c.

References usbdevice::active, NK95_HWOFF, nk95cmd, and usbdevice::pollrate.

```

9          {
10         // Put the non-RGB K95 into software mode. Nothing else needs to be done hardware wise
11         nk95cmd(kb, NK95_HWOFF);
12         // Fill out RGB features for consistency, even though the keyboard doesn't have them
13         kb->active = 1;
14         kb->pollrate = -1;
15         return 0;
16     }
```

5.4.3 Variable Documentation

5.4.3.1 devmutex

```
pthread_mutex_t devmutex[9]
```

Definition at line 12 of file device.c.

Referenced by _updateconnected(), quitWithLock(), and usb_rm_device().

5.4.3.2 inputmutex

```
pthread_mutex_t inputmutex[9]
```

Definition at line 13 of file device.c.

5.4.3.3 keyboard

```
usbdevice keyboard[9]
```

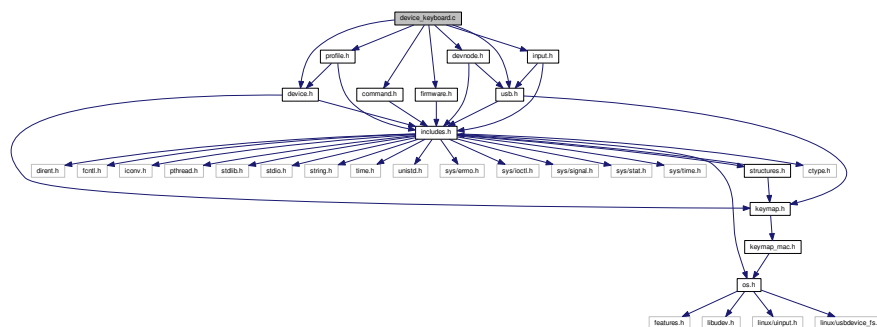
Definition at line 10 of file device.c.

Referenced by `_mkdevpath()`, `_mknotifynode()`, `_rmnotifynode()`, `_setupusb()`, `_updateconnected()`, `closeusb()`, `main()`, `mkfwnode()`, `os_closeusb()`, `os_inputmain()`, `os_inputopen()`, `os_setupusb()`, `quitWithLock()`, `rmdevpath()`, `usb_rm_device()`, and `usbadd()`.

5.5 device_keyboard.c File Reference

```
#include "command.h"
#include "device.h"
#include "devnode.h"
#include "firmware.h"
#include "input.h"
#include "profile.h"
#include "usb.h"
```

Include dependency graph for device_keyboard.c:



Functions

- int `start_kb_nrgb` (`usbdevice` *kb, int makeactive)
- int `setactive_kb` (`usbdevice` *kb, int active)
- int `cmd_active_kb` (`usbdevice` *kb, `usbmode` *dummy1, int dummy2, int dummy3, const char *dummy4)
- int `cmd_idle_kb` (`usbdevice` *kb, `usbmode` *dummy1, int dummy2, int dummy3, const char *dummy4)
- void `setmodeindex_nrgb` (`usbdevice` *kb, int index)

5.5.1 Function Documentation

5.5.1.1 cmd_active_kb()

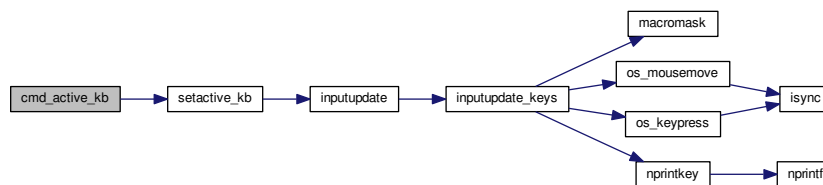
```
int cmd_active_kb (
    usbdevice * kb,
    usbmode * dummy1,
    int dummy2,
    int dummy3,
    const char * dummy4 )
```

Definition at line 112 of file device_keyboard.c.

References setactive_kb().

```
112
113     return setactive_kb(kb, 1);
114 }
```

Here is the call graph for this function:



5.5.1.2 cmd_idle_kb()

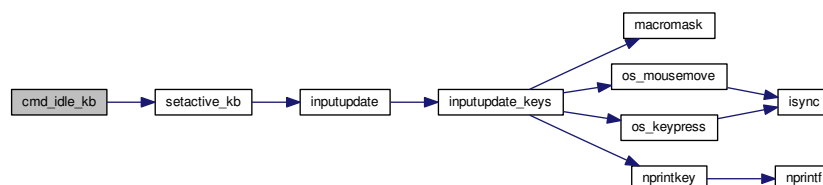
```
int cmd_idle_kb (
    usbdevice * kb,
    usbmode * dummy1,
    int dummy2,
    int dummy3,
    const char * dummy4 )
```

Definition at line 116 of file device_keyboard.c.

References setactive_kb().

```
116
117     return setactive_kb(kb, 0);
118 }
```

Here is the call graph for this function:



5.5.1.3 setactive_kb()

```
int setactive_kb (
    usbdevice * kb,
    int active )
```

Definition at line 18 of file device_keyboard.c.

References ACT_LIGHT, ACT_LOCK, ACT_M1, ACT_M2, ACT_M3, ACT_MR_RING, usbdevice::active, DELAY_MEDIUM, lighting::forceupdate, imutex, IN_CORSAIR, IN_HID, usbdevice::input, inputupdate(), keymap, usbinput::keys, usbprofile::lastlight, MSG_SIZE, N_KEYS_HW, NEEDS_FW_UPDATE, usbdevice::profile, usb_send, and usbdevice::vtable.

Referenced by cmd_active_kb(), and cmd_idle_kb().

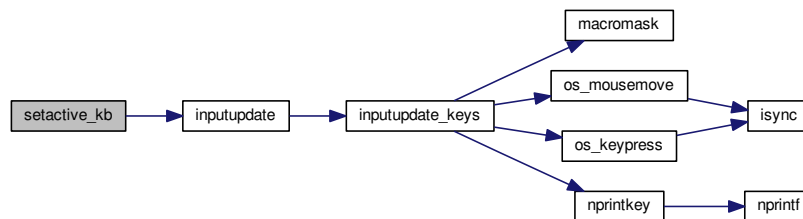
```
18                                     {
19     if (NEEDS_FW_UPDATE(kb))
20         return 0;
21
22     pthread_mutex_lock(&imutex(kb));
23     kb->active = !active;
24     kb->profile->lastlight.forceupdate = 1;
25     // Clear input
26     memset(&kb->input.keys, 0, sizeof(kb->input.keys));
27     inputupdate(kb);
28     pthread_mutex_unlock(&imutex(kb));
29
30     uchar msg[3][MSG_SIZE] = {
31         { 0x07, 0x04, 0 },           // Disables or enables HW control for top row
32         { 0x07, 0x40, 0 },           // Selects key input
33         { 0x07, 0x05, 2, 0, 0x03, 0x00 } // Commits key input selection
34     };
35     if(active){
36         // Put the M-keys (K95) as well as the Brightness/Lock keys into software-controlled mode.
37         msg[0][2] = 2;
38         if(!usb_send(kb, msg[0], 1))
39             return -1;
40         DELAY_MEDIUM(kb);
41         // Set input mode on the keys. They must be grouped into packets of 60 bytes (+ 4 bytes header)
42         // Keys are referenced in byte pairs, with the first byte representing the key and the second byte
43         // representing the mode.
44         for(int key = 0; key < N_KEYS_HW; ){
45             int pair;
46             for(pair = 0; pair < 30 && key < N_KEYS_HW; pair++, key++){
47                 // Select both standard and Corsair input. The standard input will be ignored except in
48                 BIOS mode.
49                 uchar action = IN_HID | IN_CORSAIR;
50                 // Additionally, make MR activate the MR ring (this is disabled for now, may be back later)
51                 //if(keymap[key].name && !strcmp(keymap[key].name, "mr"))
52                 //    action |= ACT_MR_RING;
53                 msg[1][4 + pair * 2] = key;
54                 msg[1][5 + pair * 2] = action;
55             }
56             // Byte 2 = pair count (usually 30, less on final message)
57             msg[1][2] = pair;
58             if(!usb_send(kb, msg[1], 1))
59                 return -1;
60         }
61         // Commit new input settings
62         if(!usb_send(kb, msg[2], 1))
63             return -1;
64         DELAY_MEDIUM(kb);
65     } else {
66         // Set the M-keys back into hardware mode, restore hardware RGB profile. It has to be sent twice
67         // for some reason.
68         msg[0][2] = 1;
69         if(!usb_send(kb, msg[0], 1))
70             return -1;
71         DELAY_MEDIUM(kb);
72         if(!usb_send(kb, msg[0], 1))
73             return -1;
74         DELAY_MEDIUM(kb);
75     }
76     #ifdef OS_LINUX
77     // On OSX the default key mappings are fine. On Linux, the G keys will freeze the keyboard. Set the
78     keyboard entirely to HID input.
79     for(int key = 0; key < N_KEYS_HW; ){
80         int pair;
81         for(pair = 0; pair < 30 && key < N_KEYS_HW; pair++, key++){
```

```

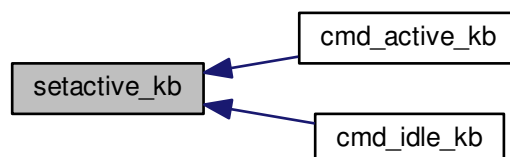
77         uchar action = IN_HID;
78         // Enable hardware actions
79         if(keymap[key].name){
80             if(!strcmp(keymap[key].name, "mr"))
81                 action = ACT_MR_RING;
82             else if(!strcmp(keymap[key].name, "m1"))
83                 action = ACT_M1;
84             else if(!strcmp(keymap[key].name, "m2"))
85                 action = ACT_M2;
86             else if(!strcmp(keymap[key].name, "m3"))
87                 action = ACT_M3;
88             else if(!strcmp(keymap[key].name, "light"))
89                 action = ACT_LIGHT;
90             else if(!strcmp(keymap[key].name, "lock"))
91                 action = ACT_LOCK;
92         }
93         msg[1][4 + pair * 2] = key;
94         msg[1][5 + pair * 2] = action;
95     }
96     // Byte 2 = pair count (usually 30, less on final message)
97     msg[1][2] = pair;
98     if(!usbSend(kb, msg[1], 1))
99         return -1;
100 }
101 // Commit new input settings
102 if(!usbSend(kb, msg[2], 1))
103     return -1;
104 DELAY_MEDIUM(kb);
105 #endif
106 }
107 // Update indicator LEDs if the profile contains settings for them
108 kb->vtable->updateIndicators(kb, 0);
109 return 0;
110 }

```

Here is the call graph for this function:



Here is the caller graph for this function:



5.5.1.4 setmodeindex_nrgb()

```
void setmodeindex_nrgb (
    usbdevice * kb,
    int index )
```

Definition at line 120 of file device_keyboard.c.

References NK95_M1, NK95_M2, NK95_M3, and nk95cmd.

```
120                                     {
121     switch(index % 3){
122     case 0:
123         nk95cmd(kb, NK95_M1);
124         break;
125     case 1:
126         nk95cmd(kb, NK95_M2);
127         break;
128     case 2:
129         nk95cmd(kb, NK95_M3);
130         break;
131     }
132 }
```

5.5.1.5 start_kb_nrgb()

```
int start_kb_nrgb (
    usbdevice * kb,
    int makeactive )
```

Definition at line 9 of file device_keyboard.c.

References usbdevice::active, NK95_HWOFF, nk95cmd, and usbdevice::pollrate.

```
9                                     {
10     // Put the non-RGB K95 into software mode. Nothing else needs to be done hardware wise
11     nk95cmd(kb, NK95_HWOFF);
12     // Fill out RGB features for consistency, even though the keyboard doesn't have them
13     kb->active = 1;
14     kb->pollrate = -1;
15     return 0;
16 }
```

5.6 device_mouse.c File Reference

```
#include "command.h"
#include "device.h"
#include "devnode.h"
#include "firmware.h"
#include "input.h"
#include "profile.h"
```


5.6.1.2 cmd_idle_mouse()

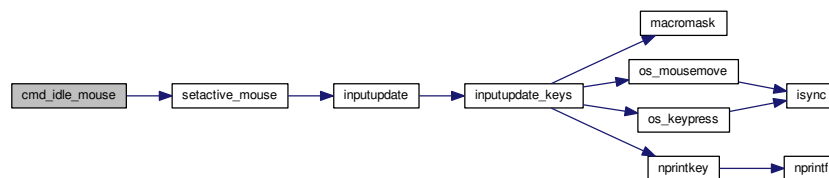
```
int cmd_idle_mouse (
    usbdevice * kb,
    usbmode * dummy1,
    int dummy2,
    int dummy3,
    const char * dummy4 )
```

Definition at line 48 of file device_mouse.c.

References [setactive_mouse\(\)](#).

```
48
49     return setactive_mouse(kb, 0);
50 }
```

Here is the call graph for this function:



5.6.1.3 cmd_pollrate()

```
int cmd_pollrate (
    usbdevice * kb,
    usbmode * dummy1,
    int dummy2,
    int rate,
    const char * dummy3 )
```

Definition at line 52 of file device_mouse.c.

References [MSG_SIZE](#), [usbdevice::pollrate](#), and [usbdevice::send](#).

```
52
53     uchar msg[MSG_SIZE] = {
54         0x07, 0x0a, 0, 0, (uchar)rate
55     };
56     if(!usbdevice::send(kb, msg, 1))
57         return -1;
58     // Device should disconnect+reconnect, but update the poll rate field in case it doesn't
59     kb->pollrate = rate;
60     return 0;
61 }
```

5.6.1.4 setactive_mouse()

```
int setactive_mouse (
    usbdevice * kb,
    int active )
```

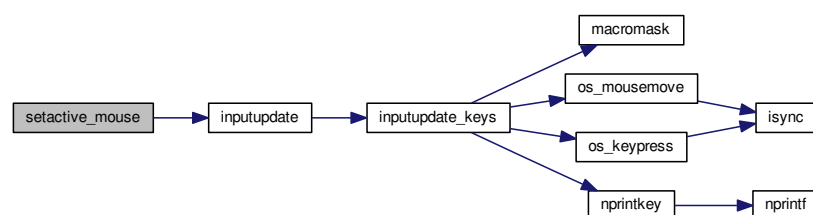
Definition at line 9 of file device_mouse.c.

References `usbdevice::active`, `lighting::forceupdate`, `imutex`, `IN_CORSAIR`, `IN_HID`, `usbdevice::input`, `inputupdate()`, `usbinput::keys`, `usbprofile::lastlight`, `MSG_SIZE`, `NEEDS_FW_UPDATE`, `usbdevice::profile`, and `usbsend`.

Referenced by `cmd_active_mouse()`, and `cmd_idle_mouse()`.

```
9                                     {
10     if(NEEDS_FW_UPDATE(kb))
11         return 0;
12     const int keycount = 20;
13     uchar msg[2][MSG_SIZE] = {
14         { 0x07, 0x04, 0 },          // Disables or enables HW control for DPI and Sniper button
15         { 0x07, 0x40, keycount, 0 }, // Select button input (similar to the packet sent to
16         keyboards, but lacks a commit packet)
17     };
18     if(active)
19         // Put the mouse into SW mode
20         msg[0][2] = 2;
21     else
22         // Restore HW mode
23         msg[0][2] = 1;
24     pthread_mutex_lock(imutex(kb));
25     kb->active = !active;
26     kb->profile->lastlight.forceupdate = 1;
27     // Clear input
28     memset(&kb->input.keys, 0, sizeof(kb->input.keys));
29     inputupdate(kb);
30     pthread_mutex_unlock(imutex(kb));
31     if(!usbsend(kb, msg[0], 1))
32         return -1;
33     if(active){
34         // Set up key input
35         if(!usbsend(kb, msg[1], 1))
36             return -1;
37         for(int i = 0; i < keycount; i++){
38             msg[1][i * 2 + 4] = i + 1;
39             msg[1][i * 2 + 5] = (i < 6 ? IN_HID : IN_CORSAIR);
40         }
41     }
42     return 0;
43 }
```

Here is the call graph for this function:



Variables

- const [devcmd vtable_keyboard](#)
- const [devcmd vtable_keyboard_nonrgb](#)
- const [devcmd vtable_mouse](#)

5.7.1 Function Documentation

5.7.1.1 cmd_io_none()

```
static int cmd_io_none (
    usbdevice * kb,
    usbmode * dummy1,
    int dummy2,
    int dummy3,
    const char * dummy4 ) [static]
```

Definition at line 13 of file device_vtable.c.

```
13                                     {
14     return 0;
15 }
```

5.7.1.2 cmd_macro_none()

```
static void cmd_macro_none (
    usbdevice * kb,
    usbmode * dummy1,
    int dummy2,
    const char * dummy3,
    const char * dummy4 ) [static]
```

Definition at line 16 of file device_vtable.c.

```
16     {
17 }
```

5.7.1.3 cmd_none()

```
static void cmd_none (
    usbdevice * kb,
    usbmode * dummy1,
    int dummy2,
    int dummy3,
    const char * dummy4 ) [static]
```

Definition at line 11 of file device_vtable.c.

```
11                                     {
12 }
```

5.7.1.4 int1_int_none()

```
static int int1_int_none (
    usbdevice * kb,
    int dummy ) [static]
```

Definition at line 23 of file device_vtable.c.

```
23                                     {
24     return 0;
25 }
```

5.7.1.5 int1_void_none()

```
static void int1_void_none (
    usbdevice * kb,
    int dummy ) [static]
```

Definition at line 21 of file device_vtable.c.

```
21                                     {
22 }
```

5.7.1.6 loadprofile_none()

```
static int loadprofile_none (
    usbdevice * kb ) [static]
```

Definition at line 18 of file device_vtable.c.

```
18                                     {
19     return 0;
20 }
```


Data Structures

- struct [_readlines_ctx](#)

Macros

- #define [S_GID_READ](#) ([gid](#) >= 0 ? [S_CUSTOM_R](#) : [S_READ](#))
- #define [MAX_BUFFER](#) (1024 * 1024 - 1)

Functions

- int [rm_recursive](#) (const char *path)
- void [_updateconnected](#) ()
_updateconnected Update the list of connected devices.
- void [updateconnected](#) ()
Update the list of connected devices.
- int [_mknotifynode](#) ([usbdevice](#) *kb, int notify)
- int [mknotifynode](#) ([usbdevice](#) *kb, int notify)
Creates a notification node for the specified keyboard.
- int [_rmnotifynode](#) ([usbdevice](#) *kb, int notify)
- int [rmnotifynode](#) ([usbdevice](#) *kb, int notify)
Removes a notification node for the specified keyboard.
- static int [_mkdevpath](#) ([usbdevice](#) *kb)
- int [mkdevpath](#) ([usbdevice](#) *kb)
Create a dev path for the keyboard at index. Returns 0 on success.
- int [rmdevpath](#) ([usbdevice](#) *kb)
Remove the dev path for the keyboard at index. Returns 0 on success.
- int [mkfwnode](#) ([usbdevice](#) *kb)
Writes a keyboard's firmware version and poll rate to its device node.
- void [readlines_ctx_init](#) ([readlines_ctx](#) *ctx)
- void [readlines_ctx_free](#) ([readlines_ctx](#) ctx)
- unsigned [readlines](#) (int fd, [readlines_ctx](#) ctx, const char **input)

Variables

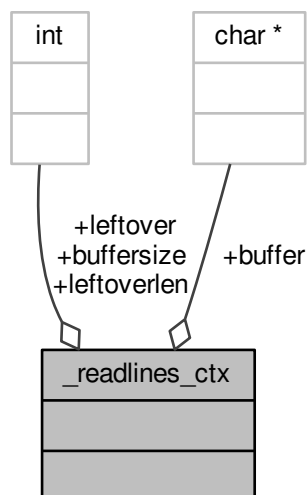
- const char *const [devpath](#) = "/dev/input/ckb"
- long [gid](#) = -1
Group ID for the control nodes. -1 to give read/write access to everybody.

5.8.1 Data Structure Documentation

5.8.1.1 struct _readlines_ctx

Definition at line 335 of file devnode.c.

Collaboration diagram for _readlines_ctx:



Data Fields

char *	buffer	
int	buffer	
int	leftover	
int	leftoverlen	

5.8.2 Macro Definition Documentation

5.8.2.1 MAX_BUFFER

```
#define MAX_BUFFER (1024 * 1024 - 1)
```

Definition at line 334 of file devnode.c.

Referenced by `readlines()`.

5.8.2.2 S_GID_READ

```
#define S_GID_READ (gid >= 0 ? S_CUSTOM_R : S_READ)
```

Definition at line 17 of file devnode.c.

Referenced by `_mkdevpath()`, `_mknotifynode()`, `_updateconnected()`, and `mkfwnode()`.

5.8.3 Function Documentation

5.8.3.1 _mkdevpath()

```
static int _mkdevpath (
    usbdevice * kb ) [static]
```

Definition at line 136 of file devnode.c.

References `_mknotifynode()`, `_updateconnected()`, `ckb_err`, `ckb_warn`, `devpath`, `FEAT_ADJRATE`, `FEAT_BIND`, `FEAT_FWUPDATE`, `FEAT_FWVERSION`, `FEAT_MONOCHROME`, `FEAT_NOTIFY`, `FEAT_POLLRATE`, `FEAT_RGB`, `gid`, `HAS_FEATURES`, `INDEX_OF`, `usbdevice::infifo`, `keyboard`, `mkfwnode()`, `usbdevice::name`, `usbdevice::product`, `product_str()`, `rm_recursive()`, `S_CUSTOM`, `S_GID_READ`, `S_READ`, `S_READDIR`, `S_READWRITE`, `usbdevice::serial`, `usbdevice::vendor`, and `vendor_str()`.

Referenced by `mkdevpath()`.

```
136         {
137     int index = INDEX_OF(kb, keyboard);
138     // Create the control path
139     char path[strlen(devpath) + 2];
140     snprintf(path, sizeof(path), "%s%d", devpath, index);
141     if(rm_recursive(path) != 0 && errno != ENOENT){
142         ckb_err("Unable to delete %s: %s\n", path, strerror(errno));
143         return -1;
144     }
145     if(mkdir(path, S_READDIR) != 0){
146         ckb_err("Unable to create %s: %s\n", path, strerror(errno));
147         rm_recursive(path);
148         return -1;
149     }
150     if(gid >= 0)
151         chown(path, 0, gid);
152
153     if(kb == keyboard + 0){
154         // Root keyboard: write a list of devices
155         _updateconnected();
156         // Write version number
157         char vpath[sizeof(path) + 8];
158         snprintf(vpath, sizeof(vpath), "%s/version", path);
159         FILE* vfile = fopen(vpath, "w");
160         if(vfile){
161             fprintf(vfile, "%s\n", CKB_VERSION_STR);
162             fclose(vfile);
163             chmod(vpath, S_GID_READ);
164             if(gid >= 0)
165                 chown(vpath, 0, gid);
166         } else {
167             ckb_warn("Unable to create %s: %s\n", vpath, strerror(errno));
168             remove(vpath);
169         }
170         // Write PID
171         char ppath[sizeof(path) + 4];
172         snprintf(ppath, sizeof(ppath), "%s/pid", path);
173         FILE* pfile = fopen(ppath, "w");
174         if(pfile){
175             fprintf(pfile, "%u\n", getpid());
```

```

176         fclose(pfile);
177         chmod(ppath, S_READ);
178         if(gid >= 0)
179             chown(vpath, 0, gid);
180     } else {
181         ckb_warn("Unable to create %s: %s\n", ppath, strerror(errno));
182         remove(ppath);
183     }
184 } else {
185     // Create command FIFO
186     char inpath[sizeof(path) + 4];
187     snprintf(inpath, sizeof(inpath), "%s/cmd", path);
188     if(mkfifo(inpath, gid >= 0 ? S_CUSTOM : S_READWRITE) != 0
189         // Open the node in RDWR mode because RDONLY will lock the thread
190         || (kb->infifo = open(inpath, O_RDWR) + 1) == 0) {
191         // Add one to the FD because 0 is a valid descriptor, but ckb uses 0 for uninitialized devices
192         ckb_err("Unable to create %s: %s\n", inpath, strerror(errno));
193         rm_recursive(path);
194         kb->infifo = 0;
195         return -1;
196     }
197     if(gid >= 0)
198         fchown(kb->infifo - 1, 0, gid);
199
200     // Create notification FIFO
201     _mknotifinode(kb, 0);
202
203     // Write the model and serial to files
204     char mpath[sizeof(path) + 6], spath[sizeof(path) + 7];
205     snprintf(mpath, sizeof(mpath), "%s/model", path);
206     snprintf(spath, sizeof(spath), "%s/serial", path);
207     FILE* mfile = fopen(mpath, "w");
208     if(mfile) {
209         fputs(kb->name, mfile);
210         fputc('\n', mfile);
211         fclose(mfile);
212         chmod(mpath, S_GID_READ);
213         if(gid >= 0)
214             chown(mpath, 0, gid);
215     } else {
216         ckb_warn("Unable to create %s: %s\n", mpath, strerror(errno));
217         remove(mpath);
218     }
219     FILE* sfile = fopen(spath, "w");
220     if(sfile) {
221         fputs(kb->serial, sfile);
222         fputc('\n', sfile);
223         fclose(sfile);
224         chmod(spath, S_GID_READ);
225         if(gid >= 0)
226             chown(spath, 0, gid);
227     } else {
228         ckb_warn("Unable to create %s: %s\n", spath, strerror(errno));
229         remove(spath);
230     }
231     // Write the keyboard's features
232     char fpath[sizeof(path) + 9];
233     snprintf(fpath, sizeof(fpath), "%s/features", path);
234     FILE* ffile = fopen(fpath, "w");
235     if(ffile) {
236         fprintf(ffile, "%s %s", vendor_str(kb->vendor),
237             product_str(kb->product));
238         if(HAS_FEATURES(kb, FEAT_MONOCHROME))
239             fputs(" monochrome", ffile);
240         if(HAS_FEATURES(kb, FEAT_RGB))
241             fputs(" rgb", ffile);
242         if(HAS_FEATURES(kb, FEAT_POLLRATE))
243             fputs(" pollrate", ffile);
244         if(HAS_FEATURES(kb, FEAT_ADJRATE))
245             fputs(" adjrate", ffile);
246         if(HAS_FEATURES(kb, FEAT_BIND))
247             fputs(" bind", ffile);
248         if(HAS_FEATURES(kb, FEAT_NOTIFY))
249             fputs(" notify", ffile);
250         if(HAS_FEATURES(kb, FEAT_FWVERSION))
251             fputs(" fwversion", ffile);
252         if(HAS_FEATURES(kb, FEAT_FWUPDATE))
253             fputs(" fwupdate", ffile);
254         fputc('\n', ffile);
255         fclose(ffile);
256         chmod(fpath, S_GID_READ);
257         if(gid >= 0)
258             chown(fpath, 0, gid);
259     } else {
260         ckb_warn("Unable to create %s: %s\n", fpath, strerror(errno));
261         remove(fpath);
262     }

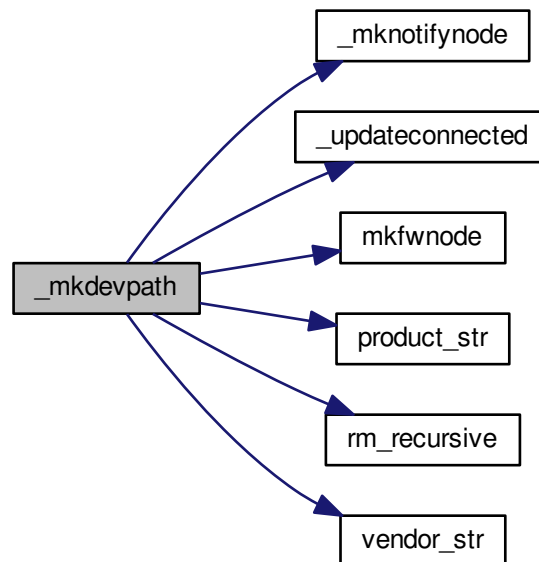
```

```

262         // Write firmware version and poll rate
263         mkfwnode(kb);
264     }
265     return 0;
266 }

```

Here is the call graph for this function:



Here is the caller graph for this function:



5.8.3.2 _mknotifynode()

```

int _mknotifynode (
    usbdevice * kb,
    int notify )

```

Definition at line 87 of file `devnode.c`.

References `ckb_warn`, `devpath`, `gid`, `INDEX_OF`, `keyboard`, `usbdevice::outfifo`, `OUTFIFO_MAX`, and `S_GID_READ`.

Referenced by `_mkdevpath()`, and `mknotifynode()`.

5.8.3.5 mkdevpath()

```
int mkdevpath (
    usbdevice * kb )
```

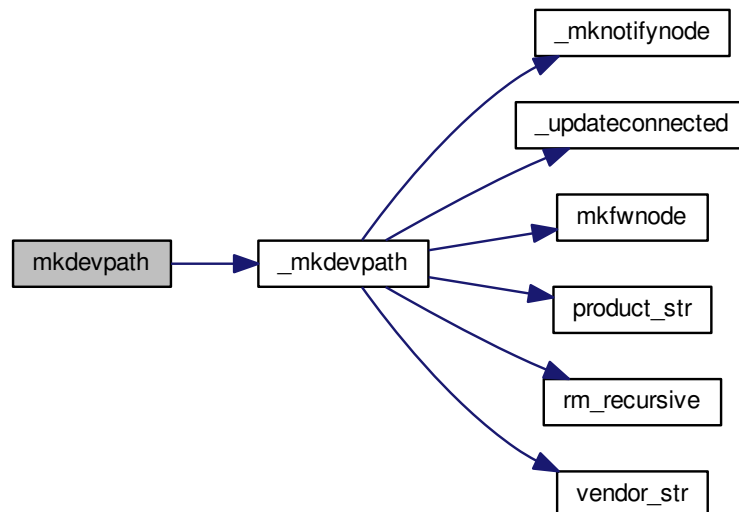
Definition at line 268 of file devnode.c.

References `_mkdevpath()`, `euid_guard_start`, and `euid_guard_stop`.

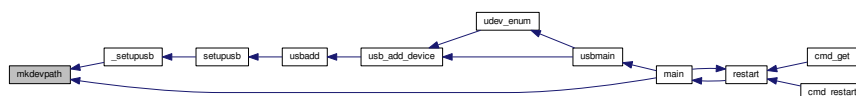
Referenced by `_setupusb()`, and `main()`.

```
268                                     {
269     euid_guard_start;
270     int res = _mkdevpath(kb);
271     euid_guard_stop;
272     return res;
273 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



5.8.3.6 mkfwnode()

```
int mkfwnode (
    usbdevice * kb )
```

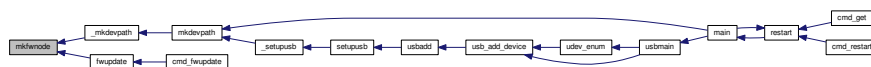
Definition at line 299 of file devnode.c.

References `ckb_warn`, `devpath`, `usbdevice::fwversion`, `gid`, `INDEX_OF`, `keyboard`, `usbdevice::pollrate`, and `S_GID_READ`.

Referenced by `_mkdevpath()`, and `fwupdate()`.

```
299     {
300     int index = INDEX_OF(kb, keyboard);
301     char fwpath[strlen(devpath) + 12];
302     snprintf(fwpath, sizeof(fwpath), "%s%d/fwversion", devpath, index);
303     FILE* fwfile = fopen(fwpath, "w");
304     if(fwfile){
305         fprintf(fwfile, "%04x", kb->fwversion);
306         fputc('\n', fwfile);
307         fclose(fwfile);
308         chmod(fwpath, S_GID_READ);
309         if(gid >= 0)
310             chown(fwpath, 0, gid);
311     } else {
312         ckb_warn("Unable to create %s: %s\n", fwpath, strerror(errno));
313         remove(fwpath);
314         return -1;
315     }
316     char ppath[strlen(devpath) + 11];
317     snprintf(ppath, sizeof(ppath), "%s%d/pollrate", devpath, index);
318     FILE* pfile = fopen(ppath, "w");
319     if(pfile){
320         fprintf(pfile, "%d ms", kb->pollrate);
321         fputc('\n', pfile);
322         fclose(pfile);
323         chmod(ppath, S_GID_READ);
324         if(gid >= 0)
325             chown(ppath, 0, gid);
326     } else {
327         ckb_warn("Unable to create %s: %s\n", fwpath, strerror(errno));
328         remove(ppath);
329         return -2;
330     }
331     return 0;
332 }
```

Here is the caller graph for this function:



5.8.3.7 mknotifynode()

```
int mknotifynode (
    usbdevice * kb,
    int notify )
```

Definition at line 108 of file devnode.c.

References `_mknotifynode()`, `euid_guard_start`, and `euid_guard_stop`.

Referenced by `readcmd()`.

```

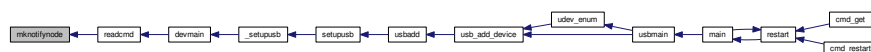
108                                     {
109     euid_guard_start;
110     int res = _mknotifynode(kb, notify);
111     euid_guard_stop;
112     return res;
113 }

```

Here is the call graph for this function:



Here is the caller graph for this function:



5.8.3.8 readlines()

```

unsigned readlines (
    int fd,
    readlines_ctx ctx,
    const char ** input )

```

Definition at line 353 of file devnode.c.

References `_readlines_ctx::buffer`, `_readlines_ctx::buffersize`, `ckb_warn`, `_readlines_ctx::leftover`, `_readlines_ctx::leftoverlen`, and `MAX_BUFFER`.

Referenced by `devmain()`.

```

353                                     {
354     // Move any data left over from a previous read to the start of the buffer
355     char* buffer = ctx->buffer;
356     int buffersize = ctx->buffersize;
357     int leftover = ctx->leftover, leftoverlen = ctx->leftoverlen;
358     memcpy(buffer, buffer + leftover, leftoverlen);
359     // Read data from the file
360     ssize_t length = read(fd, buffer + leftoverlen, buffersize - leftoverlen);
361     length = (length < 0 ? 0 : length) + leftoverlen;
362     leftover = ctx->leftover = leftoverlen = ctx->leftoverlen = 0;
363     if(length <= 0){
364         *input = 0;
365         return 0;
366     }
367     // Continue buffering until all available input is read or there's no room left
368     while(length == buffersize){
369         if(buffersize == MAX_BUFFER)
370             break;

```

```

371     int oldsize = buffersize;
372     buffersize += 4096;
373     ctx->buffersize = buffersize;
374     buffer = ctx->buffer = realloc(buffer, buffersize + 1);
375     ssize_t length2 = read(fd, buffer + oldsize, buffersize - oldsize);
376     if(length2 <= 0)
377         break;
378     length += length2;
379 }
380 buffer[length] = 0;
381 // Input should be issued one line at a time and should end with a newline.
382 char* lastline = memchr(buffer, '\n', length);
383 if(lastline == buffer + length - 1){
384     // If the buffer ends in a newline, process the whole string
385     *input = buffer;
386     return length;
387 } else if(lastline){
388     // Otherwise, chop off the last line but process everything else
389     *lastline = 0;
390     leftover = ctx->leftover = lastline + 1 - buffer;
391     leftoverlen = ctx->leftoverlen = length - leftover;
392     *input = buffer;
393     return leftover - 1;
394 } else {
395     // If a newline wasn't found at all, process the whole buffer next time
396     *input = 0;
397     if(length == MAX_BUFFER){
398         // Unless the buffer is completely full, in which case discard it
399         ckb_warn("Too much input (1MB). Dropping.\n");
400         return 0;
401     }
402     leftoverlen = ctx->leftoverlen = length;
403     return 0;
404 }
405 }

```

Here is the caller graph for this function:



5.8.3.9 readlines_ctx_free()

```

void readlines_ctx_free (
    readlines_ctx ctx )

```

Definition at line 348 of file devnode.c.

References `_readlines_ctx::buffer`.

Referenced by `devmain()`.

```

348     {
349     free(ctx->buffer);
350     free(ctx);
351 }

```

Here is the caller graph for this function:



5.8.3.10 readlines_ctx_init()

```
void readlines_ctx_init (
    readlines_ctx * ctx )
```

Definition at line 341 of file devnode.c.

References `_readlines_ctx::buffer`.

Referenced by `devmain()`.

```
341 {
342     // Allocate buffers to store data
343     *ctx = calloc(1, sizeof(struct _readlines_ctx));
344     int buffersize = (*ctx)->buffersize = 4095;
345     (*ctx)->buffer = malloc(buffersize + 1);
346 }
```

Here is the caller graph for this function:



5.8.3.11 rm_recursive()

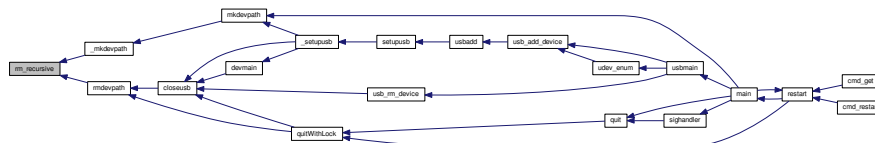
```
int rm_recursive (
    const char * path )
```

Definition at line 19 of file devnode.c.

Referenced by `_mkdevpath()`, and `rmdevpath()`.

```
19 {
20     DIR* dir = opendir(path);
21     if(!dir)
22         return remove(path);
23     struct dirent* file;
24     while((file = readdir(dir))
25     {
26         if(!strcmp(file->d_name, ".") || !strcmp(file->d_name, ".."))
27             continue;
28         char path2[FILENAME_MAX];
29         snprintf(path2, FILENAME_MAX, "%s/%s", path, file->d_name);
30         int stat = rm_recursive(path2);
31         if(stat != 0)
32             return stat;
33     }
34     closedir(dir);
35     return remove(path);
36 }
```

Here is the caller graph for this function:



```
int rmdevpath (
    usbdevice * kb )
```

References `_rmnotifynode()`, `ckb_info`, `ckb_warn`, `devpath`, `euid_guard_start`, `euid_guard_stop`, `INDEX_OF`, `usbdevice::infifo`, `keyboard`, `OUTFIFO_MAX`, and `rm_recursive()`.

```

275     {
276     uuid_guard_start;
277     int index = INDEX_OF(kb, keyboard);
278     if(kb->ininfo != 0){
279     #ifdef OS_LINUX
280         write(kb->ininfo - 1, "\n", 1); // hack to prevent the FIFO thread from perma-blocking
281     #endif
282         close(kb->ininfo - 1);
283         kb->ininfo = 0;
284     }
285     for(int i = 0; i < OUTFIFO_MAX; i++)
286         _rmnotifynode(kb, i);
287     char path[strlen(devpath) + 2];
288     snprintf(path, sizeof(path), "%s%d", devpath, index);
289     if(rm_recursive(path) != 0 && errno != ENOENT){
290         ckb_warn("Unable to delete %s: %s\n", path, strerror(errno));
291         uuid_guard_stop;
292         return -1;
293     }
294     ckb_info("Removed device path %s\n", path);
295     uuid_guard_stop;
296     return 0;
297 }

```

```
graph LR; rmdevpath --> _rmnotifynode; rmdevpath --> rm_recursive
```

5.8.3.13 rmnotifynode()

```
int rmnotifynode (
    usbdevice * kb,
    int notify )
```

Definition at line 129 of file devnode.c.

References `_rmnotifynode()`, `euid_guard_start`, and `euid_guard_stop`.

Referenced by `readcmd()`.

```
129                                     {
130     euid_guard_start;
131     int res = _rmnotifynode(kb, notify);
132     euid_guard_stop;
133     return res;
134 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



5.8.3.14 updateconnected()

```
void updateconnected ( )
```

Definition at line 81 of file devnode.c.

References `_updateconnected()`, `euid_guard_start`, and `euid_guard_stop`.

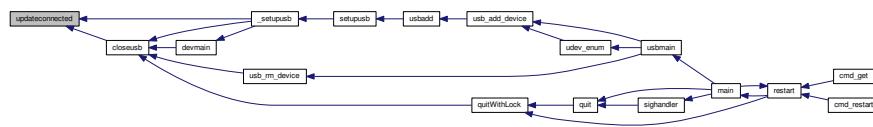
Referenced by `_setupusb()`, and `closeusb()`.

```
81     {
82     euid_guard_start;
83     _updateconnected();
84     euid_guard_stop;
85 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



5.8.4 Variable Documentation

5.8.4.1 devpath

```
const char* const devpath = "/dev/input/ckb"
```

Definition at line 11 of file devnode.c.

Referenced by `_mkdevpath()`, `_mknotifynode()`, `_rmnotifynode()`, `_setupusb()`, `_updateconnected()`, `closeusb()`, `main()`, `mkfwnode()`, `os_inputmain()`, `os_setupusb()`, and `rmdevpath()`.

5.8.4.2 gid

```
long gid = -1
```

Definition at line 16 of file devnode.c.

Referenced by `_mkdevpath()`, `_mknotifynode()`, `_updateconnected()`, `main()`, and `mkfwnode()`.

- Creates a notification node for the specified keyboard.*
- int `rmnotifynode` (`usbdevice` *kb, int notify)
- Removes a notification node for the specified keyboard.*
- int `mkfwnode` (`usbdevice` *kb)
- Writes a keyboard's firmware version and poll rate to its device node.*
- void `readlines_ctx_init` (`readlines_ctx` *ctx)
- void `readlines_ctx_free` (`readlines_ctx` ctx)
- unsigned `readlines` (int fd, `readlines_ctx` ctx, const char **input)

Variables

- const char *const `devpath`
Device path base ("/dev/input/ckb" or "/var/run/ckb")
- long `gid`
Group ID for the control nodes. -1 to give read/write access to everybody.

5.9.1 Macro Definition Documentation

5.9.1.1 S_CUSTOM

```
#define S_CUSTOM (S_IRUSR | S_IWUSR | S_IRGRP | S_IWGRP)
```

Definition at line 17 of file devnode.h.

Referenced by `_mkdevpath()`.

5.9.1.2 S_CUSTOM_R

```
#define S_CUSTOM_R (S_IRUSR | S_IWUSR | S_IRGRP)
```

Definition at line 18 of file devnode.h.

5.9.1.3 S_READ

```
#define S_READ (S_IRUSR | S_IRGRP | S_IROTH | S_IWUSR)
```

Definition at line 15 of file devnode.h.

Referenced by `_mkdevpath()`.

5.9.1.4 S_READDIR

```
#define S_READDIR (S_IRWXU | S_IRGRP | S_IROTH | S_IXGRP | S_IXOTH)
```

Definition at line 14 of file devnode.h.

Referenced by `_mkdevpath()`.

5.9.1.5 S_READWRITE

```
#define S_READWRITE (S_IRUSR | S_IRGRP | S_IROTH | S_IWUSR | S_IWGRP | S_IWOTH)
```

Definition at line 16 of file devnode.h.

Referenced by `_mkdevpath()`.

5.9.2 Typedef Documentation

5.9.2.1 readlines_ctx

```
typedef struct _readlines_ctx* readlines_ctx
```

Definition at line 39 of file devnode.h.

5.9.3 Function Documentation

5.9.3.1 mkdevpath()

```
int mkdevpath (
    usbdevice * kb )
```

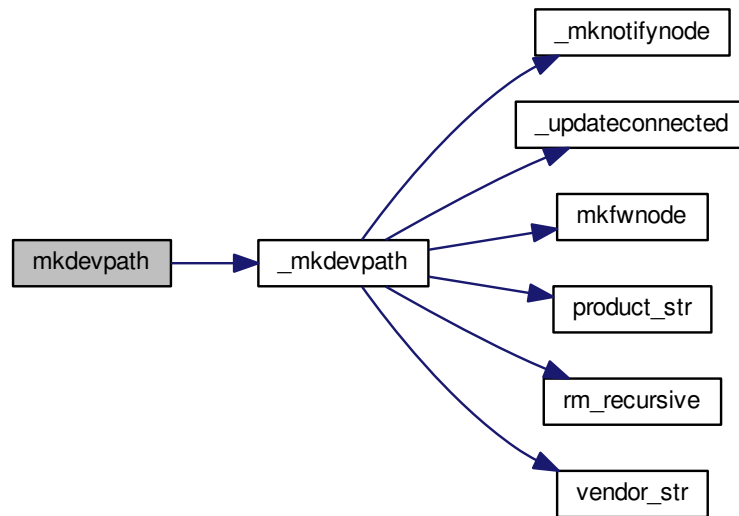
Definition at line 268 of file devnode.c.

References `_mkdevpath()`, `euid_guard_start`, and `euid_guard_stop`.

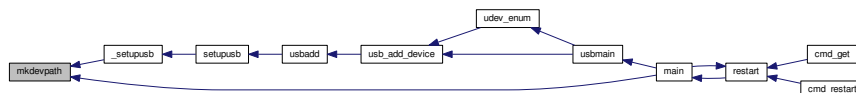
Referenced by `_setupusb()`, and `main()`.

```
268                                     {
269     euid_guard_start;
270     int res = _mkdevpath(kb);
271     euid_guard_stop;
272     return res;
273 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



5.9.3.2 mkfwnode()

```
int mkfwnode (
    usbdevice * kb )
```

Definition at line 299 of file `devnode.c`.

References `ckb_warn`, `devpath`, `usbdevice::fwversion`, `gid`, `INDEX_OF`, `keyboard`, `usbdevice::pollrate`, and `S_GI↵D_READ`.

Referenced by `_mkdevpath()`, and `fwupdate()`.

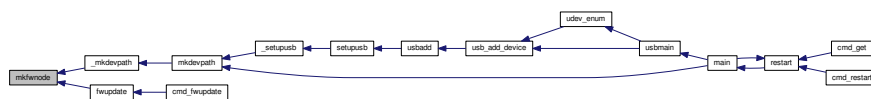
```
299     {
300         int index = INDEX_OF(kb, keyboard);
301         char fwpath[strlen(devpath) + 12];
302         snprintf(fwpath, sizeof(fwpath), "%s%d/fwversion", devpath, index);
303         FILE* fwfile = fopen(fwpath, "w");
304         if(fwfile){
305             fprintf(fwfile, "%04x", kb->fwversion);
```

```

306     fputc('\n', fwfile);
307     fclose(fwfile);
308     chmod(fwpath, S_GID_READ);
309     if(gid >= 0)
310         chown(fwpath, 0, gid);
311 } else {
312     ckb_warn("Unable to create %s: %s\n", fwpath, strerror(errno));
313     remove(fwpath);
314     return -1;
315 }
316 char ppath[strlen(devpath) + 11];
317 snprintf(ppath, sizeof(ppath), "%s%d/pollrate", devpath, index);
318 FILE* pfile = fopen(ppath, "w");
319 if(pfile){
320     fprintf(pfile, "%d ms", kb->pollrate);
321     fputc('\n', pfile);
322     fclose(pfile);
323     chmod(ppath, S_GID_READ);
324     if(gid >= 0)
325         chown(ppath, 0, gid);
326 } else {
327     ckb_warn("Unable to create %s: %s\n", fwpath, strerror(errno));
328     remove(ppath);
329     return -2;
330 }
331 return 0;
332 }

```

Here is the caller graph for this function:



5.9.3.3 mknotifynode()

```

int mknotifynode (
    usbdevice * kb,
    int notify )

```

Definition at line 108 of file devnode.c.

References `_mknotifynode()`, `euid_guard_start`, and `euid_guard_stop`.

Referenced by `readcmd()`.

```

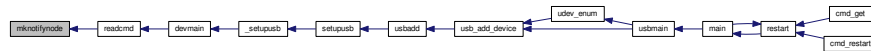
108 {
109     euid_guard_start;
110     int res = _mknotifynode(kb, notify);
111     euid_guard_stop;
112     return res;
113 }

```

Here is the call graph for this function:



Here is the caller graph for this function:



5.9.3.4 readlines()

```

unsigned readlines (
    int fd,
    readlines_ctx ctx,
    const char ** input )

```

Definition at line 353 of file devnode.c.

References `_readlines_ctx::buffer`, `_readlines_ctx::buffersize`, `ckb_warn`, `_readlines_ctx::leftover`, `_readlines_ctx::leftoverlen`, and `MAX_BUFFER`.

Referenced by `devmain()`.

```

353                                     {
354     // Move any data left over from a previous read to the start of the buffer
355     char* buffer = ctx->buffer;
356     int buffersize = ctx->buffersize;
357     int leftover = ctx->leftover, leftoverlen = ctx->leftoverlen;
358     memcpy(buffer, buffer + leftover, leftoverlen);
359     // Read data from the file
360     ssize_t length = read(fd, buffer + leftoverlen, buffersize - leftoverlen);
361     length = (length < 0 ? 0 : length) + leftoverlen;
362     leftover = ctx->leftover = leftoverlen = ctx->leftoverlen = 0;
363     if(length <= 0){
364         *input = 0;
365         return 0;
366     }
367     // Continue buffering until all available input is read or there's no room left
368     while(length == buffersize){
369         if(buffersize == MAX_BUFFER)
370             break;
371         int oldsize = buffersize;
372         buffersize += 4096;
373         ctx->buffersize = buffersize;
374         buffer = ctx->buffer = realloc(buffer, buffersize + 1);
375         ssize_t length2 = read(fd, buffer + oldsize, buffersize - oldsize);
376         if(length2 <= 0)
377             break;
378         length += length2;
379     }
380     buffer[length] = 0;
381     // Input should be issued one line at a time and should end with a newline.
382     char* lastline = memchr(buffer, '\n', length);
383     if(lastline == buffer + length - 1){
384         // If the buffer ends in a newline, process the whole string
385         *input = buffer;
386         return length;
387     } else if(lastline){
388         // Otherwise, chop off the last line but process everything else
389         *lastline = 0;
390         leftover = ctx->leftover = lastline + 1 - buffer;
391         leftoverlen = ctx->leftoverlen = length - leftover;
392         *input = buffer;
393         return leftover - 1;
394     } else {
395         // If a newline wasn't found at all, process the whole buffer next time
396         *input = 0;
397         if(length == MAX_BUFFER){
398             // Unless the buffer is completely full, in which case discard it
399             ckb_warn("Too much input (1MB). Dropping.\n");
400             return 0;
401         }
402         leftoverlen = ctx->leftoverlen = length;
403         return 0;
404     }
405 }

```

Here is the caller graph for this function:



5.9.3.5 readlines_ctx_free()

```
void readlines_ctx_free (
    readlines_ctx ctx )
```

Definition at line 348 of file devnode.c.

References `_readlines_ctx::buffer`.

Referenced by `devmain()`.

```

348                                     {
349     free(ctx->buffer);
350     free(ctx);
351 }
```

Here is the caller graph for this function:



5.9.3.6 readlines_ctx_init()

```
void readlines_ctx_init (
    readlines_ctx * ctx )
```

Definition at line 341 of file devnode.c.

References `_readlines_ctx::buffersize`.

Referenced by `devmain()`.

```

341                                     {
342     // Allocate buffers to store data
343     *ctx = calloc(1, sizeof(struct _readlines_ctx));
344     int buffersize = (*ctx)->buffersize = 4095;
345     (*ctx)->buffer = malloc(buffersize + 1);
346 }
```

Here is the caller graph for this function:



5.9.3.8 rmnotifynode()

```
int rmnotifynode (
    usbdevice * kb,
    int notify )
```

Definition at line 129 of file devnode.c.

References `_rmnotifynode()`, `euid_guard_start`, and `euid_guard_stop`.

Referenced by `readcmd()`.

```
129                                     {
130     euid_guard_start;
131     int res = _rmnotifynode(kb, notify);
132     euid_guard_stop;
133     return res;
134 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



5.9.3.9 updateconnected()

```
void updateconnected ( )
```

Definition at line 81 of file devnode.c.

References `_updateconnected()`, `euid_guard_start`, and `euid_guard_stop`.

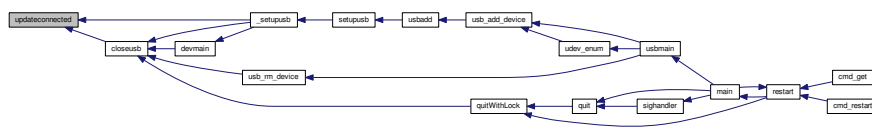
Referenced by `_setupusb()`, and `closeusb()`.

```
81                                     {
82     euid_guard_start;
83     _updateconnected();
84     euid_guard_stop;
85 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



5.9.4 Variable Documentation

5.9.4.1 devpath

```
const char* const devpath
```

Definition at line 8 of file devnode.h.

5.9.4.2 gid

```
long gid
```

Definition at line 16 of file devnode.c.

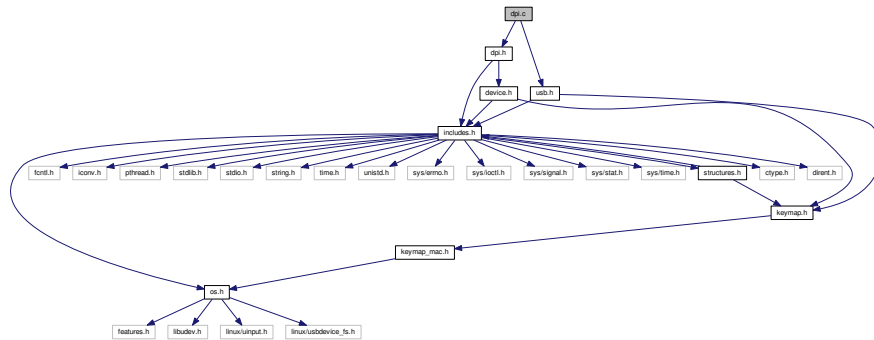
Referenced by `_mkdevpath()`, `_mknotifnode()`, `_updateconnected()`, `main()`, and `mkfwnode()`.

5.10 dpi.c File Reference

```
#include "dpi.h"
```

```
#include "usb.h"
```

Include dependency graph for dpi.c:



Functions

- void `cmd_dpi` (`usbdevice` *kb, `usbmode` *mode, int dummy, const char *stages, const char *values)
- void `cmd_dpisel` (`usbdevice` *kb, `usbmode` *mode, int dummy1, int dummy2, const char *stage)
- void `cmd_lift` (`usbdevice` *kb, `usbmode` *mode, int dummy1, int dummy2, const char *height)
- void `cmd_snap` (`usbdevice` *kb, `usbmode` *mode, int dummy1, int dummy2, const char *enable)
- char * `printdpi` (const `dpiset` *dpi, const `usbdevice` *kb)
- int `updatedpi` (`usbdevice` *kb, int force)
- int `savedpi` (`usbdevice` *kb, `dpiset` *dpi, `lighting` *light)
- int `loaddpi` (`usbdevice` *kb, `dpiset` *dpi, `lighting` *light)

5.10.1 Function Documentation

5.10.1.1 `cmd_dpi()`

```
void cmd_dpi (
    usbdevice * kb,
    usbmode * mode,
    int dummy,
    const char * stages,
    const char * values )
```

Definition at line 4 of file dpi.c.

References `usbmode::dpi`, `DPI_COUNT`, `dpiset::enabled`, `dpiset::x`, and `dpiset::y`.

```

4                                     {
5     int disable = 0;
6     ushort x, y;
7     // Try to scan X,Y values
8     if(sscanf(values, "%hu,%hu", &x, &y) != 2){
9         // If that doesn't work, scan single number
10        if(sscanf(values, "%hu", &x) == 1)
11            y = x;
12        else if(!strncmp(values, "off", 3))
13            // If the right side says "off", disable the level(s)
14            disable = 1;
15        else
16            // Otherwise, quit
17            return;
18    }
19    if((x == 0 || y == 0) && !disable)
20        return;
21    // Scan the left side for stage numbers (comma-separated)
22    int left = strlen(stages);
23    int position = 0, field = 0;
24    char stagename[3];
25    while(position < left && sscanf(stages + position, "%2[^,]%n", stagename, &field) == 1){
26        uchar stagenum;
27        if(sscanf(stagename, "%hhu", &stagenum) && stagenum < DPI_COUNT){
28            // Set DPI for this stage
29            if(disable){
30                mode->dpi.enabled &= ~(1 << stagenum);
31                mode->dpi.x[stagenum] = 0;
32                mode->dpi.y[stagenum] = 0;
33            } else {
34                mode->dpi.enabled |= 1 << stagenum;
35                mode->dpi.x[stagenum] = x;
36                mode->dpi.y[stagenum] = y;
37            }
38        }
39        if(stages[position += field] == ',')
40            position++;
41    }
42 }

```

5.10.1.2 cmd_dpisel()

```

void cmd_dpisel (
    usbdevice * kb,
    usbmode * mode,
    int dummy1,
    int dummy2,
    const char * stage )

```

Definition at line 44 of file dpi.c.

References `dpiset::current`, `usbmode::dpi`, and `DPI_COUNT`.

```

44                                     {
45     uchar stagenum;
46     if(sscanf(stage, "%hhu", &stagenum) != 1)
47         return;
48     if(stagenum > DPI_COUNT)
49         return;
50     mode->dpi.current = stagenum;
51 }

```

5.10.1.3 cmd_lift()

```
void cmd_lift (
    usbdevice * kb,
    usbmode * mode,
    int dummy1,
    int dummy2,
    const char * height )
```

Definition at line 53 of file dpi.c.

References `usbmode::dpi`, `dpiset::lift`, `LIFT_MAX`, and `LIFT_MIN`.

```
53                                     {
54     uchar heightnum;
55     if(sscanf(height, "%hhu", &heightnum) != 1)
56         return;
57     if(heightnum > LIFT_MAX || heightnum < LIFT_MIN)
58         return;
59     mode->dpi.lift = heightnum;
60 }
```

5.10.1.4 cmd_snap()

```
void cmd_snap (
    usbdevice * kb,
    usbmode * mode,
    int dummy1,
    int dummy2,
    const char * enable )
```

Definition at line 62 of file dpi.c.

References `usbmode::dpi`, and `dpiset::snap`.

```
62                                     {
63     if(!strcmp(enable, "on"))
64         mode->dpi.snap = 1;
65     if(!strcmp(enable, "off"))
66         mode->dpi.snap = 0;
67 }
```

5.10.1.5 loaddpi()

```
int loaddpi (
    usbdevice * kb,
    dpiset * dpi,
    lighting * light )
```

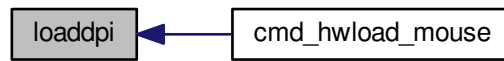
Definition at line 152 of file dpi.c.

References `lighting::b`, `ckb_err`, `dpiset::current`, `DPI_COUNT`, `dpiset::enabled`, `lighting::g`, `LED_MOUSE`, `dpiset::lift`, `LIFT_MAX`, `LIFT_MIN`, `MSG_SIZE`, `N_MOUSE_ZONES`, `lighting::r`, `dpiset::snap`, `usbrecv`, `dpiset::x`, and `dpiset::y`.

Referenced by `cmd_hwload_mouse()`.

```
152                                     {
153     // Ask for settings
154     uchar data_pkt[4][MSG_SIZE] = {
155         { 0x0e, 0x13, 0x05, 1, },
156         { 0x0e, 0x13, 0x02, 1, },
157         { 0x0e, 0x13, 0x03, 1, },
158         { 0x0e, 0x13, 0x04, 1, }
159     };
160     uchar in_pkt[4][MSG_SIZE];
161     for(int i = 0; i < 4; i++){
162         if(!usbrecv(kb, data_pkt[i], in_pkt[i]))
163             return -2;
164         if(memcmp(in_pkt[i], data_pkt[i], 4)){
165             ckb_err("Bad input header\n");
166             return -3;
167         }
168     }
169     // Copy data from device
170     dpi->enabled = in_pkt[0][4];
171     dpi->enabled &= (1 << DPI_COUNT) - 1;
172     dpi->current = in_pkt[1][4];
173     if(dpi->current >= DPI_COUNT)
174         dpi->current = 0;
175     dpi->lift = in_pkt[2][4];
176     if(dpi->lift < LIFT_MIN || dpi->lift > LIFT_MAX)
177         dpi->lift = LIFT_MIN;
178     dpi->snap = !!in_pkt[3][4];
179
180     // Get X/Y DPIs
181     for(int i = 0; i < DPI_COUNT; i++){
182         uchar data_pkt[MSG_SIZE] = { 0x0e, 0x13, 0xd0, 1 };
183         uchar in_pkt[MSG_SIZE];
184         data_pkt[2] |= i;
185         if(!usbrecv(kb, data_pkt, in_pkt))
186             return -2;
187         if(memcmp(in_pkt, data_pkt, 4)){
188             ckb_err("Bad input header\n");
189             return -3;
190         }
191         // Copy to profile
192         dpi->x[i] = *(ushort*)(in_pkt + 5);
193         dpi->y[i] = *(ushort*)(in_pkt + 7);
194         light->r[LED_MOUSE + N_MOUSE_ZONES + i] = in_pkt[9];
195         light->g[LED_MOUSE + N_MOUSE_ZONES + i] = in_pkt[10];
196         light->b[LED_MOUSE + N_MOUSE_ZONES + i] = in_pkt[11];
197     }
198     // Finished. Set SW DPI light to the current hardware level
199     light->r[LED_MOUSE + 2] = light->r[LED_MOUSE +
200     N_MOUSE_ZONES + dpi->current];
201     light->g[LED_MOUSE + 2] = light->g[LED_MOUSE +
202     N_MOUSE_ZONES + dpi->current];
203     light->b[LED_MOUSE + 2] = light->b[LED_MOUSE +
204     N_MOUSE_ZONES + dpi->current];
205     return 0;
206 }
```

Here is the caller graph for this function:



5.10.1.6 printdpi()

```
char* printdpi (
    const dpi_t * dpi,
    const usbdevice * kb )
```

Definition at line 69 of file dpi.c.

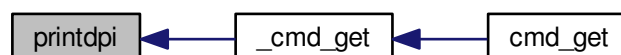
References `_readlines_ctx::buffer`, `DPI_COUNT`, `dpi_t::enabled`, `dpi_t::x`, and `dpi_t::y`.

Referenced by `_cmd_get()`.

```

69                                     {
70     // Print all DPI settings
71     const int BUFFER_LEN = 100;
72     char* buffer = malloc(BUFFER_LEN);
73     int length = 0;
74     for(int i = 0; i < DPI_COUNT; i++){
75         // Print the stage number
76         int newlen = 0;
77         snprintf(buffer + length, BUFFER_LEN - length, length == 0 ? "%d\n" : " %d\n", i, &newlen);
78         length += newlen;
79         // Print the DPI settings
80         if(!(dpi->enabled & (1 << i)))
81             snprintf(buffer + length, BUFFER_LEN - length, ":off\n", &newlen);
82         else
83             snprintf(buffer + length, BUFFER_LEN - length, ":%u,%u\n", dpi->x[i], dpi->
84 y[i], &newlen);
85         length += newlen;
86     }
87     return buffer;
  
```

Here is the caller graph for this function:



5.10.1.7 savedpi()

```
int savedpi (
    usbdevice * kb,
    dpiset * dpi,
    lighting * light )
```

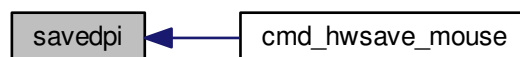
Definition at line 124 of file dpi.c.

References `lighting::b`, `dpiset::current`, `DPI_COUNT`, `dpiset::enabled`, `lighting::g`, `LED_MOUSE`, `dpiset::lift`, `MSG_SIZE`, `N_MOUSE_ZONES`, `lighting::r`, `dpiset::snap`, `usbdevice::profile`, `dpiset::x`, and `dpiset::y`.

Referenced by `cmd_hwsave_mouse()`.

```
124                                     {
125     // Send X/Y DPIs
126     for(int i = 0; i < DPI_COUNT; i++){
127         uchar data_pkt[MSG_SIZE] = { 0x07, 0x13, 0xd0, 1 };
128         data_pkt[2] |= i;
129         *(ushort*)(data_pkt + 5) = dpi->x[i];
130         *(ushort*)(data_pkt + 7) = dpi->y[i];
131         // Save the RGB value for this setting too
132         data_pkt[9] = light->r[LED_MOUSE + N_MOUSE_ZONES + i];
133         data_pkt[10] = light->g[LED_MOUSE + N_MOUSE_ZONES + i];
134         data_pkt[11] = light->b[LED_MOUSE + N_MOUSE_ZONES + i];
135         if(!usbdevice::profile)
136             return -1;
137     }
138
139     // Send settings
140     uchar data_pkt[4][MSG_SIZE] = {
141         { 0x07, 0x13, 0x05, 1, dpi->enabled },
142         { 0x07, 0x13, 0x02, 1, dpi->current },
143         { 0x07, 0x13, 0x03, 1, dpi->lift },
144         { 0x07, 0x13, 0x04, 1, dpi->snap, 0x05 }
145     };
146     if(!usbdevice::profile)
147         return -2;
148     // Finished
149     return 0;
150 }
```

Here is the caller graph for this function:



5.10.1.8 updatedpi()

```
int updatedpi (
    usbdevice * kb,
    int force )
```

Definition at line 89 of file dpi.c.

References `usbdevice::active`, `dpiset::current`, `usbprofile::currentmode`, `usbmode::dpi`, `DPI_COUNT`, `dpiset::enabled`, `dpiset::forceupdate`, `usbprofile::lastdpi`, `dpiset::lift`, `MSG_SIZE`, `usbdevice::profile`, `dpiset::snap`, `usbdevice::profile`, `dpiset::x`, and `dpiset::y`.

```

89                                     {
90     if(!kb->active)
91         return 0;
92     dpiset* lastdpi = &kb->profile->lastdpi;
93     dpiset* newdpi = &kb->profile->currentmode->dpi;
94     // Don't do anything if the settings haven't changed
95     if(!force && !lastdpi->forceupdate && !newdpi->forceupdate
96         && !memcmp(lastdpi, newdpi, sizeof(dpiset)))
97         return 0;
98     lastdpi->forceupdate = newdpi->forceupdate = 0;
99
100    // Send X/Y DPIs
101    for(int i = 0; i < DPI_COUNT; i++){
102        uchar data_pkt[MSG_SIZE] = { 0x07, 0x13, 0xd0, 0 };
103        data_pkt[2] |= i;
104        *(ushort*)(data_pkt + 5) = newdpi->x[i];
105        *(ushort*)(data_pkt + 7) = newdpi->y[i];
106        if(!usbSend(kb, data_pkt, 1))
107            return -1;
108    }
109
110    // Send settings
111    uchar data_pkt[4][MSG_SIZE] = {
112        { 0x07, 0x13, 0x05, 0, newdpi->enabled },
113        { 0x07, 0x13, 0x02, 0, newdpi->current },
114        { 0x07, 0x13, 0x03, 0, newdpi->lift },
115        { 0x07, 0x13, 0x04, 0, newdpi->snap, 0x05 }
116    };
117    if(!usbSend(kb, data_pkt[0], 4))
118        return -2;
119    // Finished
120    memcpy(lastdpi, newdpi, sizeof(dpiset));
121    return 0;
122 }

```

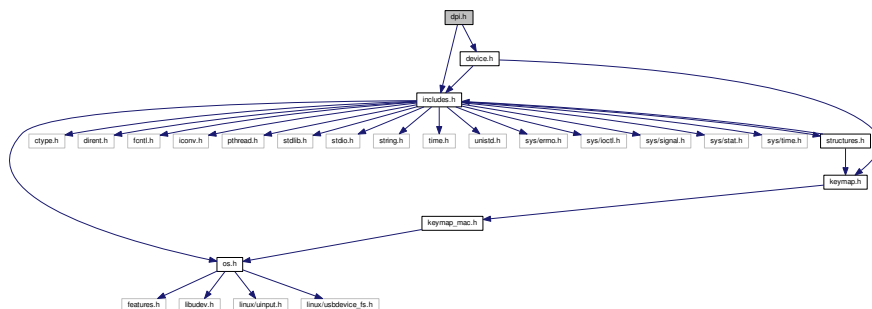
5.11 dpi.h File Reference

```

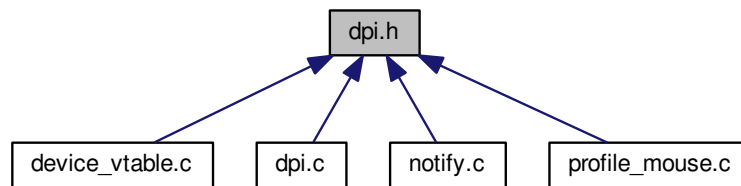
#include "includes.h"
#include "device.h"

```

Include dependency graph for dpi.h:



This graph shows which files directly or indirectly include this file:



Functions

- int `updatedpi` (`usbdevice` *kb, int force)
- int `savedpi` (`usbdevice` *kb, `dpiset` *dpi, `lighting` *light)
- int `loaddpi` (`usbdevice` *kb, `dpiset` *dpi, `lighting` *light)
- char * `printdpi` (const `dpiset` *dpi, const `usbdevice` *kb)
- void `cmd_dpi` (`usbdevice` *kb, `usbmode` *mode, int dummy, const char *stages, const char *values)
- void `cmd_dpiset` (`usbdevice` *kb, `usbmode` *mode, int dummy1, int dummy2, const char *stage)
- void `cmd_lift` (`usbdevice` *kb, `usbmode` *mode, int dummy1, int dummy2, const char *height)
- void `cmd_snap` (`usbdevice` *kb, `usbmode` *mode, int dummy1, int dummy2, const char *enable)

5.11.1 Function Documentation

5.11.1.1 cmd_dpi()

```

void cmd_dpi (
    usbdevice * kb,
    usbmode * mode,
    int dummy,
    const char * stages,
    const char * values )

```

Definition at line 4 of file dpi.c.

References `usbmode::dpi`, `DPI_COUNT`, `dpiset::enabled`, `dpiset::x`, and `dpiset::y`.

```

4
5     int disable = 0;
6     ushort x, y;
7     // Try to scan X,Y values
8     if(sscanf(values, "%hu,%hu", &x, &y) != 2){
9         // If that doesn't work, scan single number
10        if(sscanf(values, "%hu", &x) == 1)
11            y = x;
12        else if(!strcmp(values, "off", 3))
13            // If the right side says "off", disable the level(s)
14            disable = 1;
15        else
16            // Otherwise, quit

```

```

17         return;
18     }
19     if((x == 0 || y == 0) && !disable)
20         return;
21     // Scan the left side for stage numbers (comma-separated)
22     int left = strlen(stages);
23     int position = 0, field = 0;
24     char stagename[3];
25     while(position < left && sscanf(stages + position, "%2[^,]%n", stagename, &field) == 1){
26         uchar stagenum;
27         if(sscanf(stagename, "%hhu", &stagenum) && stagenum < DPI_COUNT){
28             // Set DPI for this stage
29             if(disable){
30                 mode->dpi.enabled &= ~(1 << stagenum);
31                 mode->dpi.x[stagenum] = 0;
32                 mode->dpi.y[stagenum] = 0;
33             } else {
34                 mode->dpi.enabled |= 1 << stagenum;
35                 mode->dpi.x[stagenum] = x;
36                 mode->dpi.y[stagenum] = y;
37             }
38         }
39         if(stages[position += field] == ',')
40             position++;
41     }
42 }

```

5.11.1.2 cmd_dpisel()

```

void cmd_dpisel (
    usbdevice * kb,
    usbmode * mode,
    int dummy1,
    int dummy2,
    const char * stage )

```

Definition at line 44 of file dpi.c.

References `dpiset::current`, `usbmode::dpi`, and `DPI_COUNT`.

```

44                                     {
45         uchar stagenum;
46         if(sscanf(stage, "%hhu", &stagenum) != 1)
47             return;
48         if(stagenum > DPI_COUNT)
49             return;
50         mode->dpi.current = stagenum;
51     }

```

5.11.1.3 cmd_lift()

```

void cmd_lift (
    usbdevice * kb,
    usbmode * mode,
    int dummy1,
    int dummy2,
    const char * height )

```

Definition at line 53 of file dpi.c.

References `usbmode::dpi`, `dpiset::lift`, `LIFT_MAX`, and `LIFT_MIN`.

```

53                                     {
54     uchar heightnum;
55     if(sscanf(height, "%hhu", &heightnum) != 1)
56         return;
57     if(heightnum > LIFT_MAX || heightnum < LIFT_MIN)
58         return;
59     mode->dpi.lift = heightnum;
60 }

```

5.11.1.4 cmd_snap()

```

void cmd_snap (
    usbdevice * kb,
    usbmode * mode,
    int dummy1,
    int dummy2,
    const char * enable )

```

Definition at line 62 of file dpi.c.

References `usbmode::dpi`, and `dpiset::snap`.

```

62                                     {
63     if(!strcmp(enable, "on"))
64         mode->dpi.snap = 1;
65     if(!strcmp(enable, "off"))
66         mode->dpi.snap = 0;
67 }

```

5.11.1.5 loaddpi()

```

int loaddpi (
    usbdevice * kb,
    dpiset * dpi,
    lighting * light )

```

Definition at line 152 of file dpi.c.

References `lighting::b`, `ckb_err`, `dpiset::current`, `DPI_COUNT`, `dpiset::enabled`, `lighting::g`, `LED_MOUSE`, `dpiset::lift`, `LIFT_MAX`, `LIFT_MIN`, `MSG_SIZE`, `N_MOUSE_ZONES`, `lighting::r`, `dpiset::snap`, `usbrecv`, `dpiset::x`, and `dpiset::y`.

Referenced by `cmd_hwload_mouse()`.

```

152                                     {
153     // Ask for settings
154     uchar data_pkt[4][MSG_SIZE] = {
155         { 0x0e, 0x13, 0x05, 1, },
156         { 0x0e, 0x13, 0x02, 1, },
157         { 0x0e, 0x13, 0x03, 1, },
158         { 0x0e, 0x13, 0x04, 1, }
159     };
160     uchar in_pkt[4][MSG_SIZE];
161     for(int i = 0; i < 4; i++){
162         if(!usbrecv(kb, data_pkt[i], in_pkt[i]))
163             return -2;
164         if(memcmp(in_pkt[i], data_pkt[i], 4)){
165             ckb_err("Bad input header\n");

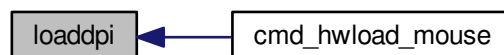
```

```

166         return -3;
167     }
168 }
169 // Copy data from device
170 dpi->enabled = in_pkt[0][4];
171 dpi->enabled &= (1 << DPI_COUNT) - 1;
172 dpi->current = in_pkt[1][4];
173 if(dpi->current >= DPI_COUNT)
174     dpi->current = 0;
175 dpi->lift = in_pkt[2][4];
176 if(dpi->lift < LIFT_MIN || dpi->lift > LIFT_MAX)
177     dpi->lift = LIFT_MIN;
178 dpi->snap = !!in_pkt[3][4];
179
180 // Get X/Y DPIs
181 for(int i = 0; i < DPI_COUNT; i++){
182     uchar data_pkt[MSG_SIZE] = { 0x0e, 0x13, 0xd0, 1 };
183     uchar in_pkt[MSG_SIZE];
184     data_pkt[2] |= i;
185     if(!usbrecv(kb, data_pkt, in_pkt))
186         return -2;
187     if(memcmp(in_pkt, data_pkt, 4)){
188         ckb_errr("Bad input header\n");
189         return -3;
190     }
191     // Copy to profile
192     dpi->x[i] = *(ushort*)(in_pkt + 5);
193     dpi->y[i] = *(ushort*)(in_pkt + 7);
194     light->r[LED_MOUSE + N_MOUSE_ZONES + i] = in_pkt[9];
195     light->g[LED_MOUSE + N_MOUSE_ZONES + i] = in_pkt[10];
196     light->b[LED_MOUSE + N_MOUSE_ZONES + i] = in_pkt[11];
197 }
198 // Finished. Set SW DPI light to the current hardware level
199 light->r[LED_MOUSE + 2] = light->r[LED_MOUSE +
N_MOUSE_ZONES + dpi->current];
200 light->g[LED_MOUSE + 2] = light->g[LED_MOUSE +
N_MOUSE_ZONES + dpi->current];
201 light->b[LED_MOUSE + 2] = light->b[LED_MOUSE +
N_MOUSE_ZONES + dpi->current];
202 return 0;
203 }

```

Here is the caller graph for this function:



5.11.1.6 printdpi()

```

char* printdpi (
    const dpiset * dpi,
    const usbdevice * kb )

```

Definition at line 69 of file dpi.c.

References `_readlines_ctx::buffer`, `DPI_COUNT`, `dpiset::enabled`, `dpiset::x`, and `dpiset::y`.

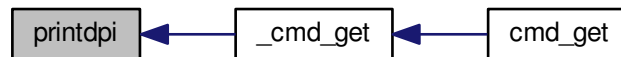
Referenced by `_cmd_get()`.

```

69                                     {
70     // Print all DPI settings
71     const int BUFFER_LEN = 100;
72     char* buffer = malloc(BUFFER_LEN);
73     int length = 0;
74     for(int i = 0; i < DPI_COUNT; i++){
75         // Print the stage number
76         int newlen = 0;
77         snprintf(buffer + length, BUFFER_LEN - length, length == 0 ? "%d\n" : " %d\n", i, &newlen);
78         length += newlen;
79         // Print the DPI settings
80         if(!(dpi->enabled & (1 << i)))
81             snprintf(buffer + length, BUFFER_LEN - length, ":off\n", &newlen);
82         else
83             snprintf(buffer + length, BUFFER_LEN - length, ":%u,%u\n", dpi->x[i], dpi->
y[i], &newlen);
84         length += newlen;
85     }
86     return buffer;
87 }

```

Here is the caller graph for this function:



5.11.1.7 savedpi()

```

int savedpi (
    usbdevice * kb,
    dpiset * dpi,
    lighting * light )

```

Definition at line 124 of file dpi.c.

References `lighting::b`, `dpiset::current`, `DPI_COUNT`, `dpiset::enabled`, `lighting::g`, `LED_MOUSE`, `dpiset::lift`, `MSG_SIZE`, `N_MOUSE_ZONES`, `lighting::r`, `dpiset::snap`, `usbdevice`, `dpiset::x`, and `dpiset::y`.

Referenced by `cmd_hwsave_mouse()`.

```

124                                     {
125     // Send X/Y DPIs
126     for(int i = 0; i < DPI_COUNT; i++){
127         uchar data_pkt[MSG_SIZE] = { 0x07, 0x13, 0xd0, 1 };
128         data_pkt[2] |= i;
129         *(ushort*)(data_pkt + 5) = dpi->x[i];
130         *(ushort*)(data_pkt + 7) = dpi->y[i];
131         // Save the RGB value for this setting too
132         data_pkt[9] = light->r[LED_MOUSE + N_MOUSE_ZONES + i];
133         data_pkt[10] = light->g[LED_MOUSE + N_MOUSE_ZONES + i];
134         data_pkt[11] = light->b[LED_MOUSE + N_MOUSE_ZONES + i];
135         if(!usbdevice(kb, data_pkt, 1))
136             return -1;
137     }
138
139     // Send settings
140     uchar data_pkt[4][MSG_SIZE] = {

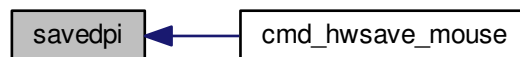
```

```

141         { 0x07, 0x13, 0x05, 1, dpi->enabled },
142         { 0x07, 0x13, 0x02, 1, dpi->current },
143         { 0x07, 0x13, 0x03, 1, dpi->lift },
144         { 0x07, 0x13, 0x04, 1, dpi->snap, 0x05 }
145     };
146     if(!usbsend(kb, data_pkt[0], 4))
147         return -2;
148     // Finished
149     return 0;
150 }

```

Here is the caller graph for this function:



5.11.1.8 updatedpi()

```

int updatedpi (
    usbdevice * kb,
    int force )

```

Definition at line 89 of file dpi.c.

References `usbdevice::active`, `dpiset::current`, `usbprofile::currentmode`, `usbmode::dpi`, `DPI_COUNT`, `dpiset::enabled`, `dpiset::forceupdate`, `usbprofile::lastdpi`, `dpiset::lift`, `MSG_SIZE`, `usbdevice::profile`, `dpiset::snap`, `usbdevice::send`, `dpiset::x`, and `dpiset::y`.

```

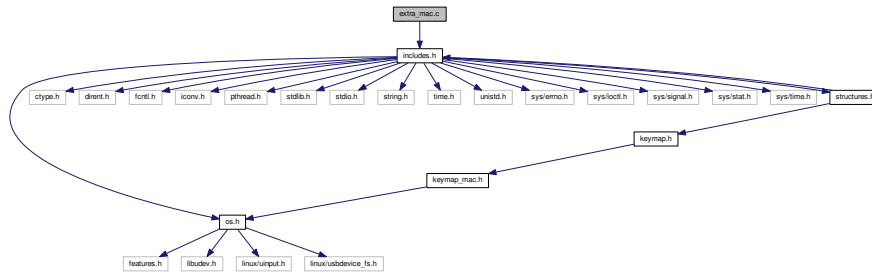
89         {
90             if(!kb->active)
91                 return 0;
92             dpiset* lastdpi = &kb->profile->lastdpi;
93             dpiset* newdpi = &kb->profile->currentmode->dpi;
94             // Don't do anything if the settings haven't changed
95             if(!force && !lastdpi->forceupdate && !newdpi->forceupdate
96                 && !memcmp(lastdpi, newdpi, sizeof(dpi)))
97                 return 0;
98             lastdpi->forceupdate = newdpi->forceupdate = 0;
99
100             // Send X/Y DPIs
101             for(int i = 0; i < DPI_COUNT; i++){
102                 uchar data_pkt[MSG_SIZE] = { 0x07, 0x13, 0x00, 0 };
103                 data_pkt[2] |= i;
104                 *(ushort*)(data_pkt + 5) = newdpi->x[i];
105                 *(ushort*)(data_pkt + 7) = newdpi->y[i];
106                 if(!usbdevice::send(kb, data_pkt, 1))
107                     return -1;
108             }
109
110             // Send settings
111             uchar data_pkt[4][MSG_SIZE] = {
112                 { 0x07, 0x13, 0x05, 0, newdpi->enabled },
113                 { 0x07, 0x13, 0x02, 0, newdpi->current },
114                 { 0x07, 0x13, 0x03, 0, newdpi->lift },
115                 { 0x07, 0x13, 0x04, 0, newdpi->snap, 0x05 }
116             };
117             if(!usbdevice::send(kb, data_pkt[0], 4))
118                 return -2;
119             // Finished
120             memcpy(lastdpi, newdpi, sizeof(dpi));
121             return 0;
122         }

```

5.12 extra_mac.c File Reference

```
#include "includes.h"
```

Include dependency graph for extra_mac.c:



5.13 firmware.c File Reference

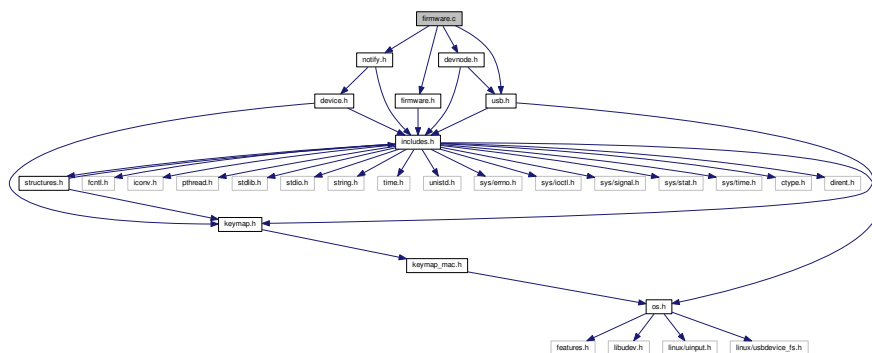
```
#include "devnode.h"
```

```
#include "firmware.h"
```

```
#include "notify.h"
```

```
#include "usb.h"
```

Include dependency graph for firmware.c:



Macros

- `#define FW_OK 0`
- `#define FW_NOFILE -1`
- `#define FW_WRONGDEV -2`
- `#define FW_USBFAIL -3`
- `#define FW_MAXSIZE (255 * 256)`

Functions

- `int getfwversion (usbdevice *kb)`
- `int fwupdate (usbdevice *kb, const char *path, int nnumber)`
- `int cmd_fwupdate (usbdevice *kb, usbmode *dummy1, int nnumber, int dummy2, const char *path)`

5.13.1 Macro Definition Documentation

5.13.1.1 FW_MAXSIZE

```
#define FW_MAXSIZE (255 * 256)
```

Definition at line 51 of file firmware.c.

Referenced by fwupdate().

5.13.1.2 FW_NOFILE

```
#define FW_NOFILE -1
```

Definition at line 7 of file firmware.c.

Referenced by cmd_fwupdate(), and fwupdate().

5.13.1.3 FW_OK

```
#define FW_OK 0
```

Definition at line 6 of file firmware.c.

Referenced by cmd_fwupdate(), and fwupdate().

5.13.1.4 FW_USBFAIL

```
#define FW_USBFAIL -3
```

Definition at line 9 of file firmware.c.

Referenced by cmd_fwupdate(), and fwupdate().

5.13.1.5 FW_WRONGDEV

```
#define FW_WRONGDEV -2
```

Definition at line 8 of file firmware.c.

Referenced by cmd_fwupdate(), and fwupdate().

5.13.2 Function Documentation

5.13.2.1 cmd_fwupdate()

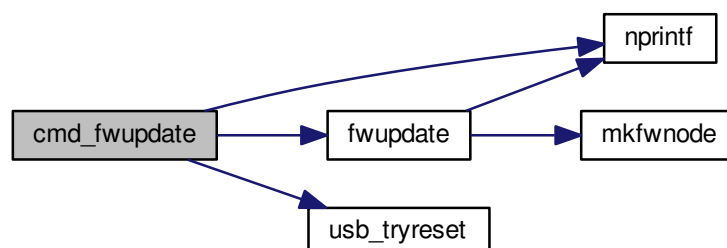
```
int cmd_fwupdate (
    usbdevice * kb,
    usbmode * dummy1,
    int nnumber,
    int dummy2,
    const char * path )
```

Definition at line 154 of file firmware.c.

References `FEAT_FWUPDATE`, `FW_NOFILE`, `FW_OK`, `FW_USBFAIL`, `FW_WRONGDEV`, `fwupdate()`, `HAS_FEATURES`, `nprintf()`, and `usb_tryreset()`.

```
154
155     if(!HAS_FEATURES(kb, FEAT_FWUPDATE))
156         return 0;
157     // Update the firmware
158     int ret = fwupdate(kb, path, nnumber);
159     while(ret == FW_USBFAIL){
160         // Try to reset the device if it fails
161         if(usb_tryreset(kb))
162             break;
163         ret = fwupdate(kb, path, nnumber);
164     }
165     switch(ret){
166     case FW_OK:
167         nprintf(kb, nnumber, 0, "fwupdate %s ok\n", path);
168         break;
169     case FW_NOFILE:
170     case FW_WRONGDEV:
171         nprintf(kb, nnumber, 0, "fwupdate %s invalid\n", path);
172         break;
173     case FW_USBFAIL:
174         nprintf(kb, nnumber, 0, "fwupdate %s fail\n", path);
175         return -1;
176     }
177     return 0;
178 }
```

Here is the call graph for this function:

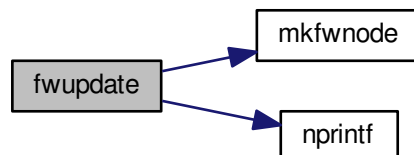



```

116         if(index == 1){
117             if(!usbSend(kb, data_pkt[0], 1)){
118                 ckb_err("Firmware update failed\n");
119                 return FW_USBFAIL;
120             }
121             // The above packet can take a lot longer to process, so wait for a while
122             sleep(3);
123             if(!usbSend(kb, data_pkt[2], npackets - 1)){
124                 ckb_err("Firmware update failed\n");
125                 return FW_USBFAIL;
126             }
127         } else {
128             // If the output ends here, set the length byte appropriately
129             if(output >= length)
130                 data_pkt[npackets][2] = length - last;
131             if(!usbSend(kb, data_pkt[1], npackets)){
132                 ckb_err("Firmware update failed\n");
133                 return FW_USBFAIL;
134             }
135         }
136         nprintf(kb, nnumber, 0, "fwupdate %s %d/%d\n", path, output, (int)length);
137     }
138     // Send the final pair of messages
139     uchar data_pkt2[2][MSG_SIZE] = {
140         { 0x07, 0x0d, 0xf0, 0x00, 0x00, 0x00, index },
141         { 0x07, 0x02, 0xf0, 0 }
142     };
143     if(!usbSend(kb, data_pkt2[0], 2)){
144         ckb_err("Firmware update failed\n");
145         return FW_USBFAIL;
146     }
147     // Updated successfully
148     kb->fwversion = version;
149     mkfwnode(kb);
150     ckb_info("Firmware update complete\n");
151     return FW_OK;
152 }

```

Here is the call graph for this function:



Here is the caller graph for this function:



5.13.2.3 getfwversion()

```
int getfwversion (
    usbdevice * kb )
```

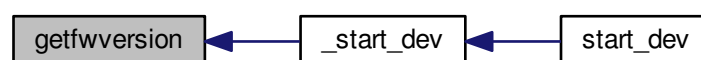
Definition at line 11 of file firmware.c.

References `ckb_err`, `ckb_warn`, `FEAT_POLLRATE`, `usbdevice::features`, `usbdevice::fwversion`, `MSG_SIZE`, `usbdevice::pollrate`, `usbdevice::product`, `usbdevc`, and `usbdevice::vendor`.

Referenced by `_start_dev()`.

```
11         {
12     // Ask board for firmware info
13     uchar data_pkt[MSG_SIZE] = { 0x0e, 0x01, 0 };
14     uchar in_pkt[MSG_SIZE];
15     if(!usbrecv(kb, data_pkt, in_pkt))
16         return -1;
17     if(in_pkt[0] != 0x0e || in_pkt[1] != 0x01){
18         ckb_err("Bad input header\n");
19         return -1;
20     }
21     short vendor, product, version, bootloader;
22     // Copy the vendor ID, product ID, version, and poll rate from the firmware data
23     memcpy(&version, in_pkt + 8, 2);
24     memcpy(&bootloader, in_pkt + 10, 2);
25     memcpy(&vendor, in_pkt + 12, 2);
26     memcpy(&product, in_pkt + 14, 2);
27     char poll = in_pkt[16];
28     if(poll <= 0){
29         poll = -1;
30         kb->features &= ~FEAT_POLLRATE;
31     }
32     // Print a warning if the message didn't match the expected data
33     if(vendor != kb->vendor)
34         ckb_warn("Got vendor ID %04x (expected %04x)\n", vendor, kb->
vendor);
35     if(product != kb->product)
36         ckb_warn("Got product ID %04x (expected %04x)\n", product, kb->
product);
37     // Set firmware version and poll rate
38     if(version == 0 || bootloader == 0){
39         // Needs firmware update
40         kb->fwversion = 0;
41         kb->pollrate = -1;
42     } else {
43         if(version != kb->fwversion && kb->fwversion != 0)
44             ckb_warn("Got firmware version %04x (expected %04x)\n", version, kb->
fwversion);
45         kb->fwversion = version;
46         kb->pollrate = poll;
47     }
48     return 0;
49 }
```

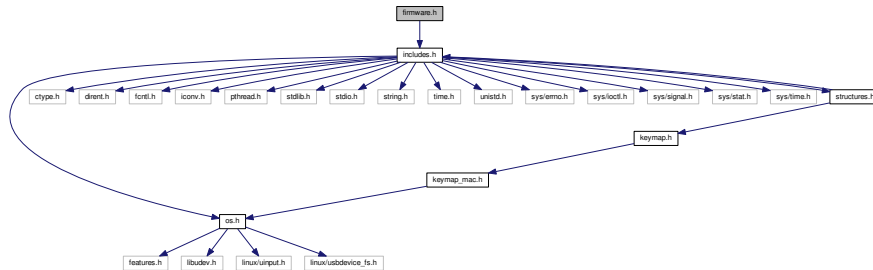
Here is the caller graph for this function:



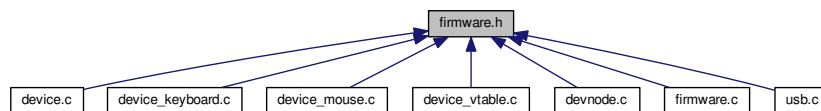
5.14 firmware.h File Reference

```
#include "includes.h"
```

Include dependency graph for firmware.h:



This graph shows which files directly or indirectly include this file:



Functions

- `int getfwversion (usbdevice *kb)`
- `int cmd_fwupdate (usbdevice *kb, usbmode *dummy1, int nnumber, int dummy2, const char *path)`

5.14.1 Function Documentation

5.14.1.1 cmd_fwupdate()

```
int cmd_fwupdate (
    usbdevice * kb,
    usbmode * dummy1,
    int nnumber,
    int dummy2,
    const char * path )
```

Definition at line 154 of file `firmware.c`.

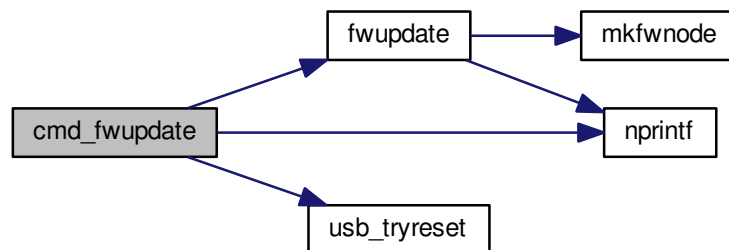
References `FEAT_FWUPDATE`, `FW_NOFILE`, `FW_OK`, `FW_USBFAIL`, `FW_WRONGDEV`, `fwupdate()`, `HAS_FEATURES`, `nprintf()`, and `usb_tryreset()`.

```

154                                     {
155     if(!HAS_FEATURES(kb, FEAT_FWUPDATE))
156         return 0;
157     // Update the firmware
158     int ret = fwupdate(kb, path, nnumber);
159     while(ret == FW_USBFAIL){
160         // Try to reset the device if it fails
161         if(usb_tryreset(kb))
162             break;
163         ret = fwupdate(kb, path, nnumber);
164     }
165     switch(ret){
166     case FW_OK:
167         nprintf(kb, nnumber, 0, "fwupdate %s ok\n", path);
168         break;
169     case FW_NOFILE:
170     case FW_WRONGDEV:
171         nprintf(kb, nnumber, 0, "fwupdate %s invalid\n", path);
172         break;
173     case FW_USBFAIL:
174         nprintf(kb, nnumber, 0, "fwupdate %s fail\n", path);
175         return -1;
176     }
177     return 0;
178 }

```

Here is the call graph for this function:



5.14.1.2 getfwversion()

```

int getfwversion (
    usbdevice * kb )

```

Definition at line 11 of file firmware.c.

References `ckb_err`, `ckb_warn`, `FEAT_POLLRATE`, `usbdevice::features`, `usbdevice::fwversion`, `MSG_SIZE`, `usbdevice::pollrate`, `usbdevice::product`, `usbrecv`, and `usbdevice::vendor`.

Referenced by `_start_dev()`.

```

11                                     {
12     // Ask board for firmware info
13     uchar data_pkt[MSG_SIZE] = { 0x0e, 0x01, 0 };
14     uchar in_pkt[MSG_SIZE];
15     if(!usbrecv(kb, data_pkt, in_pkt))
16         return -1;

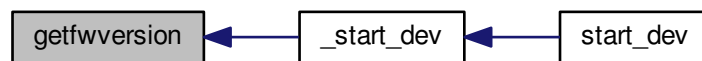
```

```

17     if(in_pkt[0] != 0x0e || in_pkt[1] != 0x01){
18         ckb_err("Bad input header\n");
19         return -1;
20     }
21     short vendor, product, version, bootloader;
22     // Copy the vendor ID, product ID, version, and poll rate from the firmware data
23     memcpy(&version, in_pkt + 8, 2);
24     memcpy(&bootloader, in_pkt + 10, 2);
25     memcpy(&vendor, in_pkt + 12, 2);
26     memcpy(&product, in_pkt + 14, 2);
27     char poll = in_pkt[16];
28     if(poll <= 0){
29         poll = -1;
30         kb->features &= ~FEAT_POLLRATE;
31     }
32     // Print a warning if the message didn't match the expected data
33     if(vendor != kb->vendor)
34         ckb_warn("Got vendor ID %04x (expected %04x)\n", vendor, kb->
vendor);
35     if(product != kb->product)
36         ckb_warn("Got product ID %04x (expected %04x)\n", product, kb->
product);
37     // Set firmware version and poll rate
38     if(version == 0 || bootloader == 0){
39         // Needs firmware update
40         kb->fwversion = 0;
41         kb->pollrate = -1;
42     } else {
43         if(version != kb->fwversion && kb->fwversion != 0)
44             ckb_warn("Got firmware version %04x (expected %04x)\n", version, kb->
fwversion);
45         kb->fwversion = version;
46         kb->pollrate = poll;
47     }
48     return 0;
49 }

```

Here is the caller graph for this function:



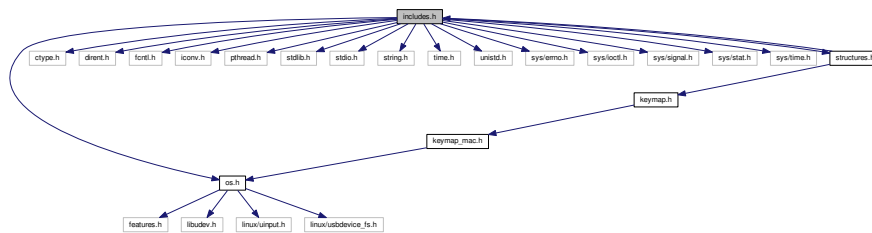
5.15 includes.h File Reference

```

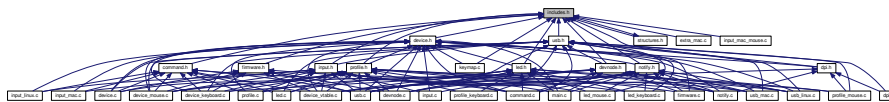
#include "os.h"
#include <ctype.h>
#include <dirent.h>
#include <fcntl.h>
#include <iconv.h>
#include <pthread.h>
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <time.h>
#include <unistd.h>
#include <sys/errno.h>
#include <sys/ioctl.h>
#include <sys/signal.h>
#include <sys/stat.h>

```

```
#include <sys/time.h>
#include "structures.h"
Include dependency graph for includes.h:
```



This graph shows which files directly or indirectly include this file:



Macros

- #define **INDEX_OF**(entry, array) (int)(entry - array)
- #define **ckb_s_out** stdout
- #define **ckb_s_err** stdout
- #define **__FILE_NOPATH__** (strchr(__FILE__, '/') ? strchr(__FILE__, '/') + 1 : __FILE__)
- #define **ckb_fatal_nofile**(fmt, args...) fprintf(**ckb_s_err**, "[F] " fmt, ## args)
- #define **ckb_fatal_fn**(fmt, file, line, args...) fprintf(**ckb_s_err**, "[F] %s (via %s:%d): " fmt, __func__, file, line, ## args)
- #define **ckb_fatal**(fmt, args...) fprintf(**ckb_s_err**, "[F] %s (%s:%d): " fmt, __func__, **__FILE_NOPATH__**, **__LINE__**, ## args)
- #define **ckb_err_nofile**(fmt, args...) fprintf(**ckb_s_err**, "[E] " fmt, ## args)
- #define **ckb_err_fn**(fmt, file, line, args...) fprintf(**ckb_s_err**, "[E] %s (via %s:%d): " fmt, __func__, file, line, ## args)
- #define **ckb_err**(fmt, args...) fprintf(**ckb_s_err**, "[E] %s (%s:%d): " fmt, __func__, **__FILE_NOPATH__**, **__LINE__**, ## args)
- #define **ckb_warn_nofile**(fmt, args...) fprintf(**ckb_s_out**, "[W] " fmt, ## args)
- #define **ckb_warn_fn**(fmt, file, line, args...) fprintf(**ckb_s_out**, "[W] %s (via %s:%d): " fmt, __func__, file, line, ## args)
- #define **ckb_warn**(fmt, args...) fprintf(**ckb_s_out**, "[W] %s (%s:%d): " fmt, __func__, **__FILE_NOPATH__**, **__LINE__**, ## args)
- #define **ckb_info_nofile**(fmt, args...) fprintf(**ckb_s_out**, "[I] " fmt, ## args)
- #define **ckb_info_fn**(fmt, file, line, args...) fprintf(**ckb_s_out**, "[I] " fmt, ## args)
- #define **ckb_info**(fmt, args...) fprintf(**ckb_s_out**, "[I] " fmt, ## args)
- #define **timespec_gt**(left, right) ((left).tv_sec > (right).tv_sec || ((left).tv_sec == (right).tv_sec && (left).tv_nsec > (right).tv_nsec))
- #define **timespec_eq**(left, right) ((left).tv_sec == (right).tv_sec && (left).tv_nsec == (right).tv_nsec)
- #define **timespec_ge**(left, right) ((left).tv_sec > (right).tv_sec || ((left).tv_sec == (right).tv_sec && (left).tv_nsec >= (right).tv_nsec))
- #define **timespec_lt**(left, right) (!**timespec_ge**(left, right))
- #define **timespec_le**(left, right) (!**timespec_gt**(left, right))

Typedefs

- typedef unsigned char [uchar](#)
- typedef unsigned short [ushort](#)

Functions

- void [timespec_add](#) (struct timespec *timespec, long nanoseconds)

5.15.1 Macro Definition Documentation

5.15.1.1 __FILE_NOPATH__

```
#define __FILE_NOPATH__ (strchr(__FILE__, '/') ? strchr(__FILE__, '/') + 1 : __FILE__)
```

Definition at line 40 of file includes.h.

5.15.1.2 ckb_err

```
#define ckb_err(  
    fmt,  
    args... ) fprintf(ckb_s_err, "[E] %s (%s:%d): " fmt, __func__, __FILE_NOPATH__  
_, __LINE__, ## args)
```

Definition at line 49 of file includes.h.

Referenced by [_mkdevpath\(\)](#), [fwupdate\(\)](#), [getfwversion\(\)](#), [loaddpi\(\)](#), [loadrgb_kb\(\)](#), [loadrgb_mouse\(\)](#), [os_sendindicators\(\)](#), [os_setupusb\(\)](#), [restart\(\)](#), [setupusb\(\)](#), [uinputopen\(\)](#), [usb_tryreset\(\)](#), [usbadd\(\)](#), and [usbclaim\(\)](#).

5.15.1.3 ckb_err_fn

```
#define ckb_err_fn(  
    fmt,  
    file,  
    line,  
    args... ) fprintf(ckb_s_err, "[E] %s (via %s:%d): " fmt, __func__, file, line,  
## args)
```

Definition at line 48 of file includes.h.

Referenced by [_nk95cmd\(\)](#), [_usbrecv\(\)](#), [os_usbrecv\(\)](#), and [os_usbsend\(\)](#).

5.15.1.4 ckb_err_nofile

```
#define ckb_err_nofile(  
    fmt,  
    args... ) fprintf(ckb_s_err, "[E] " fmt, ## args)
```

Definition at line 47 of file includes.h.

5.15.1.5 ckb_fatal

```
#define ckb_fatal(  
    fmt,  
    args... ) fprintf(ckb_s_err, "[F] %s (%s:%d): " fmt, __func__, __FILE__  
    , __LINE__, ## args)
```

Definition at line 46 of file includes.h.

Referenced by usbmain().

5.15.1.6 ckb_fatal_fn

```
#define ckb_fatal_fn(  
    fmt,  
    file,  
    line,  
    args... ) fprintf(ckb_s_err, "[F] %s (via %s:%d): " fmt, __func__, file, line,  
    ## args)
```

Definition at line 45 of file includes.h.

5.15.1.7 ckb_fatal_nofile

```
#define ckb_fatal_nofile(  
    fmt,  
    args... ) fprintf(ckb_s_err, "[F] " fmt, ## args)
```

Definition at line 44 of file includes.h.

Referenced by main().

5.15.1.8 ckb_info

```
#define ckb_info(  
    fmt,  
    args... ) fprintf(ckb_s_out, "[I] " fmt, ## args)
```

Definition at line 55 of file includes.h.

Referenced by `_setupusb()`, `_start_dev()`, `closeusb()`, `cmd_restart()`, `fwupdate()`, `main()`, `os_inputmain()`, `os_↵
setupusb()`, `quitWithLock()`, `rmdevpath()`, and `usb_tryreset()`.

5.15.1.9 ckb_info_fn

```
#define ckb_info_fn(  
    fmt,  
    file,  
    line,  
    args... ) fprintf(ckb_s_out, "[I] " fmt, ## args)
```

Definition at line 54 of file includes.h.

5.15.1.10 ckb_info_nofile

```
#define ckb_info_nofile(  
    fmt,  
    args... ) fprintf(ckb_s_out, "[I] " fmt, ## args)
```

Definition at line 53 of file includes.h.

Referenced by `main()`.

5.15.1.11 ckb_s_err

```
#define ckb_s_err stdout
```

Definition at line 36 of file includes.h.

5.15.1.12 ckb_s_out

```
#define ckb_s_out stdout
```

Definition at line 35 of file includes.h.

5.15.1.13 ckb_warn

```
#define ckb_warn(  
    fmt,  
    args... ) fprintf(ckb_s_out, "[W] %s (%s:%d): " fmt, __func__, __FILE__↵  
_, __LINE__, ## args)
```

Definition at line 52 of file includes.h.

Referenced by `_ledthread()`, `_mkdevpath()`, `_mknotifynode()`, `_start_dev()`, `_updateconnected()`, `devmain()`, `getfwversion()`, `hid_kb_translate()`, `isync()`, `mkfwnode()`, `os_inputclose()`, `os_inputmain()`, `os_keypress()`, `os_↵mousemove()`, `os_resetusb()`, `os_setupusb()`, `readlines()`, `rmdevpath()`, `uinputopen()`, `usbmain()`, and `usbunclaim()`.

5.15.1.14 ckb_warn_fn

```
#define ckb_warn_fn(  
    fmt,  
    file,  
    line,  
    args... ) fprintf(ckb_s_out, "[W] %s (via %s:%d): " fmt, __func__, file, line,  
## args)
```

Definition at line 51 of file includes.h.

Referenced by `os_usbrecv()`, and `os_usbsend()`.

5.15.1.15 ckb_warn_nofile

```
#define ckb_warn_nofile(  
    fmt,  
    args... ) fprintf(ckb_s_out, "[W] " fmt, ## args)
```

Definition at line 50 of file includes.h.

Referenced by `main()`.

5.15.1.16 INDEX_OF

```
#define INDEX_OF(  
    entry,  
    array ) (int)(entry - array)
```

Definition at line 27 of file includes.h.

Referenced by `_mkdevpath()`, `_mknotifynode()`, `_rmnotifynode()`, `_setupusb()`, `closeusb()`, `mkfwnode()`, `nprintf()`, `os_closeusb()`, `os_inputmain()`, `os_inputopen()`, `os_setupusb()`, `readcmd()`, and `rmdevpath()`.

5.15.1.17 timespec_eq

```
#define timespec_eq(  
    left,  
    right ) ((left).tv_sec == (right).tv_sec && (left).tv_nsec == (right).tv_nsec)
```

Definition at line 60 of file includes.h.

5.15.1.18 timespec_ge

```
#define timespec_ge(  
    left,  
    right ) ((left).tv_sec > (right).tv_sec || ((left).tv_sec == (right).tv_sec &&  
(left).tv_nsec >= (right).tv_nsec))
```

Definition at line 61 of file includes.h.

5.15.1.19 timespec_gt

```
#define timespec_gt(  
    left,  
    right ) ((left).tv_sec > (right).tv_sec || ((left).tv_sec == (right).tv_sec &&  
(left).tv_nsec > (right).tv_nsec))
```

Definition at line 59 of file includes.h.

5.15.1.20 timespec_le

```
#define timespec_le(  
    left,  
    right ) (!timespec_gt(left, right))
```

Definition at line 63 of file includes.h.

5.15.1.21 timespec_lt

```
#define timespec_lt(  
    left,  
    right ) (!timespec_ge(left, right))
```

Definition at line 62 of file includes.h.

Macros

- `#define IS_WHEEL(scan, kb) (((scan) == KEY_VOLUMEUP || (scan) == KEY_VOLUMEDOWN || (scan) == BTN_WHEELUP || (scan) == BTN_WHEELDOWN) && !IS_K65(kb))`

Functions

- `int macromask (const uchar *key1, const uchar *key2)`
- `static void inputupdate_keys (usbdevice *kb)`
- `void inputupdate (usbdevice *kb)`
- `void updateindicators_kb (usbdevice *kb, int force)`
- `void initbind (binding *bind)`
- `void freebind (binding *bind)`
- `void cmd_bind (usbdevice *kb, usbmode *mode, int dummy, int keyindex, const char *to)`
- `void cmd_unbind (usbdevice *kb, usbmode *mode, int dummy, int keyindex, const char *to)`
- `void cmd_rebind (usbdevice *kb, usbmode *mode, int dummy, int keyindex, const char *to)`
- `static void _cmd_macro (usbmode *mode, const char *keys, const char *assignment)`
- `void cmd_macro (usbdevice *kb, usbmode *mode, const int notifynumber, const char *keys, const char *assignment)`

5.16.1 Macro Definition Documentation

5.16.1.1 IS_WHEEL

```
#define IS_WHEEL(  
    scan,  
    kb ) (((scan) == KEY_VOLUMEUP || (scan) == KEY_VOLUMEDOWN || (scan) == BTN_WHEELUP  
|| (scan) == BTN_WHEELDOWN) && !IS_K65(kb))
```

Referenced by `inputupdate_keys()`.

5.16.2 Function Documentation

5.16.2.1 _cmd_macro()

```
static void _cmd_macro (
    usbmode * mode,
    const char * keys,
    const char * assignment ) [static]
```

Definition at line 226 of file input.c.

References `keymacro::actioncount`, `keymacro::actions`, `usbmode::bind`, `keymacro::combo`, `macroaction::down`, `keymap`, `MACRO_MAX`, `binding::macrocap`, `binding::macrocount`, `binding::macros`, `N_KEYBYTES_INPUT`, `N_KEYS_INPUT`, `macroaction::scan`, `key::scan`, and `SET_KEYBIT`.

Referenced by `cmd_macro()`.

```
226
227     binding* bind = &mode->bind;
228     if(!keys && !assignment){
229         // Null strings = "macro clear" -> erase the whole thing
230         for(int i = 0; i < bind->macrocount; i++)
231             free(bind->macros[i].actions);
232         bind->macrocount = 0;
233         return;
234     }
235     if(bind->macrocount >= MACRO_MAX)
236         return;
237     // Create a key macro
238     keymacro macro;
239     memset(&macro, 0, sizeof(macro));
240     // Scan the left side for key names, separated by +
241     int empty = 1;
242     int left = strlen(keys), right = strlen(assignment);
243     int position = 0, field = 0;
244     char keyname[12];
245     while(position < left && sscanf(keys + position, "%10[^\n]", keyname, &field) == 1){
246         int keycode;
247         if((sscanf(keyname, "%d", &keycode) && keycode >= 0 && keycode <
N_KEYS_INPUT)
248 || (sscanf(keyname, "%x%x", &keycode) && keycode >= 0 && keycode <
N_KEYS_INPUT)){
249             // Set a key numerically
250             SET_KEYBIT(macro.combo, keycode);
251             empty = 0;
252         } else {
253             // Find this key in the keymap
254             for(unsigned i = 0; i < N_KEYS_INPUT; i++){
255                 if(keymap[i].name && !strcmp(keyname, keymap[i].name)){
256                     macro.combo[i / 8] |= 1 << (i % 8);
257                     empty = 0;
258                     break;
259                 }
260             }
261         }
262         if(keys[position += field] == '+')
263             position++;
264     }
265     if(empty)
266         return;
267     // Count the number of actions (comma separated)
268     int count = 1;
269     for(const char* c = assignment; *c != 0; c++){
270         if(*c == ',')
271             count++;
272     }
273     // Allocate a buffer for them
274     macro.actions = calloc(count, sizeof(macroaction));
275     macro.actioncount = 0;
276     // Scan the actions
277     position = 0;
278     field = 0;
279     while(position < right && sscanf(assignment + position, "%11[^\n]", keyname, &field) == 1){
280         if(!strcmp(keyname, "clear"))
281             break;
282         int down = (keyname[0] == '+');
283         if(down || keyname[0] == '-'){
284             int keycode;
285             if((sscanf(keyname + 1, "%d", &keycode) && keycode >= 0 && keycode < N_KEYS_INPUT)
|| (sscanf(keyname + 1, "%x%x", &keycode) && keycode >= 0 && keycode <
```

```

N_KEYS_INPUT)){
287     // Set a key numerically
288     macro.actions[macro.actioncount].scan =
keymap[keycode].scan;
289     macro.actions[macro.actioncount].down = down;
290     macro.actioncount++;
291 } else {
292     // Find this key in the keymap
293     for(unsigned i = 0; i < N_KEYS_INPUT; i++){
294         if(keymap[i].name && !strcmp(keyname + 1, keymap[i].name)){
295             macro.actions[macro.actioncount].scan =
keymap[i].scan;
296             macro.actions[macro.actioncount].down = down;
297             macro.actioncount++;
298             break;
299         }
300     }
301 }
302 }
303 if(assignment[position += field] == ',')
304     position++;
305 }
306
307 // See if there's already a macro with this trigger
308 keymacro* macros = bind->macros;
309 for(int i = 0; i < bind->macrocount; i++){
310     if(!memcmp(macros[i].combo, macro.combo, N_KEYBYTES_INPUT)){
311         free(macros[i].actions);
312         // If the new macro has no actions, erase the existing one
313         if(!macro.actioncount){
314             for(int j = i + 1; j < bind->macrocount; j++)
315                 memcpy(macros + j - 1, macros + j, sizeof(keymacro));
316             bind->macrocount--;
317         } else
318             // If there are actions, replace the existing with the new
319             memcpy(macros + i, &macro, sizeof(keymacro));
320         return;
321     }
322 }
323
324 // Add the macro to the device settings if not empty
325 if(macro.actioncount < 1)
326     return;
327 memcpy(bind->macros + (bind->macrocount++), &macro, sizeof(
keymacro));
328 if(bind->macrocount >= bind->macrocap)
329     bind->macros = realloc(bind->macros, (bind->macrocap += 16) * sizeof(
keymacro));
330 }

```

Here is the caller graph for this function:



5.16.2.2 cmd_bind()

```

void cmd_bind (
    usbdevice * kb,
    usbmode * mode,
    int dummy,

```

```

    int keyindex,
    const char * to )

```

Definition at line 188 of file input.c.

References `binding::base`, `usbmode::bind`, `imutex`, `keymap`, `N_KEYS_INPUT`, and `key::scan`.

```

188                                     {
189     if(keyindex >= N_KEYS_INPUT)
190         return;
191     // Find the key to bind to
192     int tocode = 0;
193     if(sscanf(to, "%x%u", &tocode) != 1 && sscanf(to, "%u", &tocode) == 1 && tocode <
N_KEYS_INPUT){
194         pthread_mutex_lock(imutex(kb));
195         mode->bind.base[keyindex] = tocode;
196         pthread_mutex_unlock(imutex(kb));
197         return;
198     }
199     // If not numeric, look it up
200     for(int i = 0; i < N_KEYS_INPUT; i++){
201         if(keymap[i].name && !strcmp(to, keymap[i].name)){
202             pthread_mutex_lock(imutex(kb));
203             mode->bind.base[keyindex] = keymap[i].scan;
204             pthread_mutex_unlock(imutex(kb));
205             return;
206         }
207     }
208 }

```

5.16.2.3 cmd_macro()

```

void cmd_macro (
    usbdevice * kb,
    usbmode * mode,
    const int notifynumber,
    const char * keys,
    const char * assignment )

```

Definition at line 332 of file input.c.

References `_cmd_macro()`, and `imutex`.

```

332     {
333     pthread_mutex_lock(imutex(kb));
334     _cmd_macro(mode, keys, assignment);
335     pthread_mutex_unlock(imutex(kb));
336 }

```

Here is the call graph for this function:



5.16.2.4 cmd_rebind()

```
void cmd_rebind (
    usbdevice * kb,
    usbmode * mode,
    int dummy,
    int keyindex,
    const char * to )
```

Definition at line 218 of file input.c.

References `binding::base`, `usbmode::bind`, `imutex`, `keymap`, `N_KEYS_INPUT`, and `key::scan`.

```
218                                     {
219     if(keyindex >= N_KEYS_INPUT)
220         return;
221     pthread_mutex_lock(imutex(kb));
222     mode->bind.base[keyindex] = keymap[keyindex].scan;
223     pthread_mutex_unlock(imutex(kb));
224 }
```

5.16.2.5 cmd_unbind()

```
void cmd_unbind (
    usbdevice * kb,
    usbmode * mode,
    int dummy,
    int keyindex,
    const char * to )
```

Definition at line 210 of file input.c.

References `binding::base`, `usbmode::bind`, `imutex`, `KEY_UNBOUND`, and `N_KEYS_INPUT`.

```
210                                     {
211     if(keyindex >= N_KEYS_INPUT)
212         return;
213     pthread_mutex_lock(imutex(kb));
214     mode->bind.base[keyindex] = KEY_UNBOUND;
215     pthread_mutex_unlock(imutex(kb));
216 }
```

5.16.2.6 freebind()

```
void freebind (
    binding * bind )
```

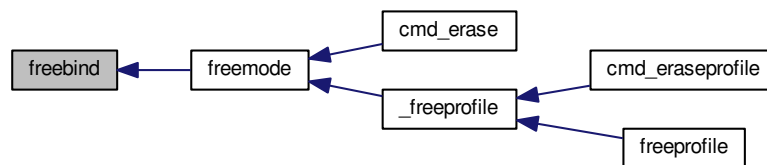
Definition at line 181 of file input.c.

References keymacro::actions, binding::macrocount, and binding::macros.

Referenced by freemode().

```
181     {
182     for(int i = 0; i < bind->macrocount; i++)
183         free(bind->macros[i].actions);
184     free(bind->macros);
185     memset(bind, 0, sizeof(*bind));
186 }
```

Here is the caller graph for this function:



5.16.2.7 initbind()

```
void initbind (
    binding * bind )
```

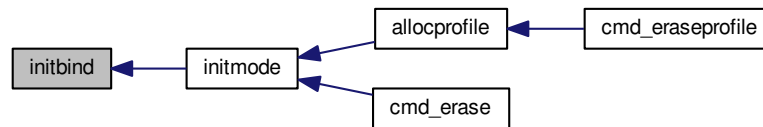
Definition at line 173 of file input.c.

References binding::base, keymap, binding::macrocap, binding::macrocount, binding::macros, N_KEYS_INPUT, and key::scan.

Referenced by initmode().

```
173     {
174     for(int i = 0; i < N_KEYS_INPUT; i++)
175         bind->base[i] = keymap[i].scan;
176     bind->macros = calloc(32, sizeof(keymacro));
177     bind->macrocap = 32;
178     bind->macrocount = 0;
179 }
```

Here is the caller graph for this function:



5.16.2.8 inputupdate()

```
void inputupdate (
    usbdevice * kb )
```

Definition at line 122 of file input.c.

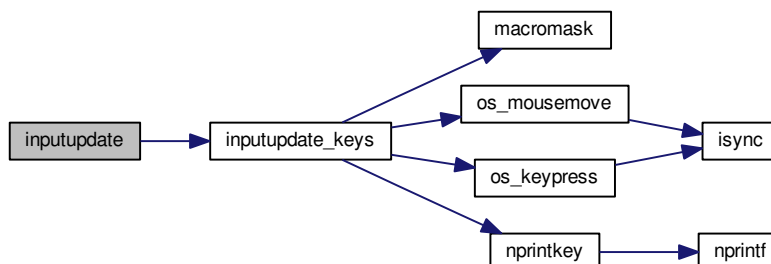
References `usbdevice::input`, `inputupdate_keys()`, `usbinput::keys`, `N_KEYBYTES_INPUT`, `usbinput::prevkeys`, `usbdevice::profile`, `usbinput::rel_x`, `usbinput::rel_y`, `usbdevice::uinput_kb`, and `usbdevice::uinput_mouse`.

Referenced by `os_inputmain()`, `setactive_kb()`, and `setactive_mouse()`.

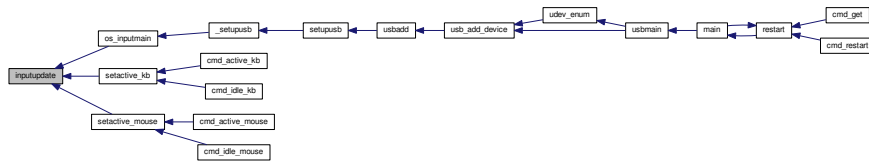
```

122                                     {
123 #ifdef OS_LINUX
124     if ((!kb->uinput_kb || !kb->uinput_mouse)
125 #else
126     if (!kb->event
127 #endif
128         || !kb->profile)
129         return;
130     // Process key/button input
131     inputupdate_keys(kb);
132     // Process mouse movement
133     usbinput* input = &kb->input;
134     if(input->rel_x != 0 || input->rel_y != 0){
135         os_mousemove(kb, input->rel_x, input->rel_y);
136         input->rel_x = input->rel_y = 0;
137     }
138     // Finish up
139     memcpy(input->prevkeys, input->keys, N_KEYBYTES_INPUT);
140 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



5.16.2.9 inputupdate_keys()

```
static void inputupdate_keys (
    usbdevice * kb ) [static]
```

Definition at line 15 of file input.c.

References `keymacro::actioncount`, `keymacro::actions`, `usbdevice::active`, `binding::base`, `usbmode::bind`, `keymacro::combo`, `usbprofile::currentmode`, `usbdevice::delay`, `macroaction::down`, `usbdevice::input`, `IS_MOD`, `IS_WHEEL`, `keymap`, `usbinput::keys`, `binding::macrocount`, `macromask()`, `binding::macros`, `N_KEYBYTES_INPUT`, `N_KEYS_INPUT`, `usbmode::notify`, `nprintkey()`, `os_keypress()`, `os_mousemove()`, `OUTFIFO_MAX`, `usbinput::prevkeys`, `usbdevice::profile`, `macroaction::rel_x`, `macroaction::rel_y`, `macroaction::scan`, `key::scan`, `SCAN_SILENT`, and `keymacro::triggered`.

Referenced by `inputupdate()`.

```

15                                     {
16     usbmode* mode = kb->profile->currentmode;
17     binding* bind = &mode->bind;
18     usbinput* input = &kb->input;
19     // Don't do anything if the state hasn't changed
20     if (!memcmp(input->prevkeys, input->keys, N_KEYBYTES_INPUT))
21         return;
22     // Look for macros matching the current state
23     int macrotrigger = 0;
24     if (kb->active) {
25         for (int i = 0; i < bind->macrocount; i++) {
26             keymacro* macro = &bind->macros[i];
27             if (macromask(input->keys, macro->combo)) {
28                 if (!macro->triggered) {
29                     macrotrigger = 1;
30                     macro->triggered = 1;
31                     // Send events for each keypress in the macro
32                     for (int a = 0; a < macro->actioncount; a++) {
33                         macroaction* action = macro->actions + a;
34                         if (action->rel_x != 0 || action->rel_y != 0)
35                             os_mousemove(kb, action->rel_x, action->
36                             rel_y);
37                         else {
38                             os_keypress(kb, action->scan, action->
39                             down);
40                             if (kb->delay) {
41                                 if (a > 200) usleep (100);
42                                 else if (a > 20) usleep (30);
43                             }
44                         }
45                     } else {
46                         macro->triggered = 0;
47                     }
48                 }
49             }
50             // Make a list of keycodes to send. Rearrange them so that modifier keydowns always come first
51             // and modifier keyups always come last. This ensures that shortcut keys will register properly

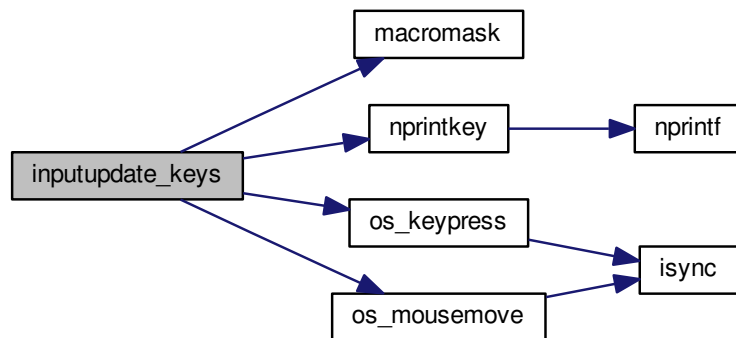
```

```

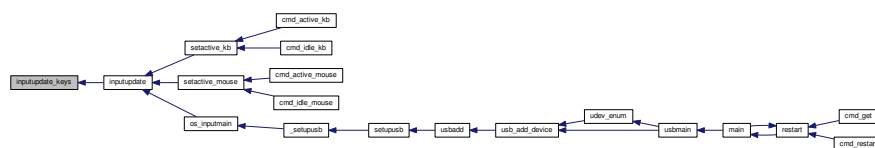
52 // even if both keydown events happen at once.
53 // N_KEYS + 4 is used because the volume wheel generates keydowns and keyups at the same time
54 // (it's currently impossible to press all four at once, but safety first)
55 int events[N_KEYS_INPUT + 4];
56 int modcount = 0, keycount = 0, rmodcount = 0;
57 for(int byte = 0; byte < N_KEYBYTES_INPUT; byte++){
58     char oldb = input->prevkeys[byte], newb = input->keys[byte];
59     if(oldb == newb)
60         continue;
61     for(int bit = 0; bit < 8; bit++){
62         int keyindex = byte * 8 + bit;
63         if(keyindex >= N_KEYS_INPUT)
64             break;
65         const key* map = keymap + keyindex;
66         int scancode = (kb->active) ? bind->base[keyindex] : map->
scan;
67         char mask = 1 << bit;
68         char old = oldb & mask, new = newb & mask;
69         // If the key state changed, send it to the input device
70         if(old != new){
71             // Don't echo a key press if a macro was triggered or if there's no scancode associated
72             if(!macrotrigger && !(scancode & SCAN_SILENT)){
73                 if(IS_MOD(scancode)){
74                     if(new){
75                         // Modifier down: Add to the end of modifier keys
76                         for(int i = keycount + rmodcount; i > 0; i--){
77                             events[modcount + i] = events[modcount + i - 1];
78                             // Add 1 to the scancode because A is zero on OSX
79                             // Positive code = keydown, negative code = keyup
80                             events[modcount++] = scancode + 1;
81                         } else {
82                             // Modifier up: Add to the end of everything
83                             events[modcount + keycount + rmodcount++] = -(scancode + 1);
84                         }
85                     } else {
86                         // Regular keypress: add to the end of regular keys
87                         for(int i = rmodcount; i > 0; i--){
88                             events[modcount + keycount + i] = events[modcount + keycount + i - 1];
89                             events[modcount + keycount++] = new ? (scancode + 1) : -(scancode + 1);
90                             // The volume wheel and the mouse wheel don't generate keyups, so create them
91                             automatically
92                             #define IS_WHEEL(scan, kb) ((scan) == KEY_VOLUMEUP || (scan) == KEY_VOLUMEDOWN || (scan) == BTN_WHEELUP
|| (scan) == BTN_WHEELDOWN) && !IS_K65(kb))
93                             if(new && IS_WHEEL(map->scan, kb)){
94                                 for(int i = rmodcount; i > 0; i--){
95                                     events[modcount + keycount + i] = events[modcount + keycount + i - 1];
96                                     events[modcount + keycount++] = -(scancode + 1);
97                                     input->keys[byte] &= ~mask;
98                                 }
99                             }
100                             // Print notifications if desired
101                             if(kb->active){
102                                 for(int notify = 0; notify < OUTFIFO_MAX; notify++){
103                                     if(mode->notify[notify][byte] & mask){
104                                         nprintkey(kb, notify, keyindex, new);
105                                         // Wheels doesn't generate keyups
106                                         if(new && IS_WHEEL(map->scan, kb))
107                                             nprintkey(kb, notify, keyindex, 0);
108                                     }
109                                 }
110                             }
111                         }
112                     }
113                 }
114                 // Process all queued keypresses
115                 int totalkeys = modcount + keycount + rmodcount;
116                 for(int i = 0; i < totalkeys; i++){
117                     int scancode = events[i];
118                     os_keypress(kb, (scancode < 0 ? -scancode : scancode) - 1, scancode > 0);
119                 }
120 }

```

Here is the call graph for this function:



Here is the caller graph for this function:



5.16.2.10 macromask()

```

int macromask (
    const uchar * key1,
    const uchar * key2 )

```

Definition at line 5 of file input.c.

References `N_KEYBYTES_INPUT`.

Referenced by `inputupdate_keys()`.

```

5
6 // Scan a macro against key input. Return 0 if any of them don't match
7 for(int i = 0; i < N_KEYBYTES_INPUT; i++){
8     // if((key1[i] & key2[i]) != key2[i])
9     if(key1[i] != key2[i]) // Changed to detect G-keys + modifiers
10         return 0;
11 }
12 return 1;
13 }

```


5.17.1 Macro Definition Documentation

5.17.1.1 IS_MOD

```
#define IS_MOD(
    s ) ((s) == KEY_CAPSLOCK || (s) == KEY_NUMLOCK || (s) == KEY_SCROLLLOCK || (s) ==
KEY_LEFTSHIFT || (s) == KEY_RIGHTSHIFT || (s) == KEY_LEFTCTRL || (s) == KEY_RIGHTCTRL || (s) == KEY_LEFTMETA || (s) == KEY_RIGHTMETA || (s) == KEY_LEFTALT || (s) == KEY_RIGHTALT || (s) == KEY_FN)
```

Definition at line 34 of file input.h.

Referenced by inputupdate_keys().

5.17.2 Function Documentation

5.17.2.1 cmd_bind()

```
void cmd_bind (
    usbdevice * kb,
    usbmode * mode,
    int dummy,
    int keyindex,
    const char * to )
```

Definition at line 188 of file input.c.

References binding::base, usbmode::bind, imutex, keymap, N_KEYS_INPUT, and key::scan.

```
188                                     {
189     if(keyindex >= N_KEYS_INPUT)
190         return;
191     // Find the key to bind to
192     int tocode = 0;
193     if(sscanf(to, "%xux", &tocode) != 1 && sscanf(to, "%u", &tocode) == 1 && tocode <
N_KEYS_INPUT){
194         pthread_mutex_lock(imutex(kb));
195         mode->bind.base[keyindex] = tocode;
196         pthread_mutex_unlock(imutex(kb));
197         return;
198     }
199     // If not numeric, look it up
200     for(int i = 0; i < N_KEYS_INPUT; i++){
201         if(keymap[i].name && !strcmp(to, keymap[i].name)){
202             pthread_mutex_lock(imutex(kb));
203             mode->bind.base[keyindex] = keymap[i].scan;
204             pthread_mutex_unlock(imutex(kb));
205             return;
206         }
207     }
208 }
```

5.17.2.2 cmd_macro()

```
void cmd_macro (
    usbdevice * kb,
    usbmode * mode,
    const int notifynumber,
    const char * keys,
    const char * assignment )
```

Definition at line 332 of file input.c.

References `_cmd_macro()`, and `imutex`.

```
332
333 {
334     pthread_mutex_lock(imutex(kb));
335     _cmd_macro(mode, keys, assignment);
336     pthread_mutex_unlock(imutex(kb));
337 }
```

Here is the call graph for this function:



5.17.2.3 cmd_rebind()

```
void cmd_rebind (
    usbdevice * kb,
    usbmode * mode,
    int dummy,
    int keyindex,
    const char * ignored )
```

Definition at line 218 of file input.c.

References `binding::base`, `usbmode::bind`, `imutex`, `keymap`, `N_KEYS_INPUT`, and `key::scan`.

```
218
219     if(keyindex >= N_KEYS_INPUT) {
220         return;
221     pthread_mutex_lock(imutex(kb));
222     mode->bind.base[keyindex] = keymap[keyindex].scan;
223     pthread_mutex_unlock(imutex(kb));
224 }
```

5.17.2.4 cmd_unbind()

```
void cmd_unbind (
    usbdevice * kb,
    usbmode * mode,
    int dummy,
    int keyindex,
    const char * ignored )
```

Definition at line 210 of file input.c.

References `binding::base`, `usbmode::bind`, `imutex`, `KEY_UNBOUND`, and `N_KEYS_INPUT`.

```
210
211     if(keyindex >= N_KEYS_INPUT)
212         return;
213     pthread_mutex_lock(&imutex(kb));
214     mode->bind.base[keyindex] = KEY_UNBOUND;
215     pthread_mutex_unlock(&imutex(kb));
216 }
```

5.17.2.5 freebind()

```
void freebind (
    binding * bind )
```

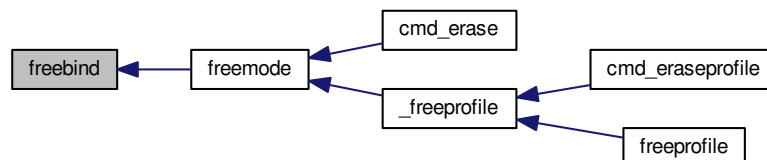
Definition at line 181 of file input.c.

References `keymacro::actions`, `binding::macrocount`, and `binding::macros`.

Referenced by `freemode()`.

```
181
182     for(int i = 0; i < bind->macrocount; i++)
183         free(bind->macros[i].actions);
184     free(bind->macros);
185     memset(bind, 0, sizeof(*bind));
186 }
```

Here is the caller graph for this function:



5.17.2.6 initbind()

```
void initbind (
    binding * bind )
```

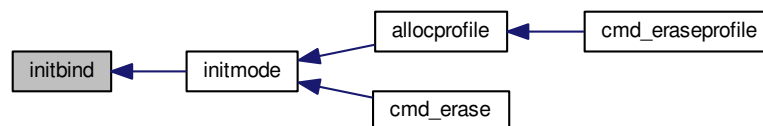
Definition at line 173 of file input.c.

References `binding::base`, `keymap`, `binding::macrocap`, `binding::macrocount`, `binding::macros`, `N_KEYS_INPUT`, and `key::scan`.

Referenced by `initmode()`.

```
173     {
174     for(int i = 0; i < N_KEYS_INPUT; i++)
175         bind->base[i] = keymap[i].scan;
176     bind->macros = calloc(32, sizeof(keymacro));
177     bind->macrocap = 32;
178     bind->macrocount = 0;
179 }
```

Here is the caller graph for this function:



5.17.2.7 inputupdate()

```
void inputupdate (
    usbdevice * kb )
```

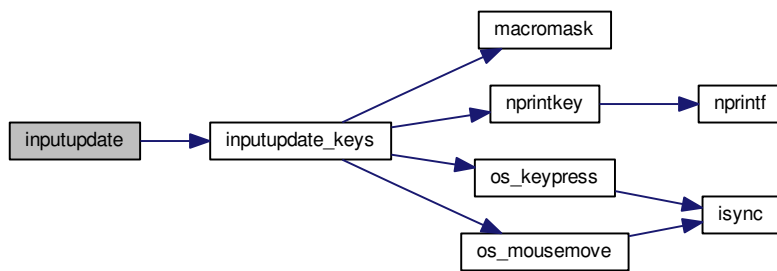
Definition at line 122 of file input.c.

References `usbdevice::input`, `inputupdate_keys()`, `usbinput::keys`, `N_KEYBYTES_INPUT`, `usbinput::prevkeys`, `usbdevice::profile`, `usbinput::rel_x`, `usbinput::rel_y`, `usbdevice::uinput_kb`, and `usbdevice::uinput_mouse`.

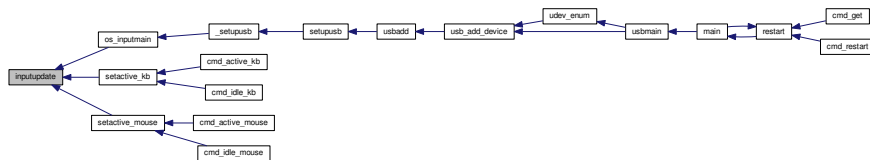
Referenced by `os_inputmain()`, `setactive_kb()`, and `setactive_mouse()`.

```
122     {
123     #ifdef OS_LINUX
124     if ((!kb->uinput_kb || !kb->uinput_mouse)
125     #else
126     if (!kb->event
127     #endif
128         || !kb->profile)
129         return;
130     // Process key/button input
131     inputupdate_keys(kb);
132     // Process mouse movement
133     usbinput* input = &kb->input;
134     if(input->rel_x != 0 || input->rel_y != 0){
135         os_mousemove(kb, input->rel_x, input->rel_y);
136         input->rel_x = input->rel_y = 0;
137     }
138     // Finish up
139     memcpy(input->prevkeys, input->keys, N_KEYBYTES_INPUT);
140 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



5.17.2.8 os_inputclose()

```
void os_inputclose (
    usbdevice * kb )
```

Definition at line 76 of file input_linux.c.

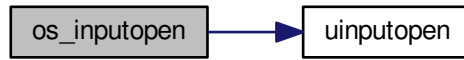
References `ckb_warn`, `usbdevice::uinput_kb`, and `usbdevice::uinput_mouse`.

Referenced by `closeusb()`.

```

76         {
77     if(kb->uinput_kb <= 0 || kb->uinput_mouse <= 0)
78         return;
79     // Set all keys released
80     struct input_event event;
81     memset(&event, 0, sizeof(event));
82     event.type = EV_KEY;
83     for(int key = 0; key < KEY_CNT; key++){
84         event.code = key;
85         if(write(kb->uinput_kb - 1, &event, sizeof(event)) <= 0)
86             ckb_warn("uinput write failed: %s\n", strerror(errno));
87         if(write(kb->uinput_mouse - 1, &event, sizeof(event)) <= 0)
88             ckb_warn("uinput write failed: %s\n", strerror(errno));
89     }
90     event.type = EV_SYN;
91     event.code = SYN_REPORT;
92     if(write(kb->uinput_kb - 1, &event, sizeof(event)) <= 0)
93         ckb_warn("uinput write failed: %s\n", strerror(errno));
94     if(write(kb->uinput_mouse - 1, &event, sizeof(event)) <= 0)
95         ckb_warn("uinput write failed: %s\n", strerror(errno));
  
```


Here is the call graph for this function:



Here is the caller graph for this function:



5.17.2.10 os_keypress()

```

void os_keypress (
    usbdevice * kb,
    int scancode,
    int down )
  
```

Definition at line 118 of file input_linux.c.

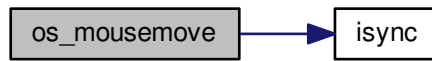
References `BTN_WHEELDOWN`, `BTN_WHEELUP`, `ckb_warn`, `isync()`, `SCAN_MOUSE`, `usbdevice::uinput_kb`, and `usbdevice::uinput_mouse`.

Referenced by `inputupdate_keys()`.

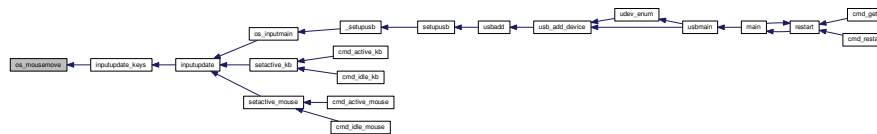
```

118                                     {
119     struct input_event event;
120     memset(&event, 0, sizeof(event));
121     int is_mouse = 0;
122     if(scancode == BTN_WHEELUP || scancode == BTN_WHEELDOWN) {
123         // The mouse wheel is a relative axis
124         if(!down)
125             return;
126         event.type = EV_REL;
127         event.code = REL_WHEEL;
128         event.value = (scancode == BTN_WHEELUP ? 1 : -1);
129         is_mouse = 1;
130     } else {
131         // Mouse buttons and key events are both EV_KEY. The scancodes are already correct, just remove the
132         ckb bit
133         event.type = EV_KEY;
134         event.code = scancode & ~SCAN_MOUSE;
135         event.value = down;
136         is_mouse = !(scancode & SCAN_MOUSE);
137     }
138     if(write((is_mouse ? kb->uinput_mouse : kb->uinput_kb) - 1, &event, sizeof(event))
139     <= 0)
140         ckb_warn("uinput write failed: %s\n", strerror(errno));
141     else
142         isync(kb);
143 }
  
```


Here is the call graph for this function:



Here is the caller graph for this function:



5.17.2.12 os_setupindicators()

```
int os_setupindicators (
    usbdevice * kb )
```

Definition at line 193 of file input_linux.c.

References `_ledthread()`, `usbdevice::hw_ileds`, `usbdevice::hw_ileds_old`, and `usbdevice::ileds`.

Referenced by `_setupusb()`.

```

193
194     // Initialize LEDs to all off
195     kb->hw_ileds = kb->hw_ileds_old = kb->ileds = 0;
196     // Create and detach thread to read LED events
197     pthread_t thread;
198     int err = pthread_create(&thread, 0, _ledthread, kb);
199     if(err != 0)
200         return err;
201     pthread_detach(thread);
202     return 0;
203 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



5.17.2.13 updateindicators_kb()

```

void updateindicators_kb (
    usbdevice * kb,
    int force )

```

Definition at line 142 of file input.c.

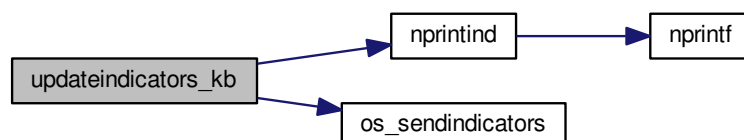
References `usbdevice::active`, `usbprofile::currentmode`, `DELAY_SHORT`, `usbdevice::hw_ileds`, `usbdevice::hw_ileds_old`, `I_CAPS`, `I_NUM`, `I_SCROLL`, `usbdevice::ileds`, `usbmode::inotify`, `usbmode::ioff`, `usbmode::ion`, `nprintind()`, `os_sendindicators()`, `OUTFIFO_MAX`, and `usbdevice::profile`.

```

142     {
143         // Read current hardware indicator state (set externally)
144         uchar old = kb->ileds, hw_old = kb->hw_ileds_old;
145         uchar new = kb->hw_ileds, hw_new = new;
146         // Update them if needed
147         if(kb->active){
148             usbmode* mode = kb->profile->currentmode;
149             new = (new & ~mode->ioff) | mode->ion;
150         }
151         kb->ileds = new;
152         kb->hw_ileds_old = hw_new;
153         if(old != new || force){
154             DELAY_SHORT(kb);
155             os_sendindicators(kb);
156         }
157         // Print notifications if desired
158         if(!kb->active)
159             return;
160         usbmode* mode = kb->profile->currentmode;
161         uchar indicators[] = { I_NUM, I_CAPS, I_SCROLL };
162         for(unsigned i = 0; i < sizeof(indicators) / sizeof(uchar); i++){
163             uchar mask = indicators[i];
164             if((hw_old & mask) == (hw_new & mask))
165                 continue;
166             for(int notify = 0; notify < OUTFIFO_MAX; notify++){
167                 if(mode->inotify[notify] & mask)
168                     nprintind(kb, notify, mask, hw_new & mask);
169             }
170         }
171     }

```

Here is the call graph for this function:



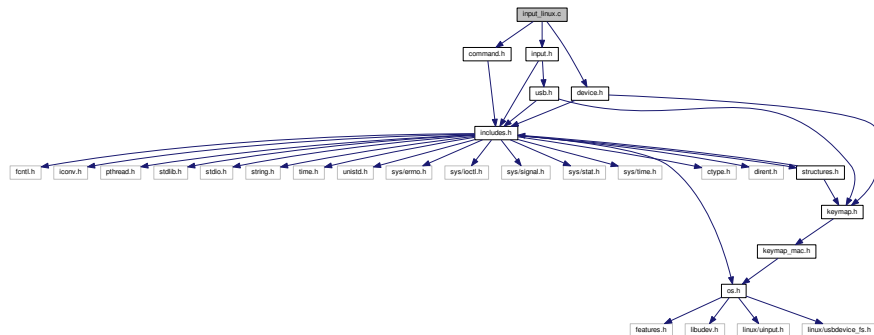
5.18 input_linux.c File Reference

```
#include "command.h"
```

```
#include "device.h"
```

```
#include "input.h"
```

Include dependency graph for input_linux.c:



Functions

- int `uinputopen` (struct `uinput_user_dev` *udev, int mouse)
- int `os_inputopen` (`usbdevice` *kb)
os_inputopen
- void `os_inputclose` (`usbdevice` *kb)
- static void `isync` (`usbdevice` *kb)
- void `os_keypress` (`usbdevice` *kb, int scancode, int down)
- void `os_mousemove` (`usbdevice` *kb, int x, int y)
- void * `_ledthread` (void *ctx)
- int `os_setupindicators` (`usbdevice` *kb)

5.18.1 Function Documentation

5.18.1.1 _ledthread()

```
void* _ledthread (
    void * ctx )
```

Definition at line 165 of file input_linux.c.

References ckb_warn, dmutex, usbdevice::hw_ileds, usbdevice::uinput_kb, and usbdevice::vtable.

Referenced by `os_setupindicators()`.

5.18.1.3 os_inputclose()

```
void os_inputclose (
    usbdevice * kb )
```

Definition at line 76 of file input_linux.c.

References `ckb_warn`, `usbdevice::uinput_kb`, and `usbdevice::uinput_mouse`.

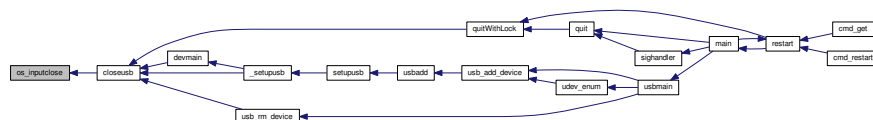
Referenced by closeusb().

```

76         {
77             if(kb->uinput_kb <= 0 || kb->uinput_mouse <= 0)
78                 return;
79             // Set all keys released
80             struct input_event event;
81             memset(&event, 0, sizeof(event));
82             event.type = EV_KEY;
83             for(int key = 0; key < KEY_CNT; key++){
84                 event.code = key;
85                 if(write(kb->uinput_kb - 1, &event, sizeof(event)) <= 0)
86                     ckb_warn("uinput write failed: %s\n", strerror(errno));
87                 if(write(kb->uinput_mouse - 1, &event, sizeof(event)) <= 0)
88                     ckb_warn("uinput write failed: %s\n", strerror(errno));
89             }
90             event.type = EV_SYN;
91             event.code = SYN_REPORT;
92             if(write(kb->uinput_kb - 1, &event, sizeof(event)) <= 0)
93                 ckb_warn("uinput write failed: %s\n", strerror(errno));
94             if(write(kb->uinput_mouse - 1, &event, sizeof(event)) <= 0)
95                 ckb_warn("uinput write failed: %s\n", strerror(errno));
96             // Close the keyboard
97             ioctl(kb->uinput_kb - 1, UI_DEV_DESTROY);
98             close(kb->uinput_kb - 1);
99             kb->uinput_kb = 0;
100             // Close the mouse
101             ioctl(kb->uinput_mouse - 1, UI_DEV_DESTROY);
102             close(kb->uinput_mouse - 1);
103             kb->uinput_mouse = 0;
104         }

```

Here is the caller graph for this function:



5.18.1.4 os_inputopen()

```
int os_inputopen (
    usbdevice * kb )
```

Parameters

<i>kb</i>	
-----------	--

Returns

Some tips on using `uinput_user_dev` in

Definition at line 55 of file `input_linux.c`.

References `usbdevice::fwversion`, `INDEX_OF`, `keyboard`, `usbdevice::name`, `usbdevice::product`, `usbdevice::uinput_↵
_kb`, `usbdevice::uinput_mouse`, `uinputopen()`, and `usbdevice::vendor`.

Referenced by `_setupusb()`.

```

55     {
56     // Create the new input device
57     int index = INDEX_OF(kb, keyboard);
58     struct uinput_user_dev indev;
59     memset(&indev, 0, sizeof(indev));
60     snprintf(indev.name, UINPUT_MAX_NAME_SIZE, "ckb%d: %s", index, kb->name);
61     indev.id.bustype = BUS_USB;
62     indev.id.vendor = kb->vendor;
63     indev.id.product = kb->product;
64     indev.id.version = kb->fwversion;
65     // Open keyboard
66     int fd = uinputopen(&indev, 0);
67     kb->uinput_kb = fd;
68     if (fd <= 0)
69         return 0;
70     // Open mouse
71     fd = uinputopen(&indev, 1);
72     kb->uinput_mouse = fd;
73     return fd <= 0;
74 }

```

Here is the call graph for this function:



Here is the caller graph for this function:



5.18.1.5 os_keypress()

```
void os_keypress (
    usbdevice * kb,
    int scancode,
    int down )
```

Definition at line 118 of file input_linux.c.

References `BTN_WHEELDOWN`, `BTN_WHEELUP`, `ckb_warn`, `isync()`, `SCAN_MOUSE`, `usbdevice::uinput_kb`, and `usbdevice::uinput_mouse`.

Referenced by `inputupdate_keys()`.

```
118                                     {
119     struct input_event event;
120     memset(&event, 0, sizeof(event));
121     int is_mouse = 0;
122     if(scancode == BTN_WHEELUP || scancode == BTN_WHEELDOWN) {
123         // The mouse wheel is a relative axis
124         if(!down)
125             return;
126         event.type = EV_REL;
127         event.code = REL_WHEEL;
128         event.value = (scancode == BTN_WHEELUP ? 1 : -1);
129         is_mouse = 1;
130     } else {
131         // Mouse buttons and key events are both EV_KEY. The scancodes are already correct, just remove the
132         ckb bit
133         event.type = EV_KEY;
134         event.code = scancode & ~SCAN_MOUSE;
135         event.value = down;
136         is_mouse = !(scancode & SCAN_MOUSE);
137     }
138     if(write((is_mouse ? kb->uinput_mouse : kb->uinput_kb) - 1, &event, sizeof(event))
139         <= 0)
140         ckb_warn("uinput write failed: %s\n", strerror(errno));
141     else
142         isync(kb);
143 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



5.18.1.6 os_mousemove()

```
void os_mousemove (
    usbdevice * kb,
    int x,
    int y )
```

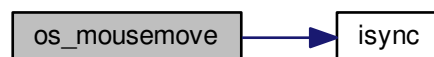
Definition at line 143 of file input_linux.c.

References `ckb_warn`, `isync()`, and `usbdevice::uinput_mouse`.

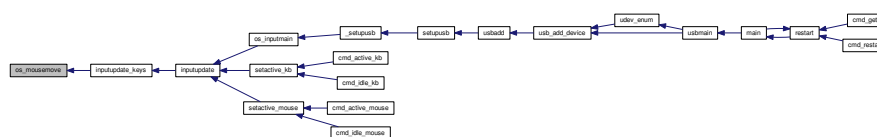
Referenced by `inputupdate_keys()`.

```
143                                     {
144     struct input_event event;
145     memset(&event, 0, sizeof(event));
146     event.type = EV_REL;
147     if(x != 0){
148         event.code = REL_X;
149         event.value = x;
150         if(write(kb->uinput_mouse - 1, &event, sizeof(event)) <= 0)
151             ckb_warn("uinput write failed: %s\n", strerror(errno));
152         else
153             isync(kb);
154     }
155     if(y != 0){
156         event.code = REL_Y;
157         event.value = y;
158         if(write(kb->uinput_mouse - 1, &event, sizeof(event)) <= 0)
159             ckb_warn("uinput write failed: %s\n", strerror(errno));
160         else
161             isync(kb);
162     }
163 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



5.18.1.7 os_setupindicators()

```
int os_setupindicators (
    usbdevice * kb )
```

Definition at line 193 of file input_linux.c.

References `_ledthread()`, `usbdevice::hw_ileads`, `usbdevice::hw_ileads_old`, and `usbdevice::ileads`.

Referenced by `_setupusb()`.

```
193                                     {
194     // Initialize LEDs to all off
195     kb->hw_ileads = kb->hw_ileads_old = kb->ileads = 0;
196     // Create and detach thread to read LED events
197     pthread_t thread;
198     int err = pthread_create(&thread, 0, _ledthread, kb);
199     if(err != 0)
200         return err;
201     pthread_detach(thread);
202     return 0;
203 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



5.18.1.8 uinputopen()

```
int uinputopen (
    struct uinput_user_dev * indev,
    int mouse )
```

Definition at line 9 of file input_linux.c.

References `ckb_err`, and `ckb_warn`.

Referenced by `os_uinputopen()`.

```

9                                     {
10     int fd = open("/dev/uinput", O_RDWR);
11     if(fd < 0){
12         // If that didn't work, try /dev/input/uinput instead
13         fd = open("/dev/input/uinput", O_RDWR);
14         if(fd < 0){
15             ckb_err("Failed to open uinput: %s\n", strerror(errno));
16             return 0;
17         }
18     }
19     // Enable all keys and mouse buttons
20     ioctl(fd, UI_SET_EVBIT, EV_KEY);
21     for(int i = 0; i < KEY_CNT; i++)
22         ioctl(fd, UI_SET_KEYBIT, i);
23     if(mouse){
24         // Enable mouse axes
25         ioctl(fd, UI_SET_EVBIT, EV_REL);
26         for(int i = 0; i < REL_CNT; i++)
27             ioctl(fd, UI_SET_RELBIT, i);
28     } else {
29         // Enable LEDs
30         ioctl(fd, UI_SET_EVBIT, EV_LED);
31         for(int i = 0; i < LED_CNT; i++)
32             ioctl(fd, UI_SET_LEDBIT, i);
33         // Enable autorepeat
34         ioctl(fd, UI_SET_EVBIT, EV_REP);
35     }
36     // Enable synchronization
37     ioctl(fd, UI_SET_EVBIT, EV_SYN);
38     // Create the device
39     if(write(fd, indev, sizeof(*indev)) <= 0)
40         ckb_warn("uinput write failed: %s\n", strerror(errno));
41     if(ioctl(fd, UI_DEV_CREATE)){
42         ckb_err("Failed to create uinput device: %s\n", strerror(errno));
43         close(fd);
44         return 0;
45     }
46     return fd + 1;
47 }

```

Here is the caller graph for this function:



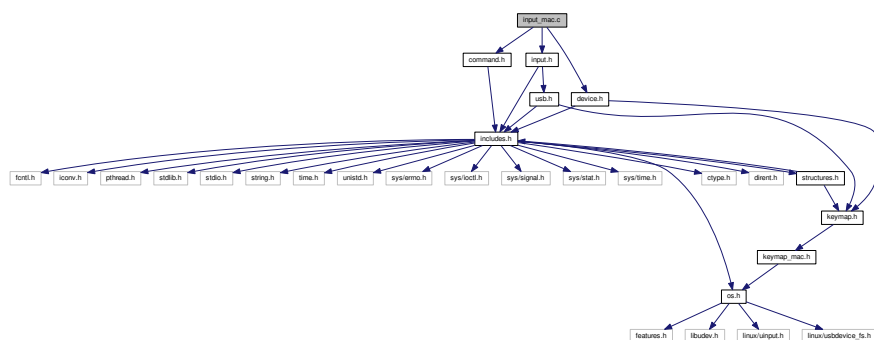
5.19 input_mac.c File Reference

```
#include "command.h"
```

```
#include "device.h"
```

```
#include "input.h"
```

Include dependency graph for input_mac.c:



5.21.1 Macro Definition Documentation

5.21.1.1 BUTTON_HID_COUNT

```
#define BUTTON_HID_COUNT 5
```

Definition at line 364 of file keymap.c.

Referenced by corsair_mousecopy(), and hid_mouse_translate().

5.21.2 Function Documentation

5.21.2.1 corsair_kbcopy()

```
void corsair_kbcopy (
    unsigned char * kbinput,
    int endpoint,
    const unsigned char * urbinput )
```

Definition at line 394 of file keymap.c.

References N_KEYBYTES_HW.

Referenced by os_inputmain().

```
394                                     {
395     if(endpoint == 2 || endpoint == -2){
396         if(urbinput[0] != 3)
397             return;
398         urbinput++;
399     }
400     memcpy(kbinput, urbinput, N_KEYBYTES_HW);
401 }
```

Here is the caller graph for this function:



5.21.2.2 corsair_mousecopy()

```
void corsair_mousecopy (
    unsigned char * kbinput,
    int endpoint,
    const unsigned char * urbinput )
```

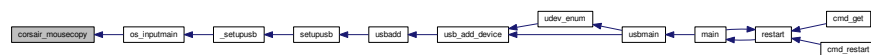
Definition at line 403 of file keymap.c.

References `BUTTON_HID_COUNT`, `CLEAR_KEYBIT`, `MOUSE_BUTTON_FIRST`, `N_BUTTONS_HW`, and `SET_KEYBIT`.

Referenced by `os_inputmain()`.

```
403                                     {
404     if(endpoint == 2 || endpoint == -2){
405         if(urbinput[0] != 3)
406             return;
407         urbinput++;
408     }
409     for(int bit = BUTTON_HID_COUNT; bit < N_BUTTONS_HW; bit++){
410         int byte = bit / 8;
411         uchar test = 1 << (bit % 8);
412         if(urbinput[byte] & test)
413             SET_KEYBIT(kbinput, MOUSE_BUTTON_FIRST + bit);
414         else
415             CLEAR_KEYBIT(kbinput, MOUSE_BUTTON_FIRST + bit);
416     }
417 }
```

Here is the caller graph for this function:



5.21.2.3 hid_kb_translate()

```
void hid_kb_translate (
    unsigned char * kbinput,
    int endpoint,
    int length,
    const unsigned char * urbinput )
```

Definition at line 223 of file keymap.c.

References `ckb_warn`, `CLEAR_KEYBIT`, and `SET_KEYBIT`.

Referenced by `os_inputmain()`.

```

223                                     {
224     if(length < 1)
225         return;
226     // LUT for HID -> Corsair scancodes (-1 for no scan code, -2 for currently unsupported)
227     // Modified from Linux drivers/hid/usbhid/usbkbd.c, key codes replaced with array indices and K95 keys
    added
228     static const short hid_codes[256] = {
229         -1, -1, -1, -1, 37, 54, 52, 39, 27, 40, 41, 42, 32, 43, 44, 45,
230         56, 55, 33, 34, 25, 28, 38, 29, 31, 53, 26, 51, 30, 50, 13, 14,
231         15, 16, 17, 18, 19, 20, 21, 22, 82, 0, 86, 24, 64, 23, 84, 35,
232         79, 80, 81, 46, 47, 12, 57, 58, 59, 36, 1, 2, 3, 4, 5, 6,
233         7, 8, 9, 10, 11, 72, 73, 74, 75, 76, 77, 78, 87, 88, 89, 95,
234         93, 94, 92, 102, 103, 104, 105, 106, 107, 115, 116, 117, 112, 113, 114, 108,
235         109, 110, 118, 119, 49, 69, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2,
236         -2, -2, -2, -2, -2, -2, -2, -2, 98, -2, -2, -2, -2, -2, 97,
237         130, 131, -1, -1, -1, -2, -1, -2, -2, -2, -2, -2, -2, -1, -1, -1,
238         -2, -2, -2, -2, -2, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
239         -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
240         -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
241         -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -3, -1, -1, -1, // <- -3 = non-RGB
    program key
242     120, 121, 122, 123, 124, 125, 126, 127, 128, 129, 136, 137, 138, 139, 140, 141,
243     60, 48, 62, 61, 91, 90, 67, 68, 142, 143, 99, 101, -2, 130, 131, 97,
244     -2, 133, 134, 135, -2, 96, -2, 132, -2, -2, 71, 71, 71, 71, -1, -1,
245 };
246 switch(endpoint){
247 case 1:
248 case -1:
249     // EP 1: 6KRO input (RGB and non-RGB)
250     // Clear previous input
251     for(int i = 0; i < 256; i++){
252         if(hid_codes[i] >= 0)
253             CLEAR_KEYBIT(kbinput, hid_codes[i]);
254     }
255     // Set new input
256     for(int i = 0; i < 8; i++){
257         if((urbinput[0] >> i) & 1)
258             SET_KEYBIT(kbinput, hid_codes[i + 224]);
259     }
260     for(int i = 2; i < length; i++){
261         if(urbinput[i] > 3){
262             int scan = hid_codes[urbinput[i]];
263             if(scan >= 0)
264                 SET_KEYBIT(kbinput, scan);
265             else
266                 ckb_warn("Got unknown key press %d on EP 1\n", urbinput[i]);
267         }
268     }
269     break;
270 case -2:
271     // EP 2 RGB: NKRO input
272     if(urbinput[0] == 1){
273         // Type 1: standard key
274         if(length != 21)
275             return;
276         for(int bit = 0; bit < 8; bit++){
277             if((urbinput[1] >> bit) & 1)
278                 SET_KEYBIT(kbinput, hid_codes[bit + 224]);
279             else
280                 CLEAR_KEYBIT(kbinput, hid_codes[bit + 224]);
281         }
282         for(int byte = 0; byte < 19; byte++){
283             char input = urbinput[byte + 2];
284             for(int bit = 0; bit < 8; bit++){
285                 int keybit = byte * 8 + bit;
286                 int scan = hid_codes[keybit];
287                 if((input >> bit) & 1){
288                     if(scan >= 0)
289                         SET_KEYBIT(kbinput, hid_codes[keybit]);
290                     else
291                         ckb_warn("Got unknown key press %d on EP 2\n", keybit);
292                 } else if(scan >= 0)
293                     CLEAR_KEYBIT(kbinput, hid_codes[keybit]);
294             }
295         }
296         break;
297     } else if(urbinput[0] == 2)
298         ; // Type 2: media key (fall through)
299     else
300         break; // No other known types
301 case 2:
302     // EP 2 Non-RGB: media keys
303     CLEAR_KEYBIT(kbinput, 97); // mute
304     CLEAR_KEYBIT(kbinput, 98); // stop
305     CLEAR_KEYBIT(kbinput, 99); // prev
306     CLEAR_KEYBIT(kbinput, 100); // play
307     CLEAR_KEYBIT(kbinput, 101); // next

```

```

308     CLEAR_KEYBIT(kbinput, 130);          // volup
309     CLEAR_KEYBIT(kbinput, 131);          // voldn
310     for(int i = 0; i < length; i++){
311         switch(urbinput[i]){
312             case 181:
313                 SET_KEYBIT(kbinput, 101); // next
314                 break;
315             case 182:
316                 SET_KEYBIT(kbinput, 99);  // prev
317                 break;
318             case 183:
319                 SET_KEYBIT(kbinput, 98);  // stop
320                 break;
321             case 205:
322                 SET_KEYBIT(kbinput, 100); // play
323                 break;
324             case 226:
325                 SET_KEYBIT(kbinput, 97);  // mute
326                 break;
327             case 233:
328                 SET_KEYBIT(kbinput, 130); // volup
329                 break;
330             case 234:
331                 SET_KEYBIT(kbinput, 131); // voldn
332                 break;
333         }
334     }
335     break;
336 case 3:
337     // EP 3 non-RGB: NKRO input
338     if(length != 15)
339         return;
340     for(int bit = 0; bit < 8; bit++){
341         if((urbinput[0] >> bit) & 1)
342             SET_KEYBIT(kbinput, hid_codes[bit + 224]);
343         else
344             CLEAR_KEYBIT(kbinput, hid_codes[bit + 224]);
345     }
346     for(int byte = 0; byte < 14; byte++){
347         char input = urbinput[byte + 1];
348         for(int bit = 0; bit < 8; bit++){
349             int keybit = byte * 8 + bit;
350             int scan = hid_codes[keybit];
351             if((input >> bit) & 1){
352                 if(scan >= 0)
353                     SET_KEYBIT(kbinput, hid_codes[keybit]);
354                 else
355                     ckb_warn("Got unknown key press %d on EP 3\n", keybit);
356             } else if(scan >= 0)
357                 CLEAR_KEYBIT(kbinput, hid_codes[keybit]);
358         }
359     }
360     break;
361 }
362 }

```

Here is the caller graph for this function:



5.21.2.4 hid_mouse_translate()

```

void hid_mouse_translate (
    unsigned char * kbinput,
    short * xaxis,
    short * yaxis,
    int endpoint,

```

```

    int length,
    const unsigned char * urbinput )

```

Definition at line 366 of file keymap.c.

References `BUTTON_HID_COUNT`, `CLEAR_KEYBIT`, `MOUSE_BUTTON_FIRST`, `MOUSE_EXTRA_FIRST`, and `SET_KEYBIT`.

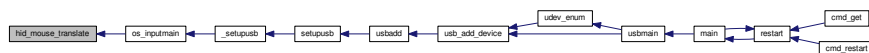
Referenced by `os_inputmain()`.

```

366
367     {
368         if((endpoint != 2 && endpoint != -2) || length < 10)
369             return;
370         // EP 2: mouse input
371         if(urbinput[0] != 1)
372             return;
373         // Byte 1 = mouse buttons (bitfield)
374         for(int bit = 0; bit < BUTTON_HID_COUNT; bit++){
375             if(urbinput[1] & (1 << bit))
376                 SET_KEYBIT(kbinput, MOUSE_BUTTON_FIRST + bit);
377             else
378                 CLEAR_KEYBIT(kbinput, MOUSE_BUTTON_FIRST + bit);
379         }
380         // Bytes 5 - 8: movement
381         *xaxis += *(short*)(urbinput + 5);
382         *yaxis += *(short*)(urbinput + 7);
383         // Byte 9: wheel
384         char wheel = urbinput[9];
385         if(wheel > 0)
386             SET_KEYBIT(kbinput, MOUSE_EXTRA_FIRST);           // wheelup
387         else
388             CLEAR_KEYBIT(kbinput, MOUSE_EXTRA_FIRST);
389         if(wheel < 0)
390             SET_KEYBIT(kbinput, MOUSE_EXTRA_FIRST + 1);       // wheeldn
391         else
392             CLEAR_KEYBIT(kbinput, MOUSE_EXTRA_FIRST + 1);
393     }

```

Here is the caller graph for this function:



5.21.3 Variable Documentation

5.21.3.1 keymap

```
const key keymap[(((152+ 3+ 12)+ 25)+ 11)]
```

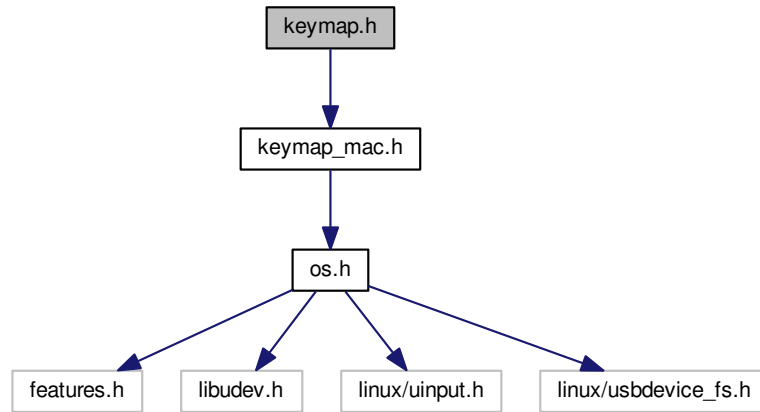
Definition at line 5 of file keymap.c.

Referenced by `_cmd_get()`, `_cmd_macro()`, `cmd_bind()`, `cmd_rebind()`, `cmd_rgb()`, `initbind()`, `inputupdate_keys()`, `nprintkey()`, `printrgb()`, `readcmd()`, and `setactive_kb()`.

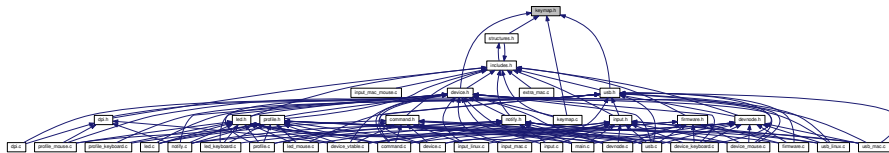
5.22 keymap.h File Reference

```
#include "keymap_mac.h"
```

Include dependency graph for keymap.h:



This graph shows which files directly or indirectly include this file:



Data Structures

- struct [key](#)

Macros

- #define [KEY_NONE](#) -1
- #define [KEY_CORSAIR](#) -2
- #define [KEY_UNBOUND](#) -3
- #define [BTN_WHEELUP](#) 0x1f01
- #define [BTN_WHEELDOWN](#) 0x1f02
- #define [KEY_BACKSLASH_ISO](#) [KEY_BACKSLASH](#)
- #define [N_KEYS_HW](#) 152
- #define [N_KEYBYTES_HW](#) $((N_KEYS_HW + 7) / 8)$
- #define [N_KEY_ZONES](#) 3
- #define [N_KEYS_EXTRA](#) 12
- #define [N_BUTTONS_HW](#) 20
- #define [N_BUTTONS_EXTENDED](#) 25

- `#define MOUSE_BUTTON_FIRST (N_KEYS_HW + N_KEY_ZONES + N_KEYS_EXTRA)`
- `#define MOUSE_EXTRA_FIRST (MOUSE_BUTTON_FIRST + N_BUTTONS_HW)`
- `#define N_KEYS_INPUT (MOUSE_BUTTON_FIRST + N_BUTTONS_EXTENDED)`
- `#define N_KEYBYTES_INPUT ((N_KEYS_INPUT + 7) / 8)`
- `#define LED_MOUSE N_KEYS_HW`
- `#define N_MOUSE_ZONES 5`
- `#define N_MOUSE_ZONES_EXTENDED 11`
- `#define LED_DPI (LED_MOUSE + 2)`
- `#define N_KEYS_EXTENDED (N_KEYS_INPUT + N_MOUSE_ZONES_EXTENDED)`
- `#define N_KEYBYTES_EXTENDED ((N_KEYS_EXTENDED + 7) / 8)`
- `#define SCAN_SILENT 0x8000`
- `#define SCAN_KBD 0`
- `#define SCAN_MOUSE 0x1000`

Functions

- void `hid_kb_translate` (unsigned char *kbinput, int endpoint, int length, const unsigned char *urbinput)
- void `hid_mouse_translate` (unsigned char *kbinput, short *xaxis, short *yaxis, int endpoint, int length, const unsigned char *urbinput)
- void `corsair_kbcopy` (unsigned char *kbinput, int endpoint, const unsigned char *urbinput)
- void `corsair_mousecopy` (unsigned char *kbinput, int endpoint, const unsigned char *urbinput)

Variables

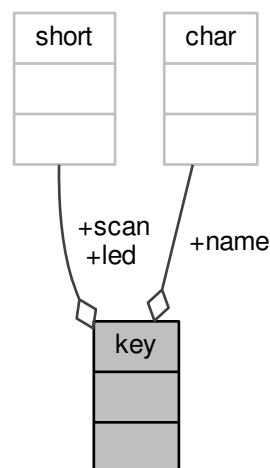
- const `key keymap` [((((152+3+12)+25)+11)]

5.22.1 Data Structure Documentation

5.22.1.1 struct key

Definition at line 49 of file keymap.h.

Collaboration diagram for key:



Data Fields

short	led	
const char *	name	
short	scan	

5.22.2 Macro Definition Documentation

5.22.2.1 BTN_WHEELDOWN

```
#define BTN_WHEELDOWN 0x1f02
```

Definition at line 13 of file keymap.h.

Referenced by os_keypress().

5.22.2.2 BTN_WHEELUP

```
#define BTN_WHEELUP 0x1f01
```

Definition at line 12 of file keymap.h.

Referenced by os_keypress().

5.22.2.3 KEY_BACKSLASH_ISO

```
#define KEY_BACKSLASH_ISO KEY_BACKSLASH
```

Definition at line 20 of file keymap.h.

5.22.2.4 KEY_CORSAIR

```
#define KEY_CORSAIR -2
```

Definition at line 8 of file keymap.h.

5.22.2.5 KEY_NONE

```
#define KEY_NONE -1
```

Definition at line 7 of file keymap.h.

5.22.2.6 KEY_UNBOUND

```
#define KEY_UNBOUND -3
```

Definition at line 9 of file keymap.h.

Referenced by cmd_unbind().

5.22.2.7 LED_DPI

```
#define LED_DPI (LED_MOUSE + 2)
```

Definition at line 43 of file keymap.h.

Referenced by loadrgb_mouse(), and savergb_mouse().

5.22.2.8 LED_MOUSE

```
#define LED_MOUSE N_KEYS_HW
```

Definition at line 39 of file keymap.h.

Referenced by isblack(), loaddpi(), loadrgb_mouse(), rgbcmp(), savedpi(), savergb_mouse(), and updatergb_mouse().

5.22.2.9 MOUSE_BUTTON_FIRST

```
#define MOUSE_BUTTON_FIRST (N_KEYS_HW + N_KEY_ZONES + N_KEYS_EXTRA)
```

Definition at line 33 of file keymap.h.

Referenced by corsair_mousecopy(), and hid_mouse_translate().

5.22.2.10 MOUSE_EXTRA_FIRST

```
#define MOUSE_EXTRA_FIRST (MOUSE_BUTTON_FIRST + N_BUTTONS_HW)
```

Definition at line 34 of file keymap.h.

Referenced by hid_mouse_translate().

5.22.2.11 N_BUTTONS_EXTENDED

```
#define N_BUTTONS_EXTENDED 25
```

Definition at line 32 of file keymap.h.

5.22.2.12 N_BUTTONS_HW

```
#define N_BUTTONS_HW 20
```

Definition at line 31 of file keymap.h.

Referenced by corsair_mousecopy().

5.22.2.13 N_KEY_ZONES

```
#define N_KEY_ZONES 3
```

Definition at line 27 of file keymap.h.

5.22.2.14 N_KEYBYTES_EXTENDED

```
#define N_KEYBYTES_EXTENDED ((N_KEYS_EXTENDED + 7) / 8)
```

Definition at line 46 of file keymap.h.

5.22.2.15 N_KEYBYTES_HW

```
#define N_KEYBYTES_HW ((N_KEYS_HW + 7) / 8)
```

Definition at line 25 of file keymap.h.

Referenced by corsair_kbcopy().

5.22.2.16 N_KEYBYTES_INPUT

```
#define N_KEYBYTES_INPUT ((N_KEYS_INPUT + 7) / 8)
```

Definition at line 37 of file keymap.h.

Referenced by `_cmd_macro()`, `inputupdate()`, `inputupdate_keys()`, and `macromask()`.

5.22.2.17 N_KEYS_EXTENDED

```
#define N_KEYS_EXTENDED (N_KEYS_INPUT + N_MOUSE_ZONES_EXTENDED)
```

Definition at line 45 of file keymap.h.

Referenced by `printrgb()`, and `readcmd()`.

5.22.2.18 N_KEYS_EXTRA

```
#define N_KEYS_EXTRA 12
```

Definition at line 29 of file keymap.h.

5.22.2.19 N_KEYS_HW

```
#define N_KEYS_HW 152
```

Definition at line 24 of file keymap.h.

Referenced by `loadrgb_kb()`, `makergb_512()`, `rgbcmp()`, and `setactive_kb()`.

5.22.2.20 N_KEYS_INPUT

```
#define N_KEYS_INPUT (MOUSE_BUTTON_FIRST + N_BUTTONS_EXTENDED)
```

Definition at line 36 of file keymap.h.

Referenced by `_cmd_get()`, `_cmd_macro()`, `cmd_bind()`, `cmd_notify()`, `cmd_rebind()`, `cmd_unbind()`, `initbind()`, and `inputupdate_keys()`.

5.22.2.21 N_MOUSE_ZONES

```
#define N_MOUSE_ZONES 5
```

Definition at line 40 of file keymap.h.

Referenced by isblack(), loaddpi(), rgbcmp(), savedpi(), and updatergb_mouse().

5.22.2.22 N_MOUSE_ZONES_EXTENDED

```
#define N_MOUSE_ZONES_EXTENDED 11
```

Definition at line 41 of file keymap.h.

5.22.2.23 SCAN_KBD

```
#define SCAN_KBD 0
```

Definition at line 57 of file keymap.h.

5.22.2.24 SCAN_MOUSE

```
#define SCAN_MOUSE 0x1000
```

Definition at line 58 of file keymap.h.

Referenced by os_keypress().

5.22.2.25 SCAN_SILENT

```
#define SCAN_SILENT 0x8000
```

Definition at line 56 of file keymap.h.

Referenced by inputupdate_keys().

5.22.3 Function Documentation

5.22.3.1 corsair_kbcopy()

```
void corsair_kbcopy (
    unsigned char * kbinput,
    int endpoint,
    const unsigned char * urbinput )
```

Definition at line 394 of file keymap.c.

References `N_KEYBYTES_HW`.

Referenced by `os_inputmain()`.

```
394                                     {
395     if(endpoint == 2 || endpoint == -2){
396         if(urbinput[0] != 3)
397             return;
398         urbinput++;
399     }
400     memcpy(kbinput, urbinput, N_KEYBYTES_HW);
401 }
```

Here is the caller graph for this function:



5.22.3.2 corsair_mousecopy()

```
void corsair_mousecopy (
    unsigned char * kbinput,
    int endpoint,
    const unsigned char * urbinput )
```

Definition at line 403 of file keymap.c.

References `BUTTON_HID_COUNT`, `CLEAR_KEYBIT`, `MOUSE_BUTTON_FIRST`, `N_BUTTONS_HW`, and `SET_KEYBIT`.

Referenced by `os_inputmain()`.

```
403                                     {
404     if(endpoint == 2 || endpoint == -2){
405         if(urbinput[0] != 3)
406             return;
407         urbinput++;
408     }
409     for(int bit = BUTTON_HID_COUNT; bit < N_BUTTONS_HW; bit++){
410         int byte = bit / 8;
411         uchar test = 1 << (bit % 8);
412         if(urbinput[byte] & test)
413             SET_KEYBIT(kbinput, MOUSE_BUTTON_FIRST + bit);
414         else
415             CLEAR_KEYBIT(kbinput, MOUSE_BUTTON_FIRST + bit);
416     }
417 }
```

Here is the caller graph for this function:



5.22.3.3 hid_kb_translate()

```
void hid_kb_translate (
    unsigned char * kbinput,
    int endpoint,
    int length,
    const unsigned char * urbinput )
```

Definition at line 223 of file keymap.c.

References `ckb_warn`, `CLEAR_KEYBIT`, and `SET_KEYBIT`.

Referenced by `os_inputmain()`.

```
223                                                                 {
224     if(length < 1)
225         return;
226     // LUT for HID -> Corsair scancodes (-1 for no scan code, -2 for currently unsupported)
227     // Modified from Linux drivers/hid/usbhid/usbkbd.c, key codes replaced with array indices and K95 keys
    added
228     static const short hid_codes[256] = {
229         -1, -1, -1, -1, 37, 54, 52, 39, 27, 40, 41, 42, 32, 43, 44, 45,
230         56, 55, 33, 34, 25, 28, 38, 29, 31, 53, 26, 51, 30, 50, 13, 14,
231         15, 16, 17, 18, 19, 20, 21, 22, 82, 0, 86, 24, 64, 23, 84, 35,
232         79, 80, 81, 46, 47, 12, 57, 58, 59, 36, 1, 2, 3, 4, 5, 6,
233         7, 8, 9, 10, 11, 72, 73, 74, 75, 76, 77, 78, 87, 88, 89, 95,
234         93, 94, 92, 102, 103, 104, 105, 106, 107, 115, 116, 117, 112, 113, 114, 108,
235         109, 110, 118, 119, 49, 69, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2,
236         -2, -2, -2, -2, -2, -2, -2, -2, 98, -2, -2, -2, -2, -2, -2, 97,
237         130, 131, -1, -1, -1, -2, -1, -2, -2, -2, -2, -2, -2, -1, -1, -1,
238         -2, -2, -2, -2, -2, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
239         -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
240         -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
241         -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -3, -1, -1, -1, // <- -3 = non-RGB
    program key
242         120, 121, 122, 123, 124, 125, 126, 127, 128, 129, 136, 137, 138, 139, 140, 141,
243         60, 48, 62, 61, 91, 90, 67, 68, 142, 143, 99, 101, -2, 130, 131, 97,
244         -2, 133, 134, 135, -2, 96, -2, 132, -2, -2, 71, 71, 71, 71, -1, -1,
245     };
246     switch(endpoint){
247     case 1:
248     case -1:
249         // EP 1: 6KRO input (RGB and non-RGB)
250         // Clear previous input
251         for(int i = 0; i < 256; i++){
252             if(hid_codes[i] >= 0)
253                 CLEAR_KEYBIT(kbinput, hid_codes[i]);
254         }
255         // Set new input
256         for(int i = 0; i < 8; i++){
257             if((urbinput[0] >> i) & 1)
258                 SET_KEYBIT(kbinput, hid_codes[i + 224]);
259         }
260         for(int i = 2; i < length; i++){
261             if(urbinput[i] > 3){
262                 int scan = hid_codes[urbinput[i]];
263                 if(scan >= 0)
264                     SET_KEYBIT(kbinput, scan);
265             }
266             else
267                 ckb_warn("Got unknown key press %d on EP 1\n", urbinput[i]);
268         }
269         break;
270     case -2:
271         // EP 2 RGB: NKRO input
272         if(urbinput[0] == 1){
273             // Type 1: standard key
274             if(length != 21)
275                 return;
276             for(int bit = 0; bit < 8; bit++){
277                 if((urbinput[1] >> bit) & 1)
278                     SET_KEYBIT(kbinput, hid_codes[bit + 224]);
279             }
280             else
281                 CLEAR_KEYBIT(kbinput, hid_codes[bit + 224]);
282         }
283         for(int byte = 0; byte < 19; byte++){
284             char input = urbinput[byte + 2];
285             for(int bit = 0; bit < 8; bit++){
286                 int keybit = byte * 8 + bit;
```

```

286         int scan = hid_codes[keybit];
287         if((input >> bit) & 1){
288             if(scan >= 0)
289                 SET_KEYBIT(kbinput, hid_codes[keybit]);
290             else
291                 ckb_warn("Got unknown key press %d on EP 2\n", keybit);
292         } else if(scan >= 0)
293             CLEAR_KEYBIT(kbinput, hid_codes[keybit]);
294     }
295 }
296 break;
297 } else if(urbinput[0] == 2)
298     ; // Type 2: media key (fall through)
299 else
300     break; // No other known types
301 case 2:
302     // EP 2 Non-RGB: media keys
303     CLEAR_KEYBIT(kbinput, 97); // mute
304     CLEAR_KEYBIT(kbinput, 98); // stop
305     CLEAR_KEYBIT(kbinput, 99); // prev
306     CLEAR_KEYBIT(kbinput, 100); // play
307     CLEAR_KEYBIT(kbinput, 101); // next
308     CLEAR_KEYBIT(kbinput, 130); // volup
309     CLEAR_KEYBIT(kbinput, 131); // voldn
310     for(int i = 0; i < length; i++){
311         switch(urbinput[i]){
312             case 181:
313                 SET_KEYBIT(kbinput, 101); // next
314                 break;
315             case 182:
316                 SET_KEYBIT(kbinput, 99); // prev
317                 break;
318             case 183:
319                 SET_KEYBIT(kbinput, 98); // stop
320                 break;
321             case 205:
322                 SET_KEYBIT(kbinput, 100); // play
323                 break;
324             case 226:
325                 SET_KEYBIT(kbinput, 97); // mute
326                 break;
327             case 233:
328                 SET_KEYBIT(kbinput, 130); // volup
329                 break;
330             case 234:
331                 SET_KEYBIT(kbinput, 131); // voldn
332                 break;
333         }
334     }
335     break;
336 case 3:
337     // EP 3 non-RGB: NKRO input
338     if(length != 15)
339         return;
340     for(int bit = 0; bit < 8; bit++){
341         if((urbinput[0] >> bit) & 1)
342             SET_KEYBIT(kbinput, hid_codes[bit + 224]);
343         else
344             CLEAR_KEYBIT(kbinput, hid_codes[bit + 224]);
345     }
346     for(int byte = 0; byte < 14; byte++){
347         char input = urbinput[byte + 1];
348         for(int bit = 0; bit < 8; bit++){
349             int keybit = byte * 8 + bit;
350             int scan = hid_codes[keybit];
351             if((input >> bit) & 1){
352                 if(scan >= 0)
353                     SET_KEYBIT(kbinput, hid_codes[keybit]);
354                 else
355                     ckb_warn("Got unknown key press %d on EP 3\n", keybit);
356             } else if(scan >= 0)
357                 CLEAR_KEYBIT(kbinput, hid_codes[keybit]);
358         }
359     }
360     break;
361 }
362 }

```

Here is the caller graph for this function:



5.22.3.4 hid_mouse_translate()

```

void hid_mouse_translate (
    unsigned char * kbinput,
    short * xaxis,
    short * yaxis,
    int endpoint,
    int length,
    const unsigned char * urbinput )

```

Definition at line 366 of file keymap.c.

References `BUTTON_HID_COUNT`, `CLEAR_KEYBIT`, `MOUSE_BUTTON_FIRST`, `MOUSE_EXTRA_FIRST`, and `SET_KEYBIT`.

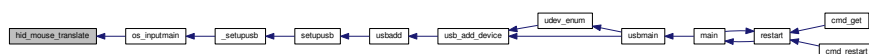
Referenced by `os_inputmain()`.

```

366
367     {
368         if((endpoint != 2 && endpoint != -2) || length < 10)
369             return;
370         // EP 2: mouse input
371         if(urbinput[0] != 1)
372             return;
373         // Byte 1 = mouse buttons (bitfield)
374         for(int bit = 0; bit < BUTTON_HID_COUNT; bit++){
375             if(urbinput[1] & (1 << bit))
376                 SET_KEYBIT(kbinput, MOUSE_BUTTON_FIRST + bit);
377             else
378                 CLEAR_KEYBIT(kbinput, MOUSE_BUTTON_FIRST + bit);
379         }
380         // Bytes 5 - 8: movement
381         *xaxis += *(short*)(urbinput + 5);
382         *yaxis += *(short*)(urbinput + 7);
383         // Byte 9: wheel
384         char wheel = urbinput[9];
385         if(wheel > 0)
386             SET_KEYBIT(kbinput, MOUSE_EXTRA_FIRST); // wheelup
387         else
388             CLEAR_KEYBIT(kbinput, MOUSE_EXTRA_FIRST);
389         if(wheel < 0)
390             SET_KEYBIT(kbinput, MOUSE_EXTRA_FIRST + 1); // wheeldn
391         else
392             CLEAR_KEYBIT(kbinput, MOUSE_EXTRA_FIRST + 1);
393     }

```

Here is the caller graph for this function:



5.22.4 Variable Documentation

5.22.4.1 keymap

```
const key keymap[((152+ 3+ 12)+ 25)+ 11])
```

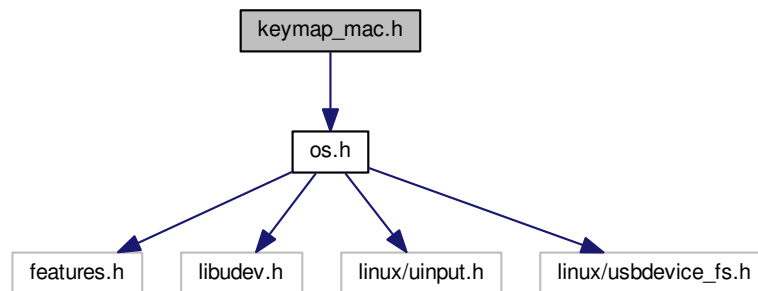
Definition at line 5 of file keymap.c.

Referenced by `_cmd_get()`, `_cmd_macro()`, `cmd_bind()`, `cmd_rebind()`, `cmd_rgb()`, `initbind()`, `inputupdate_keys()`, `nprintkey()`, `printrgb()`, `readcmd()`, and `setactive_kb()`.

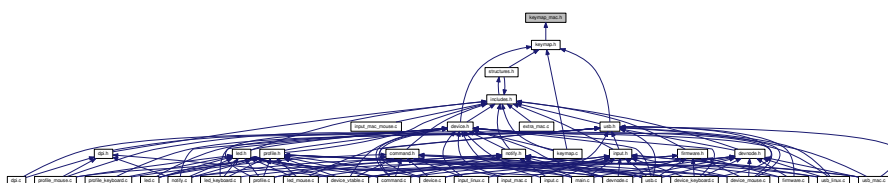
5.23 keymap_mac.h File Reference

```
#include "os.h"
```

Include dependency graph for keymap_mac.h:



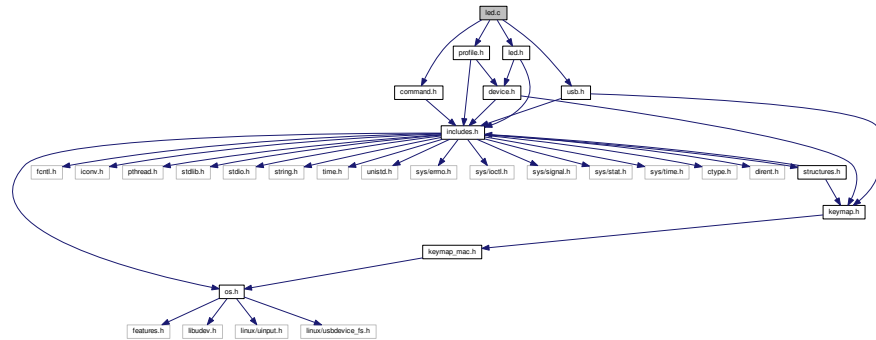
This graph shows which files directly or indirectly include this file:



5.24 led.c File Reference

```
#include "command.h"
#include "led.h"
#include "profile.h"
#include "usb.h"
```

Include dependency graph for led.c:



Functions

- void `cmd_rgb` (`usbdevice` *kb, `usbmode` *mode, int dummy, int keyindex, const char *code)
- static `uchar` `iselect` (const char *led)
- void `cmd_ion` (`usbdevice` *kb, `usbmode` *mode, int dummy1, int dummy2, const char *led)
- void `cmd_iauto` (`usbdevice` *kb, `usbmode` *mode, int dummy1, int dummy2, const char *led)
- void `cmd_inotify` (`usbdevice` *kb, `usbmode` *mode, int nnumber, int dummy, const char *led)
- static int `has_key` (const char *name, const `usbdevice` *kb)
- char * `printrgb` (const `lighting` *light, const `usbdevice` *kb)

5.24.1 Function Documentation

5.24.1.1 `cmd_iauto()`

```
void cmd_iauto (
    usbdevice * kb,
    usbmode * mode,
    int dummy1,
    int dummy2,
    const char * led )
```

Definition at line 54 of file led.c.

References `usbmode::ioff`, `usbmode::ion`, `iselect()`, and `usbdevice::vtable`.

```

54
55     uchar bits = iselect(led);
56     // Remove the bits from both ioff and ion
57     mode->ioff &= ~bits;
58     mode->ion &= ~bits;
59     kb->vtable->updateindicators(kb, 0);
60 }

```

Here is the call graph for this function:



5.24.1.2 cmd_inotify()

```

void cmd_inotify (
    usbdevice * kb,
    usbmode * mode,
    int nnumber,
    int dummy,
    const char * led )

```

Definition at line 62 of file led.c.

References `usbmode::inotify`, and `iselect()`.

```

62
63     uchar bits = iselect(led);
64     if(strstr(led, ":off"))
65         // Turn notifications for these bits off
66         mode->inotify[nnumber] &= ~bits;
67     else
68         // Turn notifications for these bits on
69         mode->inotify[nnumber] |= bits;
70 }

```

Here is the call graph for this function:



5.24.1.3 cmd_ioff()

```
void cmd_ioff (
    usbdevice * kb,
    usbmode * mode,
    int dummy1,
    int dummy2,
    const char * led )
```

Definition at line 38 of file led.c.

References `usbmode::ioff`, `usbmode::ion`, `iselect()`, and `usbdevice::vtable`.

```
38                                     {
39     uchar bits = iselect(led);
40     // Add the bits to ioff, remove them from ion
41     mode->ioff |= bits;
42     mode->ion  &= ~bits;
43     kb->vtable->updateindicators(kb, 0);
44 }
```

Here is the call graph for this function:



5.24.1.4 cmd_ion()

```
void cmd_ion (
    usbdevice * kb,
    usbmode * mode,
    int dummy1,
    int dummy2,
    const char * led )
```

Definition at line 46 of file led.c.

References `usbmode::ioff`, `usbmode::ion`, `iselect()`, and `usbdevice::vtable`.

```
46                                     {
47     uchar bits = iselect(led);
48     // Remove the bits from ioff, add them to ion
49     mode->ioff &= ~bits;
50     mode->ion  |= bits;
51     kb->vtable->updateindicators(kb, 0);
52 }
```

Here is the call graph for this function:



5.24.1.5 cmd_rgb()

```

void cmd_rgb (
    usbdevice * kb,
    usbmode * mode,
    int dummy,
    int keyindex,
    const char * code )
  
```

Definition at line 6 of file led.c.

References `lighting::b`, `lighting::g`, `keymap`, `key::led`, `usbmode::light`, `lighting::r`, and `lighting::sidelight`.

```

6                                     {
7     int index = keymap[keyindex].led;
8     if(index < 0) {
9         if (index == -2){           // Process strafe sidelights
10             uchar sideshine;
11             if (sscanf(code, "%2hhx",&sideshine)) // monochromatic
12                 mode->light.sidelight = sideshine;
13         }
14         return;
15     }
16     uchar r, g, b;
17     if(sscanf(code, "%2hhx%2hhx%2hhx", &r, &g, &b) == 3){
18         mode->light.r[index] = r;
19         mode->light.g[index] = g;
20         mode->light.b[index] = b;
21     }
22 }
  
```

5.24.1.6 has_key()

```

static int has_key (
    const char * name,
    const usbdevice * kb ) [static]
  
```

Definition at line 73 of file led.c.

References `IS_K65`, `IS_K95`, `IS_MOUSE`, `IS_SABRE`, `IS_SCIMITAR`, `usbdevice::product`, and `usbdevice::vendor`.

Referenced by `printrgb()`.

```

73                                     {
74     if(!name)
75         return 0;
76     if(IS_MOUSE(kb->vendor, kb->product)){
77         // Mice only have the RGB zones
78         if((IS_SABRE(kb) || IS_SCIMITAR(kb)) && !strcmp(name, "wheel"))
79             return 1;
80         if(IS_SCIMITAR(kb) && !strcmp(name, "thumb"))
81             return 1;
82         if(strstr(name, "dpi") == name || !strcmp(name, "front") || !strcmp(name, "back"))
83             return 1;
84         return 0;
85     } else {
86         // But keyboards don't have them at all
87         if(strstr(name, "dpi") == name || !strcmp(name, "front") || !strcmp(name, "back") || !strcmp(name,
"wheel") || !strcmp(name, "thumb"))
88             return 0;
89         // Only K95 has G keys and M keys (G1 - G18, MR, M1 - M3)
90         if(!IS_K95(kb) && ((name[0] == 'g' && name[1] >= '1' && name[1] <= '9') || (name[0] == 'm' &&
(name[1] == 'r' || name[1] == '1' || name[1] == '2' || name[1] == '3'))))
91             return 0;
92         // Only K65 has lights on VolUp/VolDn
93         if(!IS_K65(kb) && (!strcmp(name, "volup") || !strcmp(name, "voldn")))
94             return 0;
95         // K65 lacks numpad and media buttons
96         if(IS_K65(kb) && (strstr(name, "num") == name || !strcmp(name, "stop") || !strcmp(name, "prev
") || !strcmp(name, "play") || !strcmp(name, "next")))
97             return 0;
98     }
99     return 1;
100 }

```

Here is the caller graph for this function:



5.24.1.7 iselect()

```

static uchar iselect (
    const char * led ) [static]

```

Definition at line 25 of file led.c.

References `I_CAPS`, `I_NUM`, and `I_SCROLL`.

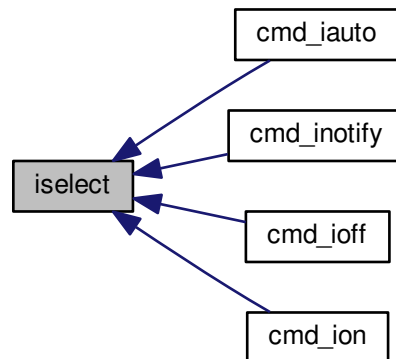
Referenced by `cmd_iauto()`, `cmd_inotify()`, `cmd_ioff()`, and `cmd_ion()`.

```

25                                     {
26     int result = 0;
27     if(!strncmp(led, "num", 3) || strstr(led, "num"))
28         result |= I_NUM;
29     if(!strncmp(led, "caps", 4) || strstr(led, "caps"))
30         result |= I_CAPS;
31     if(!strncmp(led, "scroll", 6) || strstr(led, "scroll"))
32         result |= I_SCROLL;
33     if(!strncmp(led, "all", 3) || strstr(led, "all"))
34         result |= I_NUM | I_CAPS | I_SCROLL;
35     return result;
36 }

```

Here is the caller graph for this function:



5.24.1.8 printrgb()

```
char* printrgb (
    const lighting * light,
    const usbdevice * kb )
```

Definition at line 102 of file led.c.

References `lighting::b`, `lighting::g`, `has_key()`, `keymap`, `key::led`, `N_KEYS_EXTENDED`, `key::name`, and `lighting::r`.

Referenced by `_cmd_get()`.

```

102                                     {
103     uchar r[N_KEYS_EXTENDED], g[N_KEYS_EXTENDED], b[
104     N_KEYS_EXTENDED];
105     const uchar* mr = light->r;
106     const uchar* mg = light->g;
107     const uchar* mb = light->b;
108     for(int i = 0; i < N_KEYS_EXTENDED; i++){
109         // Translate the key index to an RGB index using the key map
110         int k = keymap[i].led;
111         if(k < 0)
112             continue;
113         r[i] = mr[k];
114         g[i] = mg[k];
115         b[i] = mb[k];
116     }
117     // Make a buffer to track key names and to filter out duplicates
118     char names[N_KEYS_EXTENDED][11];
119     for(int i = 0; i < N_KEYS_EXTENDED; i++){
120         const char* name = keymap[i].name;
121         if(keymap[i].led < 0 || !has_key(name, kb))
122             names[i][0] = 0;
123         else
124             strncpy(names[i], name, 11);
125     }
126     // Check to make sure these aren't all the same color
127     int same = 1;
128     for(int i = 1; i < N_KEYS_EXTENDED; i++){
129         if(!names[i][0])
130             continue;

```

```

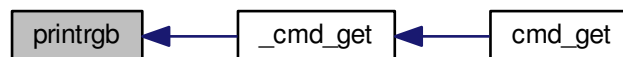
130         if(r[i] != r[0] || g[i] != g[0] || b[i] != b[0]){
131             same = 0;
132             break;
133         }
134     }
135     // If they are, just output that color
136     if(same){
137         char* buffer = malloc(7);
138         snprintf(buffer, 7, "%02x%02x%02x", r[0], g[0], b[0]);
139         return buffer;
140     }
141     const int BUFFER_LEN = 4096;    // Should be more than enough to fit all keys
142     char* buffer = malloc(BUFFER_LEN);
143     int length = 0;
144     for(int i = 0; i < N_KEYS_EXTENDED; i++){
145         if(!names[i][0])
146             continue;
147         // Print the key name
148         int newlen = 0;
149         snprintf(buffer + length, BUFFER_LEN - length, length == 0 ? "%s\n" : " %s\n", names[i], &newlen);
150         length += newlen;
151         // Look ahead to see if any other keys have this color. If so, print them here as well.
152         uchar kr = r[i], kg = g[i], kb = b[i];
153         for(int j = i + 1; j < N_KEYS_EXTENDED; j++){
154             if(!names[j][0])
155                 continue;
156             if(r[j] != kr || g[j] != kg || b[j] != kb)
157                 continue;
158             snprintf(buffer + length, BUFFER_LEN - length, "%s\n", names[j], &newlen);
159             length += newlen;
160             // Erase the key's name so it won't get printed later
161             names[j][0] = 0;
162         }
163         // Print the color
164         snprintf(buffer + length, BUFFER_LEN - length, ":%02x%02x%02x\n", kr, kg, kb, &newlen);
165         length += newlen;
166     }
167     return buffer;
168 }

```

Here is the call graph for this function:



Here is the caller graph for this function:

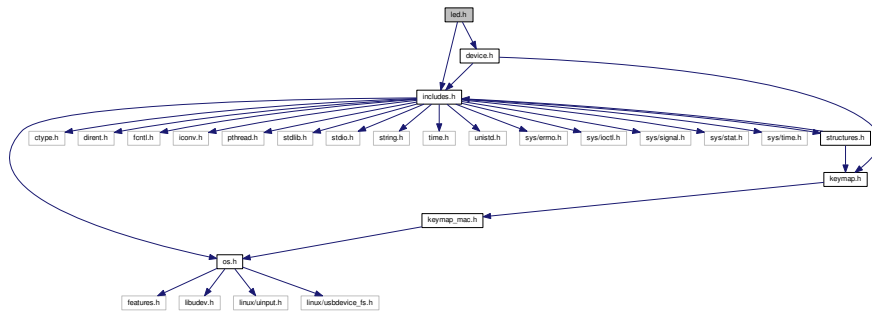


5.25 led.h File Reference

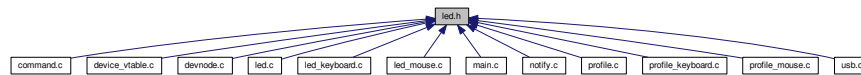
```
#include "includes.h"
```

```
#include "device.h"
```

Include dependency graph for led.h:



This graph shows which files directly or indirectly include this file:



Functions

- int `updatergb_kb` (usbdevice *kb, int force)
- int `updatergb_mouse` (usbdevice *kb, int force)
- int `savergb_kb` (usbdevice *kb, lighting *light, int mode)
- int `savergb_mouse` (usbdevice *kb, lighting *light, int mode)
- int `loadrgb_kb` (usbdevice *kb, lighting *light, int mode)
- int `loadrgb_mouse` (usbdevice *kb, lighting *light, int mode)
- char * `printrgb` (const lighting *light, const usbdevice *kb)
- void `cmd_rgb` (usbdevice *kb, usbmode *mode, int dummy, int keyindex, const char *code)
- void `cmd_ioff` (usbdevice *kb, usbmode *mode, int dummy1, int dummy2, const char *led)
- void `cmd_ion` (usbdevice *kb, usbmode *mode, int dummy1, int dummy2, const char *led)
- void `cmd_iauto` (usbdevice *kb, usbmode *mode, int dummy1, int dummy2, const char *led)
- void `cmd_inotify` (usbdevice *kb, usbmode *mode, int nnumber, int dummy, const char *led)

5.25.1 Function Documentation

5.25.1.1 cmd_iauto()

```
void cmd_iauto (
    usbdevice * kb,
    usbmode * mode,
    int dummy1,
    int dummy2,
    const char * led )
```

Definition at line 54 of file led.c.

References `usbmode::ioff`, `usbmode::ion`, `iselect()`, and `usbdevice::vtable`.

```
54                                     {
55     uchar bits = iselect(led);
56     // Remove the bits from both ioff and ion
57     mode->ioff &= ~bits;
58     mode->ion &= ~bits;
59     kb->vtable->updateindicators(kb, 0);
60 }
```

Here is the call graph for this function:



5.25.1.2 cmd_inotify()

```
void cmd_inotify (
    usbdevice * kb,
    usbmode * mode,
    int nnumber,
    int dummy,
    const char * led )
```

Definition at line 62 of file led.c.

References `usbmode::inotify`, and `iselect()`.

```
62                                     {
63     uchar bits = iselect(led);
64     if(strstr(led, ":off"))
65         // Turn notifications for these bits off
66         mode->inotify[nnumber] &= ~bits;
67     else
68         // Turn notifications for these bits on
69         mode->inotify[nnumber] |= bits;
70 }
```

Here is the call graph for this function:



5.25.1.3 cmd_ioff()

```
void cmd_ioff (
    usbdevice * kb,
    usbmode * mode,
    int dummy1,
    int dummy2,
    const char * led )
```

Definition at line 38 of file led.c.

References `usbmode::ioff`, `usbmode::ion`, `iselect()`, and `usbdevice::vtable`.

```
38                                     {
39     uchar bits = iselect(led);
40     // Add the bits to ioff, remove them from ion
41     mode->ioff |= bits;
42     mode->ion  &= ~bits;
43     kb->vtable->updateindicators(kb, 0);
44 }
```

Here is the call graph for this function:



5.25.1.4 cmd_ion()

```
void cmd_ion (
    usbdevice * kb,
    usbmode * mode,
    int dummy1,
    int dummy2,
    const char * led )
```

Definition at line 46 of file led.c.

References `usbmode::ioff`, `usbmode::ion`, `iselect()`, and `usbdevice::vtable`.

```
46                                     {
47     uchar bits = iselect(led);
48     // Remove the bits from ioff, add them to ion
49     mode->ioff &= ~bits;
50     mode->ion  |= bits;
51     kb->vtable->updateindicators(kb, 0);
52 }
```

Here is the call graph for this function:



5.25.1.5 cmd_rgb()

```

void cmd_rgb (
    usbdevice * kb,
    usbmode * mode,
    int dummy,
    int keyindex,
    const char * code )
  
```

Definition at line 6 of file led.c.

References `lighting::b`, `lighting::g`, `keymap`, `key::led`, `usbmode::light`, `lighting::r`, and `lighting::sidelight`.

```

6                                     {
7     int index = keymap[keyindex].led;
8     if(index < 0) {
9         if (index == -2){           // Process strafe sidelights
10             uchar sideshine;
11             if (sscanf(code, "%2hhx",&sideshine)) // monochromatic
12                 mode->light.sidelight = sideshine;
13         }
14         return;
15     }
16     uchar r, g, b;
17     if(sscanf(code, "%2hhx%2hhx%2hhx", &r, &g, &b) == 3){
18         mode->light.r[index] = r;
19         mode->light.g[index] = g;
20         mode->light.b[index] = b;
21     }
22 }
  
```

5.25.1.6 loadrgb_kb()

```

int loadrgb_kb (
    usbdevice * kb,
    lighting * light,
    int mode )
  
```

Since Firmware Version 2.05 the answers for getting the stored color-maps from the hardware has changed a bit. So comparing for the correct answer cannot validate against the cmd, and has to be done against a third map.

Definition at line 181 of file led_keyboard.c.

References `lighting::b`, `ckb_err`, `usbdevice::fwversion`, `lighting::g`, `MSG_SIZE`, `N_KEYS_HW`, `lighting::r`, `usbrecv`, and `usb send`.

Referenced by `hwloadmode()`.

```

181                                     {
182     if(kb->fwversion >= 0x0120){
183         uchar data_pkt[12][MSG_SIZE] = {
184             { 0x0e, 0x14, 0x03, 0x01, 0x01, mode + 1, 0x01 },
185             { 0xff, 0x01, 60, 0 },
186             { 0xff, 0x02, 60, 0 },
187             { 0xff, 0x03, 24, 0 },
188             { 0x0e, 0x14, 0x03, 0x01, 0x01, mode + 1, 0x02 },
189             { 0xff, 0x01, 60, 0 },
190             { 0xff, 0x02, 60, 0 },
191             { 0xff, 0x03, 24, 0 },
192             { 0x0e, 0x14, 0x03, 0x01, 0x01, mode + 1, 0x03 },
193             { 0xff, 0x01, 60, 0 },
194             { 0xff, 0x02, 60, 0 },
195             { 0xff, 0x03, 24, 0 },
196         };
197         uchar in_pkt[4][MSG_SIZE] = {
198             { 0x0e, 0x14, 0x03, 0x01 },
199             { 0xff, 0x01, 60, 0 },
200             { 0xff, 0x02, 60, 0 },
201             { 0xff, 0x03, 24, 0 },
202         };
203
204
205
206
207
208         uchar cmp_pkt[4][4] = {
209             { 0x0e, 0x14, 0x03, 0x01 },
210             { 0x0e, 0xff, 0x01, 60 },
211             { 0x0e, 0xff, 0x02, 60 },
212             { 0x0e, 0xff, 0x03, 24 },
213         };
214         // Read colors
215         uchar* colors[3] = { light->r, light->g, light->b };
216         for(int clr = 0; clr < 3; clr++){
217             for(int i = 0; i < 4; i++){
218                 if(!usbrecv(kb, data_pkt[i + clr * 4], in_pkt[i]))
219                     return -1;
220                 // Make sure the first four bytes match
221                 // see comment above
222                 // if(memcmp(p, data_pkt[i + clr * 4], 4)){
223                 if (memcmp(in_pkt[i], (kb->fwversion >= 0x0205)? cmp_pkt[i] : data_pkt[i + clr * 4
224 ], 4)) {
225                     ckb_err("Bad input header\n");
226                     ckb_err("color = %d, i = %d, mode = %d\nInput (Antwort): %2.2x %2.2x %2.2x %2.2x
227 %2.2x %2.2x %2.2x %2.2x\nOutput (Frage): %2.2x %2.2x %2.2x %2.2x\n", clr, i, mode,
228 in_pkt[i][0], in_pkt[i][1], in_pkt[i][2], in_pkt[i][3], in_pkt[i][4], in_pkt[i][5],
229 in_pkt[i][6], in_pkt[i][7],
230 // data_pkt[i + clr * 4][0], data_pkt[i + clr * 4][1], data_pkt[i + clr *
231 4][2], data_pkt[i + clr * 4][3]);
232 cmp_pkt[i][0], cmp_pkt[i][1], cmp_pkt[i][2], cmp_pkt[i][3]);
233 in_pkt[2][0] = 0x99;
234 in_pkt[2][1] = 0x99;
235 in_pkt[2][2] = 0x99;
236 in_pkt[2][3] = 0x99;
237 usbrecv(kb, in_pkt[2], in_pkt[2]); // just to find it in the wireshark log
238 return -1;
239 }
240 }
241 // Copy colors to lighting. in_pkt[0] is irrelevant.
242 memcpy(colors[clr], in_pkt[1] + 4, 60);
243 memcpy(colors[clr] + 60, in_pkt[2] + 4, 60);
244 memcpy(colors[clr] + 120, in_pkt[3] + 4, 24);
245 }
246 } else {
247     uchar data_pkt[5][MSG_SIZE] = {
248         { 0x0e, 0x14, 0x02, 0x01, 0x01, mode + 1, 0 },
249         { 0xff, 0x01, 60, 0 },
250         { 0xff, 0x02, 60, 0 },
251         { 0xff, 0x03, 60, 0 },
252         { 0xff, 0x04, 36, 0 },
253     };
254     uchar in_pkt[4][MSG_SIZE] = {
255         { 0xff, 0x01, 60, 0 },
256         { 0xff, 0x02, 60, 0 },
257         { 0xff, 0x03, 60, 0 },
258         { 0xff, 0x04, 36, 0 },
259     };
260     // Write initial packet
261     if(!usbwrite(kb, data_pkt[0], 1))
262         return -1;
263     // Read colors
264     for(int i = 1; i < 5; i++){
265         if(!usbrecv(kb, data_pkt[i], in_pkt[i - 1]))
266             return -1;
267         if(memcmp(in_pkt[i - 1], data_pkt[i], 4)){
268             ckb_err("Bad input header\n");
269             return -1;
270         }
271     }
272 }

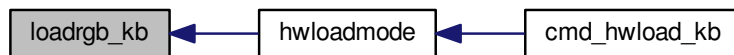
```

```

267     }
268     // Copy the data back to the mode
269     uint8_t mr[N_KEYS_HW / 2], mg[N_KEYS_HW / 2], mb[
N_KEYS_HW / 2];
270     memcpy(mr, in_pkt[0] + 4, 60);
271     memcpy(mr + 60, in_pkt[1] + 4, 12);
272     memcpy(mg, in_pkt[1] + 16, 48);
273     memcpy(mg + 48, in_pkt[2] + 4, 24);
274     memcpy(mb, in_pkt[2] + 28, 36);
275     memcpy(mb + 36, in_pkt[3] + 4, 36);
276     // Unpack LED data to 8bpc format
277     for(int i = 0; i < N_KEYS_HW; i++){
278         int i_2 = i / 2;
279         uint8_t r, g, b;
280
281         // 3-bit intensities stored in alternate nybbles.
282         if (i & 1) {
283             r = 7 - (mr[i_2] >> 4);
284             g = 7 - (mg[i_2] >> 4);
285             b = 7 - (mb[i_2] >> 4);
286         } else {
287             r = 7 - (mr[i_2] & 0x0F);
288             g = 7 - (mg[i_2] & 0x0F);
289             b = 7 - (mb[i_2] & 0x0F);
290         }
291         // Scale 3-bit values up to 8 bits.
292         light->r[i] = r << 5 | r << 2 | r >> 1;
293         light->g[i] = g << 5 | g << 2 | g >> 1;
294         light->b[i] = b << 5 | b << 2 | b >> 1;
295     }
296 }
297 return 0;
298 }

```

Here is the caller graph for this function:



5.25.1.7 loadrgb_mouse()

```

int loadrgb_mouse (
    usbdevice * kb,
    lighting * light,
    int mode )

```

Definition at line 81 of file `led_mouse.c`.

References `lighting::b`, `ckb_err`, `lighting::g`, `IS_SABRE`, `IS_SCIMITAR`, `LED_DPI`, `LED_MOUSE`, `MSG_SIZE`, `lighting::r`, and `usbrecv`.

Referenced by `cmd_hwload_mouse()`.

```

81
82     uchar data_pkt[MSG_SIZE] = { 0x0e, 0x13, 0x10, 1, 0 };
83     uchar in_pkt[MSG_SIZE] = { 0 };
84     // Load each RGB zone
85     int zonecount = IS_SCIMITAR(kb) ? 4 : IS_SABRE(kb) ? 3 : 2;
86     for(int i = 0; i < zonecount; i++){
87         if(!usbrecv(kb, data_pkt, in_pkt))
88             return -1;
89         if(memcmp(in_pkt, data_pkt, 4)){
90             ckb_err("Bad input header\n");
91             return -2;
92         }
93         // Copy data
94         int led = LED_MOUSE + i;
95         if(led >= LED_DPI)
96             led++; // Skip DPI light
97         light->r[led] = in_pkt[4];
98         light->g[led] = in_pkt[5];
99         light->b[led] = in_pkt[6];
100        // Set packet for next zone
101        data_pkt[2]++;
102    }
103    return 0;
104 }

```

Here is the caller graph for this function:



5.25.1.8 printrgb()

```

char* printrgb (
    const lighting * light,
    const usbdevice * kb )

```

Definition at line 102 of file led.c.

References `lighting::b`, `lighting::g`, `has_key()`, `keymap`, `key::led`, `N_KEYS_EXTENDED`, `key::name`, and `lighting::r`.

Referenced by `_cmd_get()`.

```

102
103     uchar r[N_KEYS_EXTENDED], g[N_KEYS_EXTENDED], b[
104     N_KEYS_EXTENDED];
105     const uchar* mr = light->r;
106     const uchar* mg = light->g;
107     const uchar* mb = light->b;
108     for(int i = 0; i < N_KEYS_EXTENDED; i++){
109         // Translate the key index to an RGB index using the key map
110         int k = keymap[i].led;
111         if(k < 0)
112             continue;
113         r[i] = mr[k];
114         g[i] = mg[k];
115         b[i] = mb[k];
116     }
117     // Make a buffer to track key names and to filter out duplicates

```

```

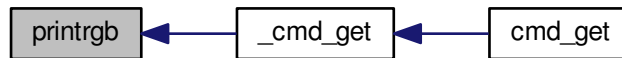
117     char names[N_KEYS_EXTENDED][11];
118     for(int i = 0; i < N_KEYS_EXTENDED; i++){
119         const char* name = keymap[i].name;
120         if(keymap[i].led < 0 || !has_key(name, kb))
121             names[i][0] = 0;
122         else
123             strncpy(names[i], name, 11);
124     }
125     // Check to make sure these aren't all the same color
126     int same = 1;
127     for(int i = 1; i < N_KEYS_EXTENDED; i++){
128         if(!names[i][0])
129             continue;
130         if(r[i] != r[0] || g[i] != g[0] || b[i] != b[0]){
131             same = 0;
132             break;
133         }
134     }
135     // If they are, just output that color
136     if(same){
137         char* buffer = malloc(7);
138         snprintf(buffer, 7, "%02x%02x%02x", r[0], g[0], b[0]);
139         return buffer;
140     }
141     const int BUFFER_LEN = 4096; // Should be more than enough to fit all keys
142     char* buffer = malloc(BUFFER_LEN);
143     int length = 0;
144     for(int i = 0; i < N_KEYS_EXTENDED; i++){
145         if(!names[i][0])
146             continue;
147         // Print the key name
148         int newlen = 0;
149         snprintf(buffer + length, BUFFER_LEN - length, length == 0 ? "%s\n" : " %s\n", names[i], &newlen);
150         length += newlen;
151         // Look ahead to see if any other keys have this color. If so, print them here as well.
152         uchar kr = r[i], kg = g[i], kb = b[i];
153         for(int j = i + 1; j < N_KEYS_EXTENDED; j++){
154             if(!names[j][0])
155                 continue;
156             if(r[j] != kr || g[j] != kg || b[j] != kb)
157                 continue;
158             snprintf(buffer + length, BUFFER_LEN - length, "%s\n", names[j], &newlen);
159             length += newlen;
160             // Erase the key's name so it won't get printed later
161             names[j][0] = 0;
162         }
163         // Print the color
164         snprintf(buffer + length, BUFFER_LEN - length, ":%02x%02x%02x\n", kr, kg, kb, &newlen);
165         length += newlen;
166     }
167     return buffer;
168 }

```

Here is the call graph for this function:



Here is the caller graph for this function:



5.25.1.9 savergb_kb()

```
int savergb_kb (
    usbdevice * kb,
    lighting * light,
    int mode )
```

Definition at line 139 of file led_keyboard.c.

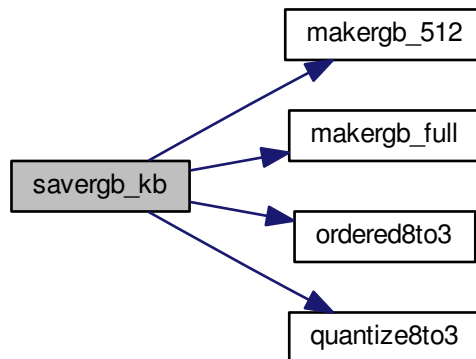
References `usbdevice::dither`, `usbdevice::fwversion`, `IS_STRAFE`, `makergb_512()`, `makergb_full()`, `MSG_SIZE`, `ordered8to3()`, `quantize8to3()`, and `usb send`.

Referenced by `cmd_hwsave_kb()`.

```

139                                     {
140     if(kb->fwversion >= 0x0120){
141         uchar data_pkt[12][MSG_SIZE] = {
142             // Red
143             { 0x7f, 0x01, 60, 0 },
144             { 0x7f, 0x02, 60, 0 },
145             { 0x7f, 0x03, 24, 0 },
146             { 0x07, 0x14, 0x03, 0x01, 0x01, mode + 1, 0x01 },
147             // Green
148             { 0x7f, 0x01, 60, 0 },
149             { 0x7f, 0x02, 60, 0 },
150             { 0x7f, 0x03, 24, 0 },
151             { 0x07, 0x14, 0x03, 0x01, 0x01, mode + 1, 0x02 },
152             // Blue
153             { 0x7f, 0x01, 60, 0 },
154             { 0x7f, 0x02, 60, 0 },
155             { 0x7f, 0x03, 24, 0 },
156             { 0x07, 0x14, 0x03, 0x01, 0x01, mode + 1, 0x03 }
157         };
158         makergb_full(light, data_pkt);
159         if(!usb send(kb, data_pkt[0], 12))
160             return -1;
161         if (IS_STRAFE(kb)){ // end save
162             uchar save_end_pkt[MSG_SIZE] = { 0x07, 0x14, 0x04, 0x01, 0x01 };
163             if(!usb send(kb, save_end_pkt, 1))
164                 return -1;
165         }
166     } else {
167         uchar data_pkt[5][MSG_SIZE] = {
168             { 0x7f, 0x01, 60, 0 },
169             { 0x7f, 0x02, 60, 0 },
170             { 0x7f, 0x03, 60, 0 },
171             { 0x7f, 0x04, 36, 0 },
172             { 0x07, 0x14, 0x02, 0x00, 0x01, mode + 1 }
173         };
174         makergb_512(light, data_pkt, kb->dither ? ordered8to3 :
quantize8to3);
175         if(!usb send(kb, data_pkt[0], 5))
176             return -1;
177     }
178     return 0;
179 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



5.25.1.10 savergb_mouse()

```

int savergb_mouse (
    usbdevice * kb,
    lighting * light,
    int mode )
  
```

Definition at line 62 of file `led_mouse.c`.

References `lighting::b`, `lighting::g`, `IS_SABRE`, `IS_SCIMITAR`, `LED_DPI`, `LED_MOUSE`, `MSG_SIZE`, `lighting::r`, and `usb send`.

Referenced by `cmd_hwsave_mouse()`.

```

62
63     uchar data_pkt[MSG_SIZE] = { 0x07, 0x13, 0x10, 1, 0 };
64     // Save each RGB zone, minus the DPI light which is sent in the DPI packets
65     int zonecount = IS_SCIMITAR(kb) ? 4 : IS_SABRE(kb) ? 3 : 2;
66     for(int i = 0; i < zonecount; i++){
67         int led = LED_MOUSE + i;
  
```

```

68     if(led >= LED_DPI)
69         led++; // Skip DPI light
70     data_pkt[4] = light->r[led];
71     data_pkt[5] = light->g[led];
72     data_pkt[6] = light->b[led];
73     if(!usbSend(kb, data_pkt, 1))
74         return -1;
75     // Set packet for next zone
76     data_pkt[2]++;
77 }
78 return 0;
79 }

```

Here is the caller graph for this function:



5.25.1.11 updatergb_kb()

```

int updatergb_kb (
    usbdevice * kb,
    int force )

```

Definition at line 77 of file led_keyboard.c.

References `usbdevice::active`, `usbprofile::currentmode`, `usbdevice::dither`, `lighting::forceupdate`, `IS_FULLRANGE`, `usbprofile::lastlight`, `usbmode::light`, `makergb_512()`, `makergb_full()`, `MSG_SIZE`, `ordered8to3()`, `usbdevice::profile`, `quantize8to3()`, `rgbcmp()`, `lighting::sidelight`, and `usbSend`.

```

77                                     {
78     if(!kb->active)
79         return 0;
80     lighting* lastlight = &kb->profile->lastlight;
81     lighting* newlight = &kb->profile->currentmode->
light;
82     // Don't do anything if the lighting hasn't changed
83     if(!force && !lastlight->forceupdate && !newlight->forceupdate
84         && !rgbcmp(lastlight, newlight) && lastlight->sidelight == newlight->
sidelight) // strafe sidelights
85         return 0;
86     lastlight->forceupdate = newlight->forceupdate = 0;
87
88     if(IS_FULLRANGE(kb)) {
89         // Update strafe sidelights if necessary
90         if(lastlight->sidelight != newlight->sidelight) {
91             uchar data_pkt[2][MSG_SIZE] = {
92                 { 0x07, 0x05, 0x08, 0x00, 0x00 },
93                 { 0x07, 0x05, 0x02, 0, 0x03 }
94             };
95             if (newlight->sidelight)
96                 data_pkt[0][4]=1; // turn on
97             if(!usbSend(kb, data_pkt[0], 2))
98                 return -1;
99         }
100         // 16.8M color lighting works fine on strafe and is the only way it actually works
101         uchar data_pkt[12][MSG_SIZE] = {
102             // Red

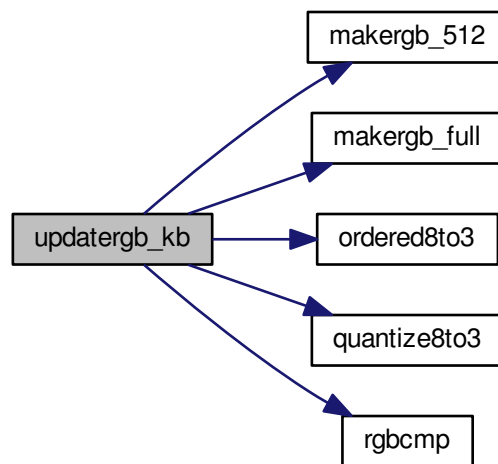
```

```

103         { 0x7f, 0x01, 0x3c, 0 },
104         { 0x7f, 0x02, 0x3c, 0 },
105         { 0x7f, 0x03, 0x18, 0 },
106         { 0x07, 0x28, 0x01, 0x03, 0x01, 0 },
107         // Green
108         { 0x7f, 0x01, 0x3c, 0 },
109         { 0x7f, 0x02, 0x3c, 0 },
110         { 0x7f, 0x03, 0x18, 0 },
111         { 0x07, 0x28, 0x02, 0x03, 0x01, 0 },
112         // Blue
113         { 0x7f, 0x01, 0x3c, 0 },
114         { 0x7f, 0x02, 0x3c, 0 },
115         { 0x7f, 0x03, 0x18, 0 },
116         { 0x07, 0x28, 0x03, 0x03, 0x02, 0 }
117     };
118     makergb_full(newlight, data_pkt);
119     if(!usbSEND(kb, data_pkt[0], 12))
120         return -1;
121 } else {
122     // On older keyboards it looks flickery and causes lighting glitches, so we don't use it.
123     uchar data_pkt[5][MSG_SIZE] = {
124         { 0x7f, 0x01, 60, 0 },
125         { 0x7f, 0x02, 60, 0 },
126         { 0x7f, 0x03, 60, 0 },
127         { 0x7f, 0x04, 36, 0 },
128         { 0x07, 0x27, 0x00, 0x00, 0xD8 }
129     };
130     makergb_512(newlight, data_pkt, kb->dither ?
ordered8to3 : quantize8to3);
131     if(!usbSEND(kb, data_pkt[0], 5))
132         return -1;
133 }
134
135 memcpy(lastlight, newlight, sizeof(lighting));
136 return 0;
137 }

```

Here is the call graph for this function:



5.25.1.12 updatergb_mouse()

```

int updatergb_mouse (
    usbdevice * kb,
    int force )

```

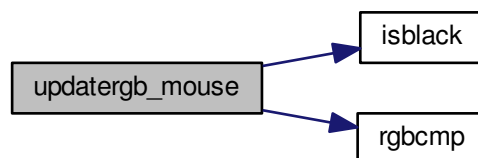
Definition at line 20 of file led_mouse.c.

References `usbdevice::active`, `lighting::b`, `usbprofile::currentmode`, `lighting::forceupdate`, `lighting::g`, `isblack()`, `usbprofile::lastlight`, `LED_MOUSE`, `usbmode::light`, `MSG_SIZE`, `N_MOUSE_ZONES`, `usbdevice::profile`, `lighting::r`, `rgbcmp()`, and `usbsend`.

```

20                                     {
21     if(!kb->active)
22         return 0;
23     lighting* lastlight = &kb->profile->lastlight;
24     lighting* newlight = &kb->profile->currentmode->
light;
25     // Don't do anything if the lighting hasn't changed
26     if(!force && !lastlight->forceupdate && !newlight->forceupdate
27         && !rgbcmp(lastlight, newlight))
28         return 0;
29     lastlight->forceupdate = newlight->forceupdate = 0;
30
31     // Send the RGB values for each zone to the mouse
32     uchar data_pkt[2][MSG_SIZE] = {
33         { 0x07, 0x22, N_MOUSE_ZONES, 0x01, 0 }, // RGB colors
34         { 0x07, 0x05, 0x02, 0 }                // Lighting on/off
35     };
36     uchar* rgb_data = &data_pkt[0][4];
37     for(int i = 0; i < N_MOUSE_ZONES; i++){
38         *rgb_data++ = i + 1;
39         *rgb_data++ = newlight->r[LED_MOUSE + i];
40         *rgb_data++ = newlight->g[LED_MOUSE + i];
41         *rgb_data++ = newlight->b[LED_MOUSE + i];
42     }
43     // Send RGB data
44     if(!usbsend(kb, data_pkt[0], 1))
45         return -1;
46     int was_black = isblack(kb, lastlight), is_black = isblack(kb, newlight);
47     if(is_black){
48         // If the lighting is black, send the deactivation packet (M65 only)
49         if(!usbsend(kb, data_pkt[1], 1))
50             return -1;
51     } else if(was_black || force){
52         // If the lighting WAS black, or if we're on forced update, send the activation packet
53         data_pkt[1][4] = 1;
54         if(!usbsend(kb, data_pkt[1], 1))
55             return -1;
56     }
57
58     memcpy(lastlight, newlight, sizeof(lighting));
59     return 0;
60 }
```

Here is the call graph for this function:



5.26 led_keyboard.c File Reference

```

#include <stdint.h>
#include "led.h"
```


Generated by Doxygen

Generated by Doxygen

Generated by Doxygen

Generated by Doxygen

Generated by Doxygen

Generated by Doxygen

Generated by Doxygen

Generated by Doxygen

Generated by Doxygen

[illegible]

5.26.1 Macro Definition Documentation

5.26.1.1 BR1

```
#define BR1(  
    x ) (((x) & 0xaa) >> 1) | (((x) & 0x55) << 1))
```

Definition at line 9 of file led keyboard.c.

5.26.1.2 BR2

```
#define BR2(  
    x ) ((BR1(x) & 0xcc) >> 2) | ((BR1(x) & 0x33) << 2)
```

Definition at line 10 of file led_keyboard.c.

5.26.1.3 BR4

```
#define BR4(  
    x ) ((BR2(x) & 0xf0) >> 4) | ((BR2(x) & 0x0f) << 4)
```

Definition at line 11 of file led_keyboard.c.

5.26.1.4 O0

```
#define O0(  
    i ) BR4(i),
```

Definition at line 12 of file led_keyboard.c.

5.26.1.5 O1

```
#define O1(  
    i ) O0(i) O0((i) + 1)
```

Definition at line 13 of file led_keyboard.c.

5.26.1.6 O2

```
#define O2(  
    i ) O1(i) O1((i) + 2)
```

Definition at line 14 of file led_keyboard.c.

5.26.1.7 O3

```
#define O3(  
    i ) O2(i) O2((i) + 4)
```

Definition at line 15 of file led_keyboard.c.

5.26.1.8 O4

```
#define O4(  
    i ) O3(i) O3((i) + 8)
```

Definition at line 16 of file led_keyboard.c.

5.26.1.9 O5

```
#define O5(  
    i ) O4(i) O4((i) + 16)
```

Definition at line 17 of file led_keyboard.c.

5.26.1.10 O6

```
#define O6(  
    i ) O5(i) O5((i) + 32)
```

Definition at line 18 of file led_keyboard.c.

5.26.1.11 O7

```
#define O7(  
    i ) O6(i) O6((i) + 64)
```

Definition at line 19 of file led_keyboard.c.

5.26.1.12 O8

```
#define O8(  
    i ) O7(i) O7((i) + 127)
```

Definition at line 20 of file led_keyboard.c.

5.26.2 Function Documentation

5.26.2.1 loadrgb_kb()

```
int loadrgb_kb (
    usbdevice * kb,
    lighting * light,
    int mode )
```

Since Firmware Version 2.05 the answers for getting the stored color-maps from the hardware has changed a bit. So comparing for the correct answer cannot validate against the cmd, and has to be done against a third map.

Definition at line 181 of file led_keyboard.c.

References lighting::b, ckb_err, usbdevice::fwversion, lighting::g, MSG_SIZE, N_KEYS_HW, lighting::r, usbrecv, and usbsend.

Referenced by hwloadmode().

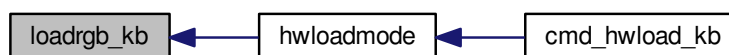
```
181                                     {
182     if(kb->fwversion >= 0x0120){
183         uchar data_pkt[12][MSG_SIZE] = {
184             { 0x0e, 0x14, 0x03, 0x01, 0x01, mode + 1, 0x01 },
185             { 0xff, 0x01, 60, 0 },
186             { 0xff, 0x02, 60, 0 },
187             { 0xff, 0x03, 24, 0 },
188             { 0x0e, 0x14, 0x03, 0x01, 0x01, mode + 1, 0x02 },
189             { 0xff, 0x01, 60, 0 },
190             { 0xff, 0x02, 60, 0 },
191             { 0xff, 0x03, 24, 0 },
192             { 0x0e, 0x14, 0x03, 0x01, 0x01, mode + 1, 0x03 },
193             { 0xff, 0x01, 60, 0 },
194             { 0xff, 0x02, 60, 0 },
195             { 0xff, 0x03, 24, 0 },
196         };
197         uchar in_pkt[4][MSG_SIZE] = {
198             { 0x0e, 0x14, 0x03, 0x01 },
199             { 0xff, 0x01, 60, 0 },
200             { 0xff, 0x02, 60, 0 },
201             { 0xff, 0x03, 24, 0 },
202         };
203
204         uchar cmp_pkt[4][4] = {
205             { 0x0e, 0x14, 0x03, 0x01 },
206             { 0x0e, 0xff, 0x01, 60 },
207             { 0x0e, 0xff, 0x02, 60 },
208             { 0x0e, 0xff, 0x03, 24 },
209         };
210         // Read colors
211         uchar* colors[3] = { light->r, light->g, light->b };
212         for(int clr = 0; clr < 3; clr++){
213             for(int i = 0; i < 4; i++){
214                 if(!usbrecv(kb, data_pkt[i + clr * 4], in_pkt[i]))
215                     return -1;
216                 // Make sure the first four bytes match
217                 // see comment above
218                 if(memcmp(p, data_pkt[i + clr * 4], 4)){
219                     if (memcmp(in_pkt[i], (kb->fwversion >= 0x0205)? cmp_pkt[i] : data_pkt[i + clr * 4], 4)) {
220                         ckb_err("Bad input header\n");
221                         ckb_err("color = %d, i = %d, mode = %d\nInput (Antwort): %2.2x %2.2x %2.2x %2.2x\nOutput (Frage): %2.2x %2.2x %2.2x %2.2x\n", clr, i, mode,
222                             in_pkt[i][0], in_pkt[i][1], in_pkt[i][2], in_pkt[i][3], in_pkt[i][4], in_pkt[i][5],
223                             in_pkt[i][6], in_pkt[i][7],
224                             // data_pkt[i + clr * 4][0],    data_pkt[i + clr * 4 ][1],    data_pkt[i + clr * 4 ][2],    data_pkt[i + clr * 4 ][3]);
225                             cmp_pkt[i][0], cmp_pkt[i][1], cmp_pkt[i][2], cmp_pkt[i][3]);
226                             in_pkt[2][0] = 0x99;
227                             in_pkt[2][1] = 0x99;
228                             in_pkt[2][2] = 0x99;
229                             in_pkt[2][3] = 0x99;
230                             usbrecv(kb, in_pkt[2], in_pkt[2]); // just to find it in the wireshark log
231                             return -1;
232                         }
233                     }
234                 // Copy colors to lighting. in_pkt[0] is irrelevant.
235                 memcpy(colors[clr], in_pkt[1] + 4, 60);
236                 memcpy(colors[clr] + 60, in_pkt[2] + 4, 60);
237             }
238         }
239     }
```

```

240     memcpy(colors[clr] + 120, in_pkt[3] + 4, 24);
241 }
242 } else {
243     uchar data_pkt[5][MSG_SIZE] = {
244         { 0x0e, 0x14, 0x02, 0x01, 0x01, mode + 1, 0 },
245         { 0xff, 0x01, 60, 0 },
246         { 0xff, 0x02, 60, 0 },
247         { 0xff, 0x03, 60, 0 },
248         { 0xff, 0x04, 36, 0 },
249     };
250     uchar in_pkt[4][MSG_SIZE] = {
251         { 0xff, 0x01, 60, 0 },
252         { 0xff, 0x02, 60, 0 },
253         { 0xff, 0x03, 60, 0 },
254         { 0xff, 0x04, 36, 0 },
255     };
256     // Write initial packet
257     if(!usbsend(kb, data_pkt[0], 1))
258         return -1;
259     // Read colors
260     for(int i = 1; i < 5; i++){
261         if(!usbrecv(kb, data_pkt[i], in_pkt[i - 1]))
262             return -1;
263         if(memcmp(in_pkt[i - 1], data_pkt[i], 4)){
264             ckb_err("Bad input header\n");
265             return -1;
266         }
267     }
268     // Copy the data back to the mode
269     uint8_t mr[N_KEYS_HW / 2], mg[N_KEYS_HW / 2], mb[
N_KEYS_HW / 2];
270     memcpy(mr, in_pkt[0] + 4, 60);
271     memcpy(mr + 60, in_pkt[1] + 4, 12);
272     memcpy(mg, in_pkt[1] + 16, 48);
273     memcpy(mg + 48, in_pkt[2] + 4, 24);
274     memcpy(mb, in_pkt[2] + 28, 36);
275     memcpy(mb + 36, in_pkt[3] + 4, 36);
276     // Unpack LED data to 8bpc format
277     for(int i = 0; i < N_KEYS_HW; i++){
278         int i_2 = i / 2;
279         uint8_t r, g, b;
280
281         // 3-bit intensities stored in alternate nybbles.
282         if (i & 1) {
283             r = 7 - (mr[i_2] >> 4);
284             g = 7 - (mg[i_2] >> 4);
285             b = 7 - (mb[i_2] >> 4);
286         } else {
287             r = 7 - (mr[i_2] & 0x0F);
288             g = 7 - (mg[i_2] & 0x0F);
289             b = 7 - (mb[i_2] & 0x0F);
290         }
291         // Scale 3-bit values up to 8 bits.
292         light->r[i] = r << 5 | r << 2 | r >> 1;
293         light->g[i] = g << 5 | g << 2 | g >> 1;
294         light->b[i] = b << 5 | b << 2 | b >> 1;
295     }
296 }
297 return 0;
298 }

```

Here is the caller graph for this function:



5.26.2.2 makergb_512()

```
static void makergb_512 (
    const lighting * light,
    uchar data_pkt[5][64],
    uchar(*)(int, uchar) ditherfn ) [static]
```

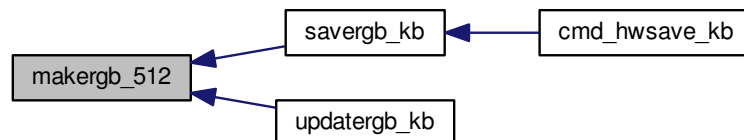
Definition at line 36 of file `led_keyboard.c`.

References `lighting::b`, `lighting::g`, `N_KEYS_HW`, and `lighting::r`.

Referenced by `savergb_kb()`, and `updatergb_kb()`.

```
37
38     uchar r[N_KEYS_HW / 2], g[N_KEYS_HW / 2], b[N_KEYS_HW / 2];
39     // Compress RGB values to a 512-color palette
40     for(int i = 0; i < N_KEYS_HW; i += 2){
41         char r1 = ditherfn(i, light->r[i]), r2 = ditherfn(i + 1, light->r[i + 1]);
42         char g1 = ditherfn(i, light->g[i]), g2 = ditherfn(i + 1, light->g[i + 1]);
43         char b1 = ditherfn(i, light->b[i]), b2 = ditherfn(i + 1, light->b[i + 1]);
44         r[i / 2] = (7 - r2) << 4 | (7 - r1);
45         g[i / 2] = (7 - g2) << 4 | (7 - g1);
46         b[i / 2] = (7 - b2) << 4 | (7 - b1);
47     }
48     memcpy(data_pkt[0] + 4, r, 60);
49     memcpy(data_pkt[1] + 4, r + 60, 12);
50     memcpy(data_pkt[1] + 16, g, 48);
51     memcpy(data_pkt[2] + 4, g + 48, 24);
52     memcpy(data_pkt[2] + 28, b, 36);
53     memcpy(data_pkt[3] + 4, b + 36, 36);
54 }
```

Here is the caller graph for this function:



5.26.2.3 makergb_full()

```
static void makergb_full (
    const lighting * light,
    uchar data_pkt[12][64] ) [static]
```

Definition at line 56 of file `led_keyboard.c`.

References `lighting::b`, `lighting::g`, and `lighting::r`.

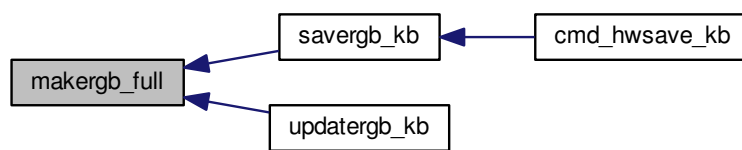
Referenced by `savergb_kb()`, and `updatergb_kb()`.

```

56
57     const uchar* r = light->r, *g = light->g, *b = light->b;
58     // Red
59     memcpy(data_pkt[0] + 4, r, 60);
60     memcpy(data_pkt[1] + 4, r + 60, 60);
61     memcpy(data_pkt[2] + 4, r + 120, 24);
62     // Green (final R packet is blank)
63     memcpy(data_pkt[4] + 4, g, 60);
64     memcpy(data_pkt[5] + 4, g + 60, 60);
65     memcpy(data_pkt[6] + 4, g + 120, 24);
66     // Blue (final G packet is blank)
67     memcpy(data_pkt[8] + 4, b, 60);
68     memcpy(data_pkt[9] + 4, b + 60, 60);
69     memcpy(data_pkt[10] + 4, b + 120, 24);
70 }

```

Here is the caller graph for this function:



5.26.2.4 ordered8to3()

```

static uchar ordered8to3 (
    int index,
    uchar value ) [static]

```

Definition at line 24 of file led_keyboard.c.

References `bit_reverse_table`.

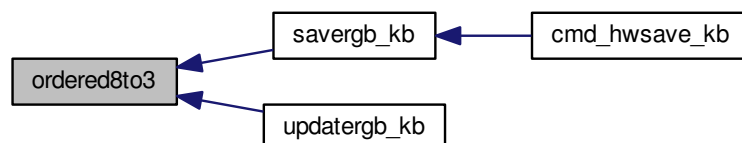
Referenced by `savergb_kb()`, and `updatergb_kb()`.

```

24
25     int m = value * 7;
26     int b = m / 255;
27     if ((m % 255) > bit_reverse_table[index & 0xff])
28         b++;
29     return b;
30 }

```

Here is the caller graph for this function:



5.26.2.5 quantize8to3()

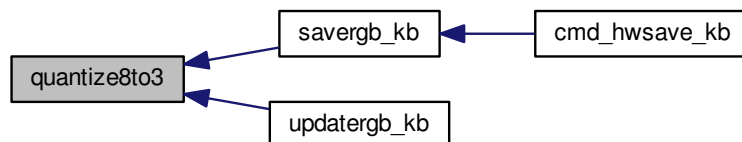
```
static uchar quantize8to3 (
    int index,
    uchar value ) [static]
```

Definition at line 32 of file led_keyboard.c.

Referenced by savergb_kb(), and updatergb_kb().

```
32                                     {
33     return value >> 5;
34 }
```

Here is the caller graph for this function:



5.26.2.6 rgbcmp()

```
static int rgbcmp (
    const lighting * lhs,
    const lighting * rhs ) [static]
```

Definition at line 72 of file led_keyboard.c.

References `lighting::b`, `lighting::g`, `N_KEYS_HW`, and `lighting::r`.

Referenced by `updatergb_kb()`.

```
72                                     {
73     // Compare two light structures, ignore mouse zones
74     return memcmp(lhs->r, rhs->r, N_KEYS_HW) || memcmp(lhs->g, rhs->
75     g, N_KEYS_HW) || memcmp(lhs->b, rhs->b, N_KEYS_HW);
76 }
```

Here is the caller graph for this function:



5.26.2.7 savergb_kb()

```
int savergb_kb (
    usbdevice * kb,
    lighting * light,
    int mode )
```

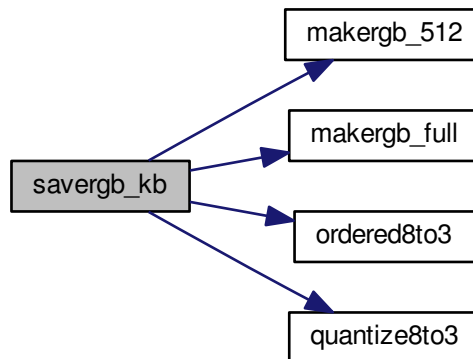
Definition at line 139 of file led_keyboard.c.

References `usbdevice::dither`, `usbdevice::fwversion`, `IS_STRAFE`, `makergb_512()`, `makergb_full()`, `MSG_SIZE`, `ordered8to3()`, `quantize8to3()`, and `usb send`.

Referenced by `cmd_hwsave_kb()`.

```
139                                     {
140     if(kb->fwversion >= 0x0120){
141         uchar data_pkt[12][MSG_SIZE] = {
142             // Red
143             { 0x7f, 0x01, 60, 0 },
144             { 0x7f, 0x02, 60, 0 },
145             { 0x7f, 0x03, 24, 0 },
146             { 0x07, 0x14, 0x03, 0x01, 0x01, mode + 1, 0x01 },
147             // Green
148             { 0x7f, 0x01, 60, 0 },
149             { 0x7f, 0x02, 60, 0 },
150             { 0x7f, 0x03, 24, 0 },
151             { 0x07, 0x14, 0x03, 0x01, 0x01, mode + 1, 0x02 },
152             // Blue
153             { 0x7f, 0x01, 60, 0 },
154             { 0x7f, 0x02, 60, 0 },
155             { 0x7f, 0x03, 24, 0 },
156             { 0x07, 0x14, 0x03, 0x01, 0x01, mode + 1, 0x03 }
157         };
158         makergb_full(light, data_pkt);
159         if(!usb send(kb, data_pkt[0], 12))
160             return -1;
161         if (IS_STRAFE(kb)){ // end save
162             uchar save_end_pkt[MSG_SIZE] = { 0x07, 0x14, 0x04, 0x01, 0x01 };
163             if(!usb send(kb, save_end_pkt, 1))
164                 return -1;
165         }
166     } else {
167         uchar data_pkt[5][MSG_SIZE] = {
168             { 0x7f, 0x01, 60, 0 },
169             { 0x7f, 0x02, 60, 0 },
170             { 0x7f, 0x03, 60, 0 },
171             { 0x7f, 0x04, 36, 0 },
172             { 0x07, 0x14, 0x02, 0x00, 0x01, mode + 1 }
173         };
174         makergb_512(light, data_pkt, kb->dither ? ordered8to3 :
quantize8to3);
175         if(!usb send(kb, data_pkt[0], 5))
176             return -1;
177     }
178     return 0;
179 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



5.26.2.8 updatergb_kb()

```

int updatergb_kb (
    usbdevice * kb,
    int force )

```

Definition at line 77 of file `led_keyboard.c`.

References `usbdevice::active`, `usbprofile::currentmode`, `usbdevice::dither`, `lighting::forceupdate`, `IS_FULLRANGE`, `usbprofile::lastlight`, `usbmode::light`, `makergb_512()`, `makergb_full()`, `MSG_SIZE`, `ordered8to3()`, `usbdevice::profile`, `quantize8to3()`, `rgbcmp()`, `lighting::sidelight`, and `usbsend`.

```

77                                     {
78     if (!kb->active)
79         return 0;
80     lighting* lastlight = &kb->profile->lastlight;
81     lighting* newlight = &kb->profile->currentmode->
light;
82     // Don't do anything if the lighting hasn't changed
83     if (!force && !lastlight->forceupdate && !newlight->forceupdate
84         && !rgbcmp(lastlight, newlight) && lastlight->sidelight == newlight->

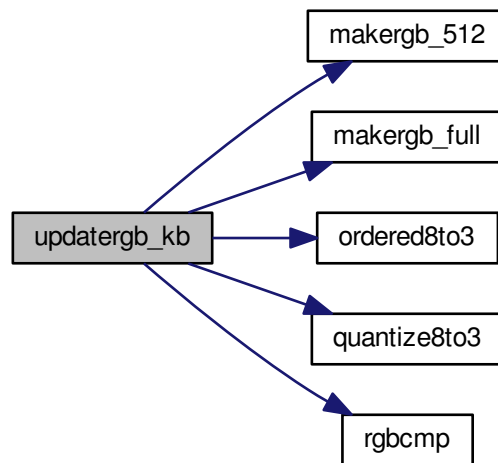
```

```

sidelight) // strafe sidelights
85     return 0;
86     lastlight->forceupdate = newlight->forceupdate = 0;
87
88     if(IS_FULLRANGE(kb)){
89         // Update strafe sidelights if necessary
90         if(lastlight->sidelight != newlight->sidelight) {
91             uchar data_pkt[2][MSG_SIZE] = {
92                 { 0x07, 0x05, 0x08, 0x00, 0x00 },
93                 { 0x07, 0x05, 0x02, 0, 0x03 }
94             };
95             if (newlight->sidelight)
96                 data_pkt[0][4]=1; // turn on
97             if(!usbseend(kb, data_pkt[0], 2))
98                 return -1;
99         }
100         // 16.8M color lighting works fine on strafe and is the only way it actually works
101         uchar data_pkt[12][MSG_SIZE] = {
102             // Red
103             { 0x7f, 0x01, 0x3c, 0 },
104             { 0x7f, 0x02, 0x3c, 0 },
105             { 0x7f, 0x03, 0x18, 0 },
106             { 0x07, 0x28, 0x01, 0x03, 0x01, 0},
107             // Green
108             { 0x7f, 0x01, 0x3c, 0 },
109             { 0x7f, 0x02, 0x3c, 0 },
110             { 0x7f, 0x03, 0x18, 0 },
111             { 0x07, 0x28, 0x02, 0x03, 0x01, 0},
112             // Blue
113             { 0x7f, 0x01, 0x3c, 0 },
114             { 0x7f, 0x02, 0x3c, 0 },
115             { 0x7f, 0x03, 0x18, 0 },
116             { 0x07, 0x28, 0x03, 0x03, 0x02, 0}
117         };
118         makergb_full(newlight, data_pkt);
119         if(!usbseend(kb, data_pkt[0], 12))
120             return -1;
121     } else {
122         // On older keyboards it looks flickery and causes lighting glitches, so we don't use it.
123         uchar data_pkt[5][MSG_SIZE] = {
124             { 0x7f, 0x01, 60, 0 },
125             { 0x7f, 0x02, 60, 0 },
126             { 0x7f, 0x03, 60, 0 },
127             { 0x7f, 0x04, 36, 0 },
128             { 0x07, 0x27, 0x00, 0x00, 0xD8 }
129         };
130         makergb_512(newlight, data_pkt, kb->dither ?
ordered8to3 : quantize8to3);
131         if(!usbseend(kb, data_pkt[0], 5))
132             return -1;
133     }
134
135     memcpy(lastlight, newlight, sizeof(lighting));
136     return 0;
137 }

```

Here is the call graph for this function:



5.26.3 Variable Documentation

5.26.3.1 bit_reverse_table

```

uchar bit_reverse_table[256] = { ((((((((( ( 0 ) & 0xaa) >> 1) | ((( 0 ) & 0x55) << 1)) &
0xcc) >> 2) | (((((( 0 ) & 0xaa) >> 1) | ((( 0 ) & 0x55) << 1)) & 0x33) << 2)) & 0xf0) >>
4) | ((((((((( ( 0 ) & 0xaa) >> 1) | ((( 0 ) & 0x55) << 1)) & 0xcc) >> 2) | (((((( 0 ) & 0xaa)
>> 1) | ((( 0 ) & 0x55) << 1)) & 0x33) << 2)) & 0x0f) << 4)), (((((((((( ( 0 ) + 1 ) & 0xaa)
>> 1) | ((( ( 0 ) + 1 ) & 0x55) << 1)) & 0xcc) >> 2) | (((((( ( 0 ) + 1 ) & 0xaa) >> 1) | ((( ( 0 ) + 1 ) & 0x55) << 1)) & 0x33) << 2)) & 0xf0) >> 4) | (((((((((( ( 0 ) + 1 ) & 0xaa) >>
1) | ((( ( 0 ) + 1 ) & 0x55) << 1)) & 0xcc) >> 2) | (((((( ( 0 ) + 1 ) & 0xaa) >> 1) | ((( ( 0 ) + 1 ) & 0x55) << 1)) & 0x33) << 2)) & 0x0f) << 4)), (((((((((( ( 0 ) + 2 ) & 0xaa) >>
1) | ((( ( 0 ) + 2 ) & 0x55) << 1)) & 0xcc) >> 2) | (((((( ( 0 ) + 2 ) & 0xaa) >> 1) | ((( ( 0 ) + 2 ) & 0x55) << 1)) & 0x33) << 2)) & 0xf0) >> 4) | (((((((((( ( 0 ) + 2 ) & 0xaa) >> 1) |
((( ( 0 ) + 2 ) & 0x55) << 1)) & 0xcc) >> 2) | (((((( ( 0 ) + 2 ) & 0xaa) >> 1) | ((( ( 0 ) + 2 ) & 0x55) << 1)) & 0x33) << 2)) & 0x0f) << 4)), (((((((((( ( ( 0 ) + 2 ) + 1 ) & 0xaa) >>
1) | ((( ( ( 0 ) + 2 ) + 1 ) & 0x55) << 1)) & 0xcc) >> 2) | (((((( ( ( 0 ) + 2 ) + 1 ) & 0xaa)
>> 1) | ((( ( ( 0 ) + 2 ) + 1 ) & 0x55) << 1)) & 0x33) << 2)) & 0xf0) >> 4) | (((((((((( ( ( 0 ) + 2 ) + 1 ) & 0xaa) >> 1) | ((( ( ( 0 ) + 2 ) + 1 ) & 0x55) << 1)) & 0xcc) >> 2) | (((((( ( ( 0 ) + 2 ) + 1 ) & 0xaa) >> 1) |
((( ( ( 0 ) + 2 ) + 1 ) & 0x55) << 1)) & 0x33) << 2)) & 0x0f) << 4)), (((((((((( ( ( 0 ) + 4 ) & 0xaa) >> 1) | ((( ( 0 ) + 4 ) & 0x55) << 1)) & 0xcc)
>> 2) | (((((( ( 0 ) + 4 ) & 0xaa) >> 1) | ((( ( 0 ) + 4 ) & 0x55) << 1)) & 0x33) << 2)) &
0xf0) >> 4) | (((((((((( ( 0 ) + 4 ) & 0xaa) >> 1) | ((( ( 0 ) + 4 ) & 0x55) << 1)) & 0xcc)
>> 2) | (((((( ( 0 ) + 4 ) & 0xaa) >> 1) | ((( ( 0 ) + 4 ) & 0x55) << 1)) & 0x33) << 2)) &
0x0f) << 4)), (((((((((( ( ( 0 ) + 4 ) + 1 ) & 0xaa) >> 1) | ((( ( ( 0 ) + 4 ) + 1 ) & 0x55)
<< 1)) & 0xcc) >> 2) | (((((( ( ( 0 ) + 4 ) + 1 ) & 0xaa) >> 1) | ((( ( ( 0 ) + 4 ) + 1 ) &
0x55) << 1)) & 0x33) << 2)) & 0xf0) >> 4) | (((((((((( ( ( 0 ) + 4 ) + 1 ) & 0xaa) >> 1) |

```

Generated by Doxygen

Generated by Doxygen

Generated by Doxygen

Generated by Doxygen

Generated by Doxygen

Generated by Doxygen

Generated by Doxygen

Generated by Doxygen

Generated by Doxygen

Generated by Doxygen

Generated by Doxygen

Generated by Doxygen

Generated by Doxygen

Generated by Doxygen

Generated by Doxygen

Generated by Doxygen

Generated by Doxygen

Generated by Doxygen

Generated by Doxygen

```
<< 2)) & 0x0f) << 4)), } [static]
```

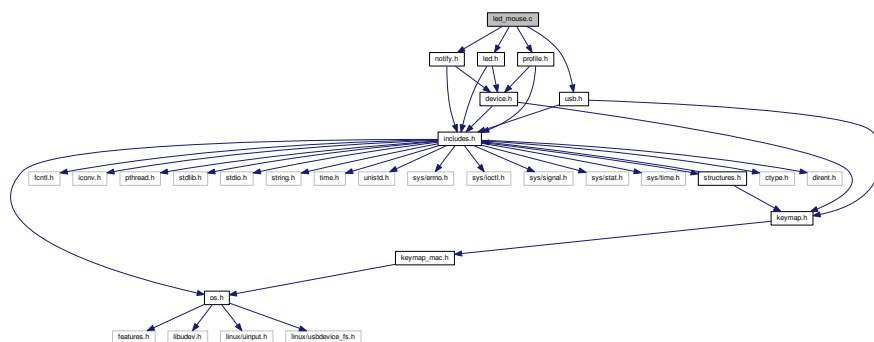
Definition at line 22 of file led_keyboard.c.

Referenced by ordered8to3().

5.27 led_mouse.c File Reference

```
#include "led.h"
#include "notify.h"
#include "profile.h"
#include "usb.h"
```

Include dependency graph for led_mouse.c:



Functions

- static int [rgbcmp](#) (const [lighting](#) *lhs, const [lighting](#) *rhs)
- static int [isblack](#) (const [usbdevice](#) *kb, const [lighting](#) *light)
- int [updatergb_mouse](#) ([usbdevice](#) *kb, int force)
- int [savergb_mouse](#) ([usbdevice](#) *kb, [lighting](#) *light, int mode)
- int [loadrgb_mouse](#) ([usbdevice](#) *kb, [lighting](#) *light, int mode)

5.27.1 Function Documentation

5.27.1.1 isblack()

```
static int isblack (
    const usbdevice * kb,
    const lighting * light ) [static]
```

Definition at line 13 of file led_mouse.c.

References [lighting::b](#), [lighting::g](#), [IS_M65](#), [LED_MOUSE](#), [N_MOUSE_ZONES](#), and [lighting::r](#).

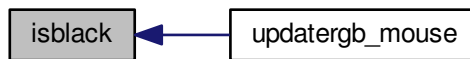
Referenced by [updatergb_mouse\(\)](#).

```

13                                     {
14     if(!IS_M65(kb))
15         return 0;
16     uchar black[N_MOUSE_ZONES] = { 0 };
17     return !memcmp(light->r + LED_MOUSE, black, sizeof(black)) && !memcmp(light->
18     g + LED_MOUSE, black, sizeof(black)) && !memcmp(light->b + LED_MOUSE, black, sizeof(
19     black));
20 }

```

Here is the caller graph for this function:



5.27.1.2 loadrgb_mouse()

```

int loadrgb_mouse (
    usbdevice * kb,
    lighting * light,
    int mode )

```

Definition at line 81 of file led_mouse.c.

References `lighting::b`, `ckb_err`, `lighting::g`, `IS_SABRE`, `IS_SCIMITAR`, `LED_DPI`, `LED_MOUSE`, `MSG_SIZE`, `lighting::r`, and `usbrecv`.

Referenced by `cmd_hwload_mouse()`.

```

81                                     {
82     uchar data_pkt[MSG_SIZE] = { 0x0e, 0x13, 0x10, 1, 0 };
83     uchar in_pkt[MSG_SIZE] = { 0 };
84     // Load each RGB zone
85     int zonecount = IS_SCIMITAR(kb) ? 4 : IS_SABRE(kb) ? 3 : 2;
86     for(int i = 0; i < zonecount; i++){
87         if(!usbrecv(kb, data_pkt, in_pkt))
88             return -1;
89         if(memcmp(in_pkt, data_pkt, 4)){
90             ckb_err("Bad input header\n");
91             return -2;
92         }
93         // Copy data
94         int led = LED_MOUSE + i;
95         if(led >= LED_DPI)
96             led++; // Skip DPI light
97         light->r[led] = in_pkt[4];
98         light->g[led] = in_pkt[5];
99         light->b[led] = in_pkt[6];
100        // Set packet for next zone
101        data_pkt[2]++;
102    }
103    return 0;
104 }

```

Here is the caller graph for this function:



5.27.1.3 rgbcmp()

```
static int rgbcmp (
    const lighting * lhs,
    const lighting * rhs ) [static]
```

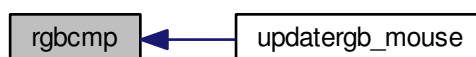
Definition at line 7 of file `led_mouse.c`.

References `lighting::b`, `lighting::g`, `LED_MOUSE`, `N_MOUSE_ZONES`, and `lighting::r`.

Referenced by `updatergb_mouse()`.

```
7
8     return memcmp(lhs->r + LED_MOUSE, rhs->r + LED_MOUSE,
9                 N_MOUSE_ZONES) || memcmp(lhs->g + LED_MOUSE, rhs->g +
    LED_MOUSE, N_MOUSE_ZONES) || memcmp(lhs->b + LED_MOUSE, rhs->
    b + LED_MOUSE, N_MOUSE_ZONES);
```

Here is the caller graph for this function:



5.27.1.4 savergb_mouse()

```
int savergb_mouse (
    usbdevice * kb,
    lighting * light,
    int mode )
```

Definition at line 62 of file led_mouse.c.

References `lighting::b`, `lighting::g`, `IS_SABRE`, `IS_SCIMITAR`, `LED_DPI`, `LED_MOUSE`, `MSG_SIZE`, `lighting::r`, and `usbsend`.

Referenced by `cmd_hwsave_mouse()`.

```
62
63     uchar data_pkt[MSG_SIZE] = { 0x07, 0x13, 0x10, 1, 0 };
64     // Save each RGB zone, minus the DPI light which is sent in the DPI packets
65     int zonecount = IS_SCIMITAR(kb) ? 4 : IS_SABRE(kb) ? 3 : 2;
66     for(int i = 0; i < zonecount; i++){
67         int led = LED_MOUSE + i;
68         if(led >= LED_DPI)
69             led++; // Skip DPI light
70         data_pkt[4] = light->r[led];
71         data_pkt[5] = light->g[led];
72         data_pkt[6] = light->b[led];
73         if(!usbsend(kb, data_pkt, 1))
74             return -1;
75         // Set packet for next zone
76         data_pkt[2]++;
77     }
78     return 0;
79 }
```

Here is the caller graph for this function:



5.27.1.5 updatergb_mouse()

```
int updatergb_mouse (
    usbdevice * kb,
    int force )
```

Definition at line 20 of file led_mouse.c.

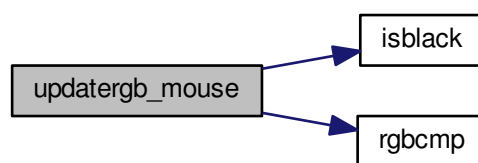
References `usbdevice::active`, `lighting::b`, `usbprofile::currentmode`, `lighting::forceupdate`, `lighting::g`, `isblack()`, `usbprofile::lastlight`, `LED_MOUSE`, `usbmode::light`, `MSG_SIZE`, `N_MOUSE_ZONES`, `usbdevice::profile`, `lighting::r`, `rgbcmp()`, and `usbsend`.

```

20                                     {
21     if(!kb->active)
22         return 0;
23     lighting* lastlight = &kb->profile->lastlight;
24     lighting* newlight = &kb->profile->currentmode->
light;
25     // Don't do anything if the lighting hasn't changed
26     if(!force && !lastlight->forceupdate && !newlight->forceupdate
27         && !rgbcmp(lastlight, newlight))
28         return 0;
29     lastlight->forceupdate = newlight->forceupdate = 0;
30
31     // Send the RGB values for each zone to the mouse
32     uchar data_pkt[2][MSG_SIZE] = {
33         { 0x07, 0x22, N_MOUSE_ZONES, 0x01, 0 }, // RGB colors
34         { 0x07, 0x05, 0x02, 0 }                // Lighting on/off
35     };
36     uchar* rgb_data = &data_pkt[0][4];
37     for(int i = 0; i < N_MOUSE_ZONES; i++){
38         *rgb_data++ = i + 1;
39         *rgb_data++ = newlight->r[LED_MOUSE + i];
40         *rgb_data++ = newlight->g[LED_MOUSE + i];
41         *rgb_data++ = newlight->b[LED_MOUSE + i];
42     }
43     // Send RGB data
44     if(!usb_send(kb, data_pkt[0], 1))
45         return -1;
46     int was_black = isblack(kb, lastlight), is_black = isblack(kb, newlight);
47     if(is_black){
48         // If the lighting is black, send the deactivation packet (M65 only)
49         if(!usb_send(kb, data_pkt[1], 1))
50             return -1;
51     } else if(was_black || force){
52         // If the lighting WAS black, or if we're on forced update, send the activation packet
53         data_pkt[1][4] = 1;
54         if(!usb_send(kb, data_pkt[1], 1))
55             return -1;
56     }
57
58     memcpy(lastlight, newlight, sizeof(lighting));
59     return 0;
60 }

```

Here is the call graph for this function:



5.28 main.c File Reference

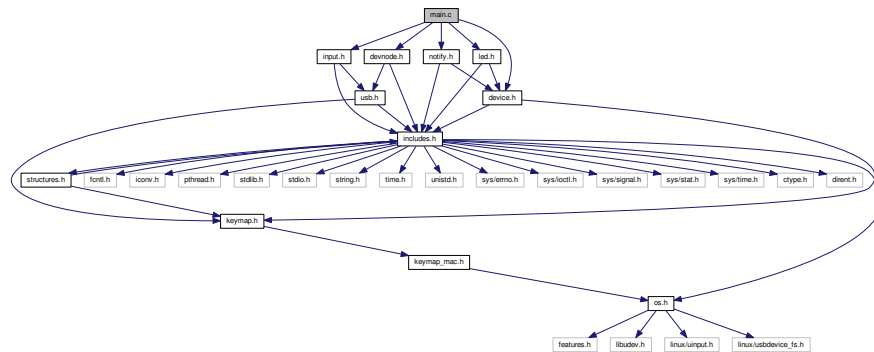
```

#include "device.h"
#include "devnode.h"
#include "input.h"
#include "led.h"

```

```
#include "notify.h"
```

Include dependency graph for main.c:



Functions

- static void [quitWithLock](#) (char mut)
quitWithLock
- int [restart](#) ()
- void [timespec_add](#) (struct timespec *timespec, long nanoseconds)
- static void [quit](#) ()
quit Stop working the daemon. function is called if the daemon received a sigterm In this case, locking the device-mutex is ok.
- void [sighandler2](#) (int type)
- void [sighandler](#) (int type)
- void [localecase](#) (char *dst, size_t length, const char *src)
- int [main](#) (int argc, char **argv)

Variables

- static int [main_ac](#)
- static char ** [main_av](#)
- volatile int [reset_stop](#)
brief .
- int [features_mask](#)
brief .
- int [hwload_mode](#)

5.28.1 Function Documentation

5.28.1.1 localecase()

```
void localecase (
    char * dst,
    size_t length,
    const char * src )
```

Definition at line 71 of file main.c.

```
71                                     {
72     char* ldst = dst + length;
73     char s;
74     while((s = *src++){
75         if(s == '_')
76             s = '-';
77         else
78             s = tolower(s);
79         *dst++ = s;
80         if(dst == ldst){
81             dst--;
82             break;
83         }
84     }
85     *dst = 0;
86 }
```

5.28.1.2 main()

```
int main (
    int argc,
    char ** argv )
```

Definition at line 88 of file main.c.

References `ckb_fatal_nofile`, `ckb_info`, `ckb_info_nofile`, `ckb_warn_nofile`, `devpath`, `FEAT_BIND`, `FEAT_MOUSE`, `ACCEL`, `FEAT_NOTIFY`, `features_mask`, `gid`, `hwload_mode`, `keyboard`, `main_ac`, `main_av`, `mkdevpath()`, `quit()`, `restart()`, `sighandler()`, and `usbmain()`.

Referenced by `restart()`.

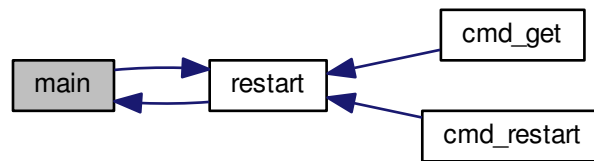
```
88                                     {
89     // Set output pipes to buffer on newlines, if they weren't set that way already
90     setlinebuf(stdout);
91     setlinebuf(stderr);
92     main_ac = argc;
93     main_av = argv;
94
95     printf("    ckb: Corsair RGB driver %s\n", CKB_VERSION_STR);
96     // If --help occurs anywhere in the command-line, don't launch the program but instead print usage
97     for(int i = 1; i < argc; i++){
98         if(!strcmp(argv[i], "--help")){
99             printf(
100 #ifdef OS_MAC
101                 "Usage: ckb-daemon [--gid=<gid>] [--hwload=<always|try|never>] [--nonotify]
102                 [--nobind] [--nomouseaccel] [--nonroot]\n"
103 #else
104                 "Usage: ckb-daemon [--gid=<gid>] [--hwload=<always|try|never>] [--nonotify]
105                 [--nobind] [--nonroot]\n"
106 #endif
107                 "\n"
108                 "See https://github.com/ccMSC/ckb/blob/master/DAEMON.md for full instructions.\n"
109                 "\n"
110                 "Command-line parameters:\n"
111                 "    --gid=<gid>\n"
112                 "    Restrict access to %s* nodes to users in group <gid>.\n"
113                 "    (Ordinarily they are accessible to anyone)\n"
```

```

112         "        --hwload=<always|try|never>\n"
113         "        --hwload=always will force loading of stored hardware profiles on
compatible devices. May result in long start up times.\n"
114         "        --hwload=try will try to load the profiles, but give up if not immediately
successful (default).\n"
115         "        --hwload=never will ignore hardware profiles completely.\n"
116         "        --nonotify\n"
117         "        Disables key monitoring/notifications.\n"
118         "        Note that this makes reactive lighting impossible.\n"
119         "        --nobind\n"
120         "        Disables all key rebinding, macros, and notifications. Implies --nonotify.
\n"
121 #ifdef OS_MAC
122         "        --nomouseaccel\n"
123         "        Disables mouse acceleration, even if the system preferences enable it.\n"
124 #endif
125         "        --nonroot\n"
126         "        Allows running ckb-daemon as a non root user.\n"
127         "        This will almost certainly not work. Use only if you know what you're
doing.\n"
128         "\n", devpath);
129     exit(0);
130 }
131 }
132
133 // Check PID, quit if already running
134 char pidpath[strlen(devpath) + 6];
135 snprintf(pidpath, sizeof(pidpath), "%s0/pid", devpath);
136 FILE* pidfile = fopen(pidpath, "r");
137 if(pidfile){
138     pid_t pid;
139     fscanf(pidfile, "%d", &pid);
140     fclose(pidfile);
141     if(pid > 0){
142         // kill -s 0 checks if the PID is active but doesn't send a signal
143         if(!kill(pid, 0)){
144             ckb_fatal_nofile("ckb-daemon is already running (PID %d). Try 'killall
ckb-daemon'.\n", pid);
145             ckb_fatal_nofile("(If you're certain the process is dead, delete %s and try
again)\n", pidpath);
146             return 0;
147         }
148     }
149 }
150
151 // Read parameters
152 int forceroor = 1;
153 for(int i = 1; i < argc; i++){
154     char* argument = argv[i];
155     unsigned newgid;
156     char hwload[7];
157     if(sscanf(argument, "--gid=%u", &newgid) == 1){
158         // Set dev node GID
159         gid = newgid;
160         ckb_info_nofile("Setting /dev node gid: %u\n", newgid);
161     } else if(!strcmp(argument, "--nobind")){
162         // Disable key notifications and rebinding
163         features_mask &= ~FEAT_BIND & ~FEAT_NOTIFY;
164         ckb_info_nofile("Key binding and key notifications are disabled\n");
165     } else if(!strcmp(argument, "--nonotify")){
166         // Disable key notifications
167         features_mask &= ~FEAT_NOTIFY;
168         ckb_info_nofile("Key notifications are disabled\n");
169     } else if(sscanf(argument, "--hwload=%6s", hwload) == 1){
170         if(!strcmp(hwload, "always") || !strcmp(hwload, "yes") || !strcmp(hwload, "y") || !strcmp(
hwload, "a")){
171             hwload_mode = 2;
172             ckb_info_nofile("Setting hardware load: always\n");
173         } else if(!strcmp(hwload, "tryonce") || !strcmp(hwload, "try") || !strcmp(hwload, "once") || !
strcmp(hwload, "t") || !strcmp(hwload, "o")){
174             hwload_mode = 1;
175             ckb_info_nofile("Setting hardware load: tryonce\n");
176         } else if(!strcmp(hwload, "never") || !strcmp(hwload, "none") || !strcmp(hwload, "no") || !
strcmp(hwload, "n")){
177             hwload_mode = 0;
178             ckb_info_nofile("Setting hardware load: never\n");
179         }
180     } else if(!strcmp(argument, "--nonroot")){
181         // Allow running as a non-root user
182         forceroor = 0;
183     }
184 #ifdef OS_MAC
185     else if(!strcmp(argument, "--nomouseaccel")){
186         // On OSX, provide an option to disable mouse acceleration
187         features_mask &= ~FEAT_MOUSEACCEL;
188         ckb_info_nofile("Mouse acceleration disabled\n");
189     }
190 #endif

```


Here is the caller graph for this function:



5.28.1.3 quit()

```
static void quit ( ) [static]
```

Definition at line 30 of file main.c.

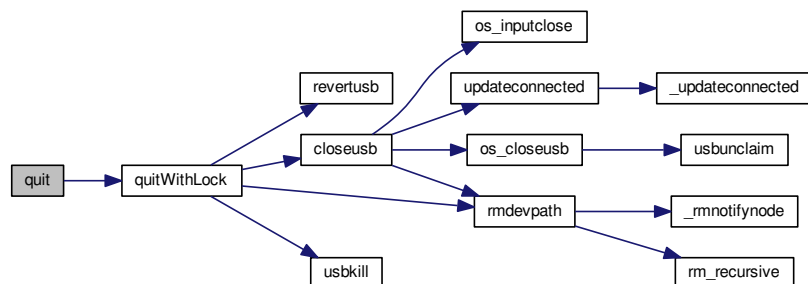
References `quitWithLock()`.

Referenced by `main()`, and `sighandler()`.

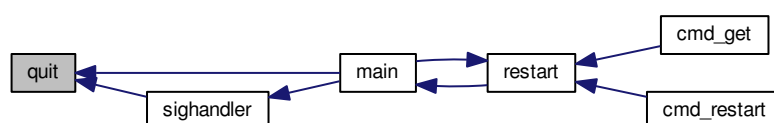
```

30 {
31     quitWithLock(1);
32 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



5.28.1.4 quitWithLock()

```
void quitWithLock (
    char mut ) [static]
```

Parameters

<i>mut</i>	try to close files maybe without locking the mutex if mut == true then lock
------------	---

Definition at line 40 of file main.c.

References `ckb_info`, `closeusb()`, `DEV_MAX`, `devmutex`, `IS_CONNECTED`, `keyboard`, `reset_stop`, `revertusb()`, `rmdevpath()`, and `usbkill()`.

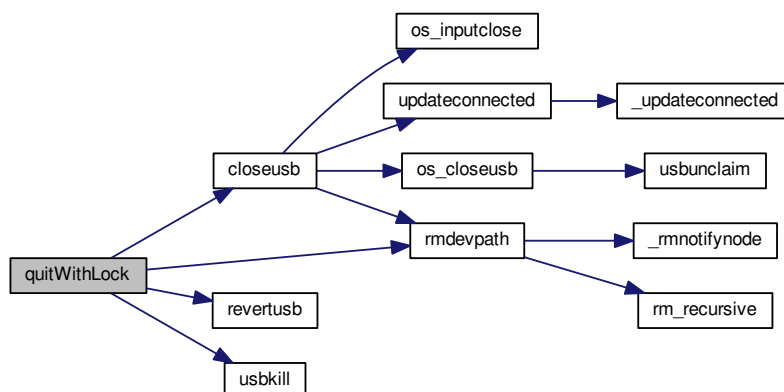
Referenced by `quit()`, and `restart()`.

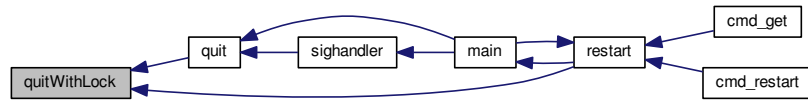
```

40     {
41         // Abort any USB resets in progress
42         reset_stop = 1;
43         for(int i = 1; i < DEV_MAX; i++){
44             // Before closing, set all keyboards back to HID input mode so that the stock driver can still talk
45             to them
46             if (mut) pthread_mutex_lock(devmutex + i);
47             if (IS_CONNECTED(keyboard + i)) {
48                 revertusb(keyboard + i);
49                 closeusb(keyboard + i);
50             }
51             pthread_mutex_unlock(devmutex + i);
52         }
53         ckb_info("Closing root controller\n");
54         rmdevpath(keyboard);
55         usbkill();
56     }

```

Here is the call graph for this function:



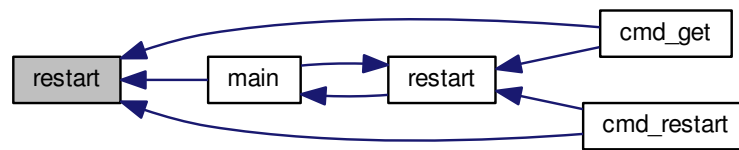


```
int restart ( )
```

References `ckb_err`, `main()`, `main_ac`, `main_av`, and `quitWithLock()`.

```
228         {
229             ckb_err("restart called, running quit without mutex-lock.\n");
230             quitWithLock(0);
231             return main(main_ac, main_av);
232     }
```

Here is the caller graph for this function:



5.28.1.6 sighandler()

```
void sighandler (
    int type )
```

Definition at line 62 of file main.c.

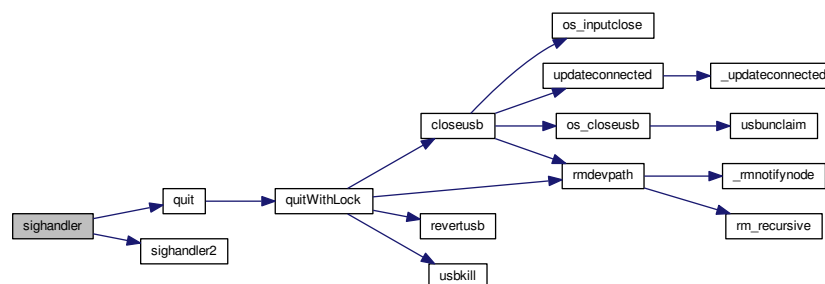
References `quit()`, and `sighandler2()`.

Referenced by `main()`.

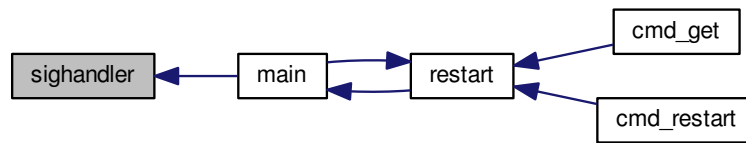
```

62     {
63         signal(SIGTERM, sighandler2);
64         signal(SIGINT, sighandler2);
65         signal(SIGQUIT, sighandler2);
66         printf("\n[I] Caught signal %d\n", type);
67         quit();
68         exit(0);
69     }
```

Here is the call graph for this function:



Here is the caller graph for this function:



5.28.1.7 sighandler2()

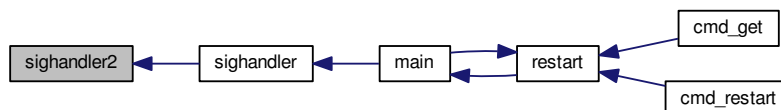
```
void sighandler2 (
    int type )
```

Definition at line 57 of file main.c.

Referenced by `sighandler()`.

```
57     {
58         // Don't use ckb_warn, we want an extra \n at the beginning
59         printf("\n[W] Ignoring signal %d (already shutting down)\n", type);
60     }
```

Here is the caller graph for this function:



5.28.1.8 timespec_add()

```
void timespec_add (
    struct timespec * timespec,
    long nanoseconds )
```

Definition at line 19 of file main.c.

```
19     {
20         nanoseconds += timespec->tv_nsec;
21         timespec->tv_sec += nanoseconds / 1000000000;
22         timespec->tv_nsec = nanoseconds % 1000000000;
23     }
```

5.28.2 Variable Documentation

5.28.2.1 features_mask

```
int features_mask
```

features_mask Mask of features to exclude from all devices

That bit mask ist set to enable all (-1). When interpreting the input parameters, some of these bits can be cleared. At the moment binding, notifying and mouse-acceleration can be disabled via command line. Have a look at [main\(\)](#) in [main.c](#) for details.

Definition at line 35 of file usb.c.

Referenced by [_setupusb\(\)](#), and [main\(\)](#).

5.28.2.2 hwload_mode

```
int hwload_mode
```

Definition at line 7 of file device.c.

Referenced by [main\(\)](#).

5.28.2.3 main_ac

```
int main_ac [static]
```

Definition at line 7 of file main.c.

Referenced by [main\(\)](#), and [restart\(\)](#).

5.28.2.4 main_av

```
char** main_av [static]
```

Definition at line 8 of file main.c.

Referenced by [main\(\)](#), and [restart\(\)](#).

5.28.2.5 reset_stop

```
volatile int reset_stop
```

reset_stop is boolean: Reset stopper for when the program shuts down.

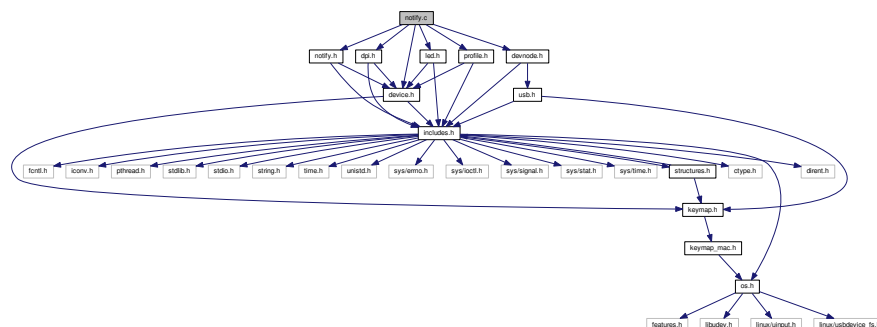
Is set only by `quit()` to true (1) to inform several `usb_*` functions to end their loops and tries.

Definition at line 25 of file usb.c.

Referenced by `_usbrecv()`, `_usb_send()`, `quitWithLock()`, and `usb_tryreset()`.

5.29 notify.c File Reference

```
#include "device.h"
#include "devnode.h"
#include "dpi.h"
#include "led.h"
#include "notify.h"
#include "profile.h"
Include dependency graph for notify.c:
```



Macros

- #define **HWMODE_OR_RETURN**(kb, index)
- #define **HW_STANDARD**

Functions

- void `nprintf` (`usbdevice` *kb, int nodenumber, `usbmode` *mode, const char *format,...)
- void `nprintkey` (`usbdevice` *kb, int nnumber, int keyindex, int down)
- void `nprintind` (`usbdevice` *kb, int nnumber, int led, int on)
- void `cmd_notify` (`usbdevice` *kb, `usbmode` *mode, int nnumber, int keyindex, const char *toggle)
- static void `_cmd_get` (`usbdevice` *kb, `usbmode` *mode, int nnumber, const char *setting)
- void `cmd_get` (`usbdevice` *kb, `usbmode` *mode, int nnumber, int dummy, const char *setting)
- int `restart` ()
- void `cmd_restart` (`usbdevice` *kb, `usbmode` *mode, int nnumber, int dummy, const char *content)

5.29.1 Macro Definition Documentation

5.29.1.1 HW_STANDARD

```
#define HW_STANDARD
```

Value:

```
if (!kb->hw)                                \
    return;                                \
unsigned index = INDEX_OF(mode, profile->mode); \
/* Make sure the mode number is valid */      \
HWMODE_OR_RETURN(kb, index)
```

Definition at line 83 of file notify.c.

Referenced by `_cmd_get()`.

5.29.1.2 HWMODE_OR_RETURN

```
#define HWMODE_OR_RETURN(  
    kb,  
    index )
```

Value:

```
if (IS_K95(kb)) {                            \
    if ((index) >= HWMODE_K95)                \
        return;                              \
    } else {                                  \
        if ((index) >= HWMODE_K70)            \
            return;                          \
    }
```

Definition at line 73 of file notify.c.

5.29.2 Function Documentation

5.29.2.1 _cmd_get()

```
static void _cmd_get (
    usbdevice * kb,
    usbmode * mode,
    int nnumber,
    const char * setting ) [static]
```

Definition at line 90 of file notify.c.

References `dpiset::current`, `usbmode::dpi`, `hwprofile::dpi`, `gethwmodename()`, `gethwprofilename()`, `getid()`, `getmodename()`, `getprofilename()`, `usbdevice::hw`, `usbdevice::hw_ileds`, `HW_STANDARD`, `I_CAPS`, `I_NUM`, `I_SCROLL`, `usbmode::id`, `usbprofile::id`, `hwprofile::id`, `usbdevice::input`, `keymap`, `usbinput::keys`, `dpiset::lift`, `usbmode::light`, `hwprofile::light`, `usbid::modified`, `N_KEYS_INPUT`, `nprintf()`, `nprintfind()`, `nprintfkey()`, `printdpi()`, `printrgb()`, `usbdevice::profile`, and `dpiset::snap`.

Referenced by `cmd_get()`.

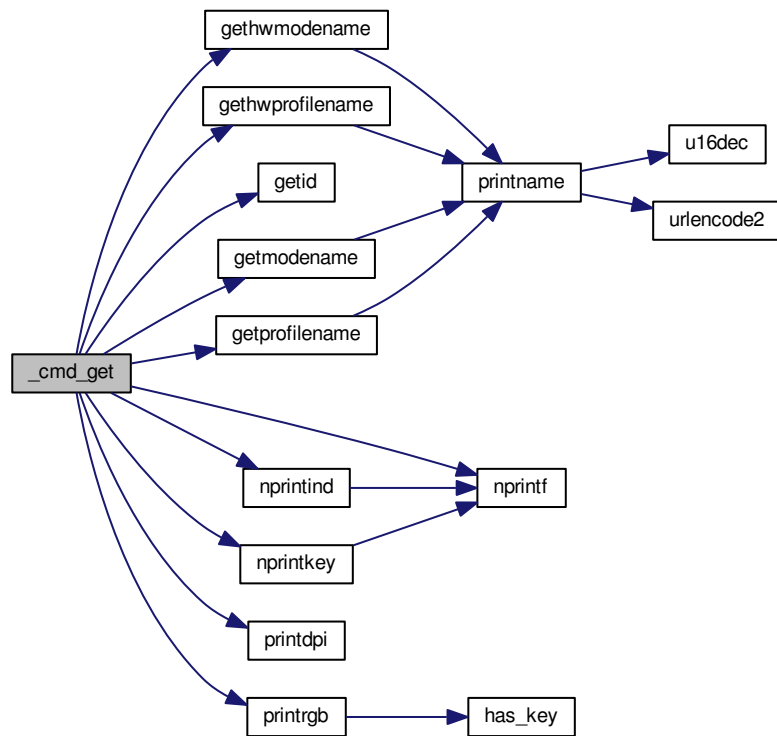
```
90
91     usbprofile* profile = kb->profile;
92     if(!strcmp(setting, ":mode")){
93         // Get the current mode number
94         nprintf(kb, nnumber, mode, "switch\n");
95         return;
96     } else if(!strcmp(setting, ":rgb")){
97         // Get the current RGB settings
98         char* rgb = printrgb(&mode->light, kb);
99         nprintf(kb, nnumber, mode, "rgb %s\n", rgb);
100        free(rgb);
101        return;
102    } else if(!strcmp(setting, ":hwr gb")){
103        // Get the current hardware RGB settings
104        HW_STANDARD;
105        char* rgb = printrgb(kb->hw->light + index, kb);
106        nprintf(kb, nnumber, mode, "hwr gb %s\n", rgb);
107        free(rgb);
108        return;
109    } else if(!strcmp(setting, ":profilename")){
110        // Get the current profile name
111        char* name = getprofilename(profile);
112        nprintf(kb, nnumber, 0, "profilename %s\n", name[0] ? name : "Unnamed");
113        free(name);
114    } else if(!strcmp(setting, ":name")){
115        // Get the current mode name
116        char* name = getmodename(mode);
117        nprintf(kb, nnumber, mode, "name %s\n", name[0] ? name : "Unnamed");
118        free(name);
119    } else if(!strcmp(setting, ":hwprofilename")){
120        // Get the current hardware profile name
121        if(!kb->hw)
122            return;
123        char* name = gethwprofilename(kb->hw);
124        nprintf(kb, nnumber, 0, "hwprofilename %s\n", name[0] ? name : "Unnamed");
125        free(name);
126    } else if(!strcmp(setting, ":hwname")){
127        // Get the current hardware mode name
128        HW_STANDARD;
129        char* name = gethwmodename(kb->hw, index);
130        nprintf(kb, nnumber, mode, "hwname %s\n", name[0] ? name : "Unnamed");
131        free(name);
132    } else if(!strcmp(setting, ":profileid")){
133        // Get the current profile ID
134        char* guid = getid(&profile->id);
135        int modified;
136        memcpy(&modified, &profile->id.modified, sizeof(modified));
137        nprintf(kb, nnumber, 0, "profileid %s %x\n", guid, modified);
138        free(guid);
139    } else if(!strcmp(setting, ":id")){
140        // Get the current mode ID
141        char* guid = getid(&mode->id);
142        int modified;
143        memcpy(&modified, &mode->id.modified, sizeof(modified));
144        nprintf(kb, nnumber, mode, "id %s %x\n", guid, modified);
145        free(guid);
146    } else if(!strcmp(setting, ":hwprofileid")){
147        // Get the current hardware profile ID
148        if(!kb->hw)
```

```

149         return;
150         char* guid = getid(&kb->hw->id[0]);
151         int modified;
152         memcpy(&modified, &kb->hw->id[0].modified, sizeof(modified));
153         nprintf(kb, nnumber, 0, "hwprofileid %s %x\n", guid, modified);
154         free(guid);
155     } else if(!strcmp(setting, ":hwid")){
156         // Get the current hardware mode ID
157         HW_STANDARD;
158         char* guid = getid(&kb->hw->id[index + 1]);
159         int modified;
160         memcpy(&modified, &kb->hw->id[index + 1].modified, sizeof(modified));
161         nprintf(kb, nnumber, mode, "hwid %s %x\n", guid, modified);
162         free(guid);
163     } else if(!strcmp(setting, ":keys")){
164         // Get the current state of all keys
165         for(int i = 0; i < N_KEYS_INPUT; i++){
166             if(!keymap[i].name)
167                 continue;
168             int byte = i / 8, bit = 1 << (i & 7);
169             uchar state = kb->input.keys[byte] & bit;
170             if(state)
171                 nprintkey(kb, nnumber, i, 1);
172         }
173     } else if(!strcmp(setting, ":i")){
174         // Get the current state of all indicator LEDs
175         if(kb->hw_ileds & I_NUM) nprintind(kb, nnumber,
176 I_NUM, 1);
177         if(kb->hw_ileds & I_CAPS) nprintind(kb, nnumber,
178 I_CAPS, 1);
179         if(kb->hw_ileds & I_SCROLL) nprintind(kb, nnumber,
180 I_SCROLL, 1);
181     } else if(!strcmp(setting, ":dpi")){
182         // Get the current DPI levels
183         char* dpi = printdpi(&mode->dpi, kb);
184         nprintf(kb, nnumber, mode, "dpi %s\n", dpi);
185         free(dpi);
186         return;
187     } else if(!strcmp(setting, ":hwdpi")){
188         // Get the current hardware DPI levels
189         HW_STANDARD;
190         char* dpi = printdpi(kb->hw->dpi + index, kb);
191         nprintf(kb, nnumber, mode, "hwdpi %s\n", dpi);
192         free(dpi);
193         return;
194     } else if(!strcmp(setting, ":dpiisel")){
195         // Get the currently-selected DPI
196         nprintf(kb, nnumber, mode, "dpiisel %d\n", mode->dpi.current);
197     } else if(!strcmp(setting, ":hwdpiisel")){
198         // Get the currently-selected hardware DPI
199         HW_STANDARD;
200         nprintf(kb, nnumber, mode, "hwdpiisel %d\n", kb->hw->dpi[index].
201 current);
202     } else if(!strcmp(setting, ":lift")){
203         // Get the mouse lift height
204         nprintf(kb, nnumber, mode, "lift %d\n", mode->dpi.lift);
205     } else if(!strcmp(setting, ":hwlift")){
206         // Get the hardware lift height
207         HW_STANDARD;
208         nprintf(kb, nnumber, mode, "hwlift %d\n", kb->hw->dpi[index].
209 lift);
210     } else if(!strcmp(setting, ":snap")){
211         // Get the angle snap status
212         nprintf(kb, nnumber, mode, "snap %s\n", mode->dpi.snap ? "on" : "off");
213     } else if(!strcmp(setting, ":hwsnap")){
214         // Get the hardware angle snap status
215         HW_STANDARD;
216         nprintf(kb, nnumber, mode, "hwsnap %s\n", kb->hw->dpi[index].
217 snap ? "on" : "off");
218     }
219 }

```

Here is the call graph for this function:



Here is the caller graph for this function:



5.29.2.2 cmd_get()

```

void cmd_get (
    usbdevice * kb,
    usbmode * mode,
    int nnumber,
    int dummy,
    const char * setting )

```

Definition at line 215 of file notify.c.

References `_cmd_get()`, `imutex`, and `restart()`.

5.29.2.4 cmd_restart()

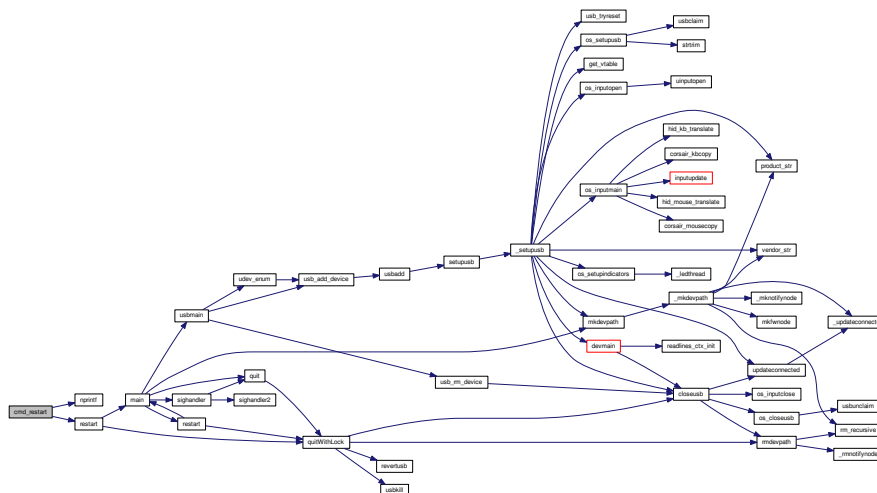
```
void cmd_restart (
    usbdevice * kb,
    usbmode * mode,
    int nnumber,
    int dummy,
    const char * content )
```

Definition at line 223 of file notify.c.

References [ckb_info](#), [nprintf\(\)](#), and [restart\(\)](#).

```
223
224     ckb_info("RESTART called with %s\n", content);
225     nprintf(kb, -1, 0, "RESTART called with %s\n", content);
226     restart();
227 }
```

Here is the call graph for this function:



5.29.2.5 nprintf()

```
void nprintf (
    usbdevice * kb,
    int nodenumber,
    usbmode * mode,
    const char * format,
    ... )
```

Definition at line 8 of file notify.c.

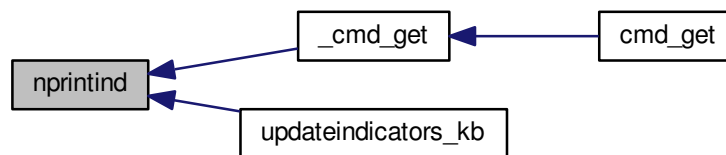
References [INDEX_OF](#), [usbprofile::mode](#), [usbdevice::outfifo](#), [OUTFIFO_MAX](#), and [usbdevice::profile](#).

Referenced by [_cmd_get\(\)](#), [cmd_fwupdate\(\)](#), [cmd_restart\(\)](#), [fwupdate\(\)](#), [nprintfind\(\)](#), and [nprintfkey\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



5.29.2.7 nprintkey()

```

void nprintkey (
    usbdevice * kb,
    int nnumber,
    int keyindex,
    int down )
  
```

Definition at line 35 of file notify.c.

References `keymap`, `key::name`, and `nprintf()`.

Referenced by `_cmd_get()`, and `inputupdate_keys()`.

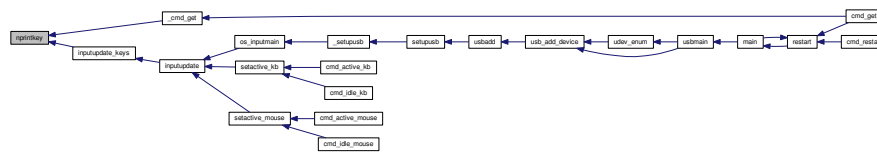
```

35                                     {
36     const key* map = keymap + keyindex;
37     if (map->name)
38         nprintf(kb, nnumber, 0, "key %c%s\n", down ? '+' : '-', map->name);
39     else
40         nprintf(kb, nnumber, 0, "key %c%d\n", down ? '+' : '-', keyindex);
41 }
  
```

Here is the call graph for this function:



Here is the caller graph for this function:



5.29.2.8 restart()

```
int restart ( )
```

Definition at line 228 of file main.c.

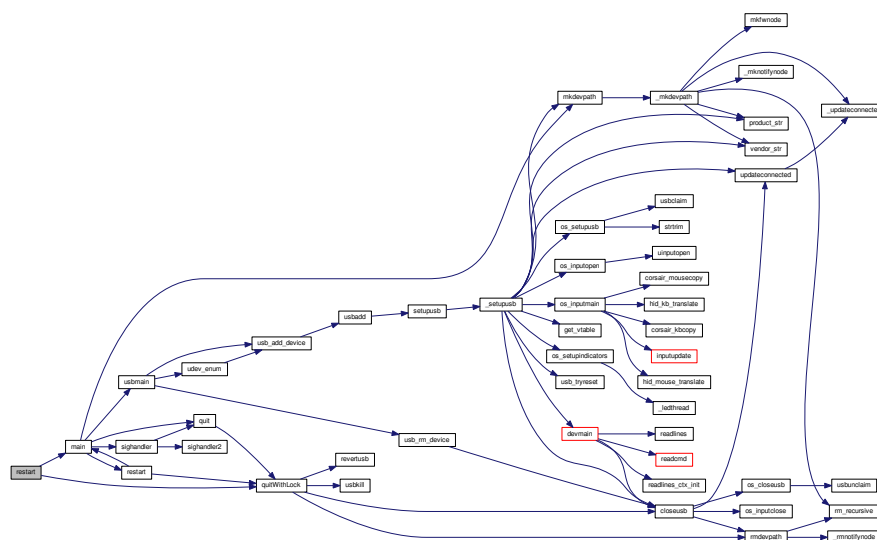
References ckb_err, main(), main_ac, main_av, and quitWithLock().

Referenced by cmd_get(), cmd_restart(), and main().

```

228 {
229     ckb_err("restart called, running quit without mutex-lock.\n");
230     quitWithLock(0);
231     return main(main_ac, main_av);
232 }
  
```

Here is the call graph for this function:



5.30.1.2 cmd_notify()

```
void cmd_notify (
    usbdevice * kb,
    usbmode * mode,
    int nnumber,
    int keyindex,
    const char * toggle )
```

Definition at line 61 of file notify.c.

References CLEAR_KEYBIT, imutex, N_KEYS_INPUT, usbmode::notify, and SET_KEYBIT.

```

61
62     if(keyindex >= N_KEYS_INPUT)
63         return;
64     pthread_mutex_lock(&mutex(kb));
65     if(!strcmp(toggle, "on") || *toggle == 0)
66         SET_KEYBIT(mode->notify[number], keyindex);
67     else if(!strcmp(toggle, "off"))
68         CLEAR_KEYBIT(mode->notify[number], keyindex);
69     pthread_mutex_unlock(&mutex(kb));
70 }

```

5.30.1.3 cmd_restart()

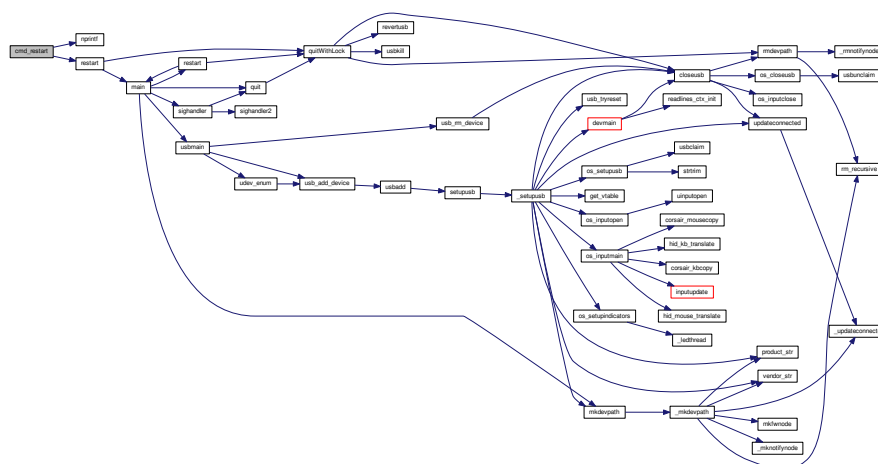
```
void cmd_restart (
    usbdevice * kb,
    usbmode * mode,
    int nnumber,
    int dummy,
    const char * content )
```

Definition at line 223 of file notify.c.

References `ckb_info`, `nprintf()`, and `restart()`.

```
223
224     ckb_info("RESTART called with %s\n", content);
225     nprintf(kb, -1, 0, "RESTART called with %s\n", content);
226     restart();
227 }
```

Here is the call graph for this function:



5.30.1.4 nprintf()

```
void nprintf (
    usbdevice * kb,
    int nodenumber,
    usbmode * mode,
    const char * format,
    ... )
```

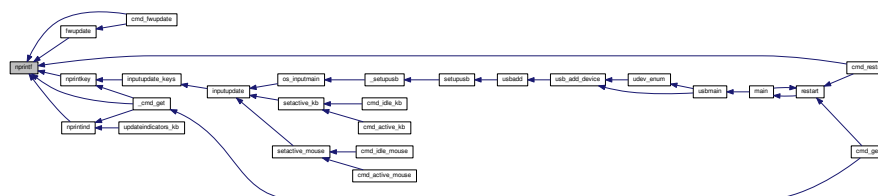
Definition at line 8 of file notify.c.

References INDEX_OF, usbprofile::mode, usbdevice::outfifo, OUTFIFO_MAX, and usbdevice::profile.

Referenced by _cmd_get(), cmd_fwupdate(), cmd_restart(), fwupdate(), nprintind(), and nprintkey().

```
8
9         if(!kb)                                {
10             return;
11         usbprofile* profile = kb->profile;
12         va_list va_args;
13         int fifo;
14         if(nodenumber >= 0){
15             // If node number was given, print to that node (if open)
16             if((fifo = kb->outfifo[nodenumber] - 1) != -1){
17                 va_start(va_args, format);
18                 if(mode)
19                     dprintf(fifo, "mode %d ", INDEX_OF(mode, profile->mode) + 1);
20                 vdprintf(fifo, format, va_args);
21             }
22             return;
23         }
24         // Otherwise, print to all nodes
25         for(int i = 0; i < OUTFIFO_MAX; i++){
26             if((fifo = kb->outfifo[i] - 1) != -1){
27                 va_start(va_args, format);
28                 if(mode)
29                     dprintf(fifo, "mode %d ", INDEX_OF(mode, profile->mode) + 1);
30                 vdprintf(fifo, format, va_args);
31             }
32         }
33     }
```

Here is the caller graph for this function:



5.30.1.5 nprintind()

```
void nprintind (
    usbdevice * kb,
    int nnumber,
    int led,
    int on )
```

Definition at line 43 of file notify.c.

References I_CAPS, I_NUM, I_SCROLL, and nprintf().

Referenced by _cmd_get(), and updateindicators_kb().

```

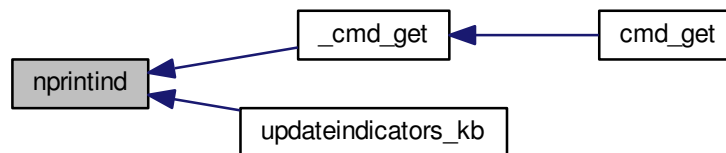
43                                     {
44     const char* name = 0;
45     switch(led){
46     case I_NUM:
47         name = "num";
48         break;
49     case I_CAPS:
50         name = "caps";
51         break;
52     case I_SCROLL:
53         name = "scroll";
54         break;
55     default:
56         return;
57     }
58     nprintf(kb, nnumber, 0, "i %c%s\n", on ? '+' : '-', name);
59 }

```

Here is the call graph for this function:



Here is the caller graph for this function:



5.30.1.6 nprintkey()

```

void nprintkey (
    usbdevice * kb,
    int nnumber,
    int keyindex,
    int down )

```

Definition at line 35 of file notify.c.

References keymap, key::name, and nprintf().

Referenced by _cmd_get(), and inputupdate_keys().

```

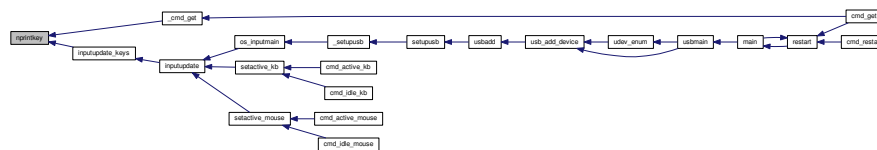
35                                     {
36     const key* map = keymap + keyindex;
37     if (map->name)
38         nprintf(kb, nnumber, 0, "key %c%s\n", down ? '+' : '-', map->name);
39     else
40         nprintf(kb, nnumber, 0, "key %c#%d\n", down ? '+' : '-', keyindex);
41 }

```

Here is the call graph for this function:



Here is the caller graph for this function:



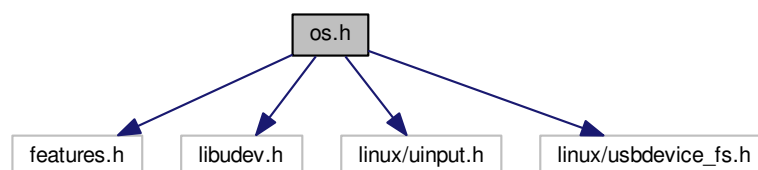
5.31 os.h File Reference

```

#include <features.h>
#include <libudev.h>
#include <linux/uinput.h>
#include <linux/usbdevice_fs.h>

```

Include dependency graph for `os.h`:



- #define _DEFAULT_SOURCE
- #define _GNU_SOURCE
- #define UINPUT_VERSION 2
- #define euuid_guard_start
- #define euuid_guard_stop

5.31.1.1 _DEFAULT_SOURCE

Definition at line 22 of file os.h.

Definition at line 26 of file os.h.

Referenced by `mkdevpath()`, `mknotifynode()`, `rmdevpath()`, `rmnotifynode()`, and `updateconnected()`.

5.31.1.4 `euid_guard_stop`

```
#define euid_guard_stop
```

Definition at line 41 of file `os.h`.

Referenced by `mkdevpath()`, `mknotifynode()`, `rmdevpath()`, `rmnotifynode()`, and `updateconnected()`.

5.31.1.5 `UINPUT_VERSION`

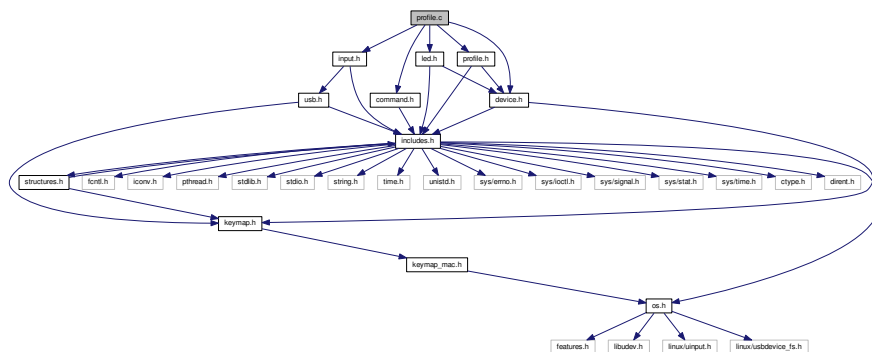
```
#define UINPUT_VERSION 2
```

Definition at line 35 of file `os.h`.

5.32 `profile.c` File Reference

```
#include "command.h"
#include "device.h"
#include "input.h"
#include "led.h"
#include "profile.h"
```

Include dependency graph for `profile.c`:



Functions

- void `urldecode2` (char *dst, const char *src)
- void `urlencode2` (char *dst, const char *src)
- int `setid` (usbid *id, const char *guid)
- char * `getid` (usbid *id)
- void `u16enc` (char *in, ushort *out, size_t *srclen, size_t *dstlen)
- void `u16dec` (ushort *in, char *out, size_t *srclen, size_t *dstlen)
- void `cmd_name` (usbdevice *kb, usbmode *mode, int dummy1, int dummy2, const char *name)
- void `cmd_profilename` (usbdevice *kb, usbmode *dummy1, int dummy2, int dummy3, const char *name)
- char * `printname` (ushort *name, int length)
- char * `getmodename` (usbmode *mode)

- char * [getprofilename](#) (usbprofile *profile)
- char * [gethwmodename](#) (hwprofile *profile, int index)
- char * [gethwprofilename](#) (hwprofile *profile)
- void [cmd_id](#) (usbdevice *kb, usbmode *mode, int dummy1, int dummy2, const char *id)
- void [cmd_profileid](#) (usbdevice *kb, usbmode *mode, int dummy1, int dummy2, const char *id)
- static void [initmode](#) (usbmode *mode)
- void [allocprofile](#) (usbdevice *kb)
- int [loadprofile](#) (usbdevice *kb)
- static void [freemode](#) (usbmode *mode)
- void [cmd_erase](#) (usbdevice *kb, usbmode *mode, int dummy1, int dummy2, const char *dummy3)
- static void [_freeprofile](#) (usbdevice *kb)
- void [cmd_eraseprofile](#) (usbdevice *kb, usbmode *dummy1, int dummy2, int dummy3, const char *dummy4)
- void [freeprofile](#) (usbdevice *kb)
- void [hwtonative](#) (usbprofile *profile, hwprofile *hw, int modecount)
- void [nativetohw](#) (usbprofile *profile, hwprofile *hw, int modecount)

Variables

- static iconv_t [utf8to16](#) = 0
- static iconv_t [utf16to8](#) = 0

5.32.1 Function Documentation

5.32.1.1 _freeprofile()

```
static void _freeprofile (
    usbdevice * kb ) [static]
```

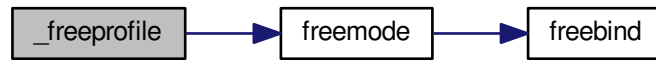
Definition at line 210 of file profile.c.

References [freemode\(\)](#), [usbprofile::mode](#), [MODE_COUNT](#), and [usbdevice::profile](#).

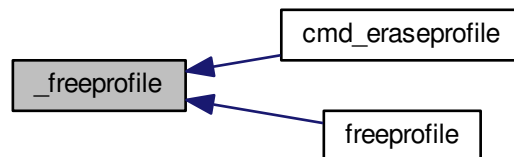
Referenced by [cmd_eraseprofile\(\)](#), and [freeprofile\(\)](#).

```
210
211     usbprofile* profile = kb->profile; {
212     if(!profile)
213         return;
214     // Clear all mode data
215     for(int i = 0; i < MODE_COUNT; i++)
216         freemode(profile->mode + i);
217     free(profile);
218     kb->profile = 0;
219 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



5.32.1.2 allocprofile()

```
void allocprofile (
    usbdevice * kb )
```

Definition at line 182 of file profile.c.

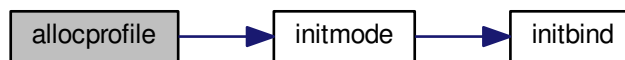
References `usbprofile::currentmode`, `dpiset::forceupdate`, `lighting::forceupdate`, `initmode()`, `usbprofile::lastdpi`, `usbprofile::lastlight`, `usbprofile::mode`, `MODE_COUNT`, and `usbdevice::profile`.

Referenced by `cmd_eraseprofile()`.

```

182                                     {
183     if(kb->profile)
184         return;
185     usbprofile* profile = kb->profile = calloc(1, sizeof(
usbprofile));
186     for(int i = 0; i < MODE_COUNT; i++)
187         initmode(profile->mode + i);
188     profile->currentmode = profile->mode;
189     profile->lastlight.forceupdate = profile->lastdpi.
forceupdate = 1;
190 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



5.32.1.3 cmd_erase()

```

void cmd_erase (
    usbdevice * kb,
    usbmode * mode,
    int dummy1,
    int dummy2,
    const char * dummy3 )
  
```

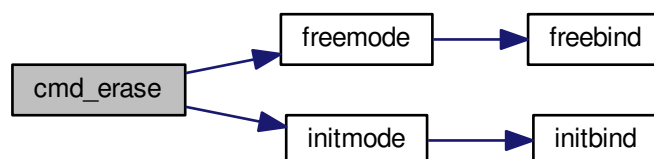
Definition at line 203 of file profile.c.

References `freemode()`, `imutex`, and `initmode()`.

```

203 {
204     pthread_mutex_lock(imutex(kb));
205     freemode(mode);
206     initmode(mode);
207     pthread_mutex_unlock(imutex(kb));
208 }
  
```

Here is the call graph for this function:



5.32.1.4 cmd_eraseprofile()

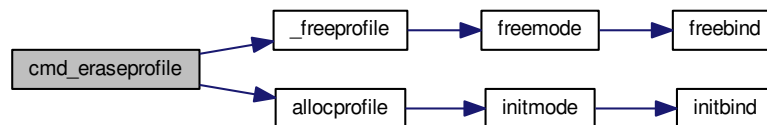
```
void cmd_eraseprofile (
    usbdevice * kb,
    usbmode * dummy1,
    int dummy2,
    int dummy3,
    const char * dummy4 )
```

Definition at line 221 of file profile.c.

References `_freeprofile()`, `allocprofile()`, and `imutex`.

```
221                                     {
222     pthread_mutex_lock(imutex(kb));
223     _freeprofile(kb);
224     allocprofile(kb);
225     pthread_mutex_unlock(imutex(kb));
226 }
```

Here is the call graph for this function:



5.32.1.5 cmd_id()

```
void cmd_id (
    usbdevice * kb,
    usbmode * mode,
    int dummy1,
    int dummy2,
    const char * id )
```

Definition at line 160 of file profile.c.

References `usbmode::id`, `usbid::modified`, and `setid()`.

```
160                                     {
161     // ID is either a GUID or an 8-digit hex number
162     int newmodified;
163     if(!setid(&mode->id, id) && sscanf(id, "%08x", &newmodified) == 1)
164         memcpy(mode->id.modified, &newmodified, sizeof(newmodified));
165 }
```

Here is the call graph for this function:



5.32.1.6 cmd_name()

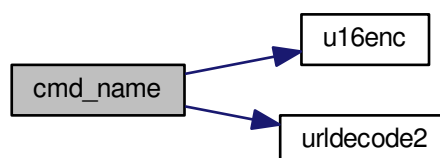
```
void cmd_name (  
    usbdevice * kb,  
    usbmode * mode,  
    int dummy1,  
    int dummy2,  
    const char * name )
```

Definition at line 117 of file profile.c.

References MD_NAME_LEN, usbmode::name, u16enc(), and urldecode2().

```
117                                     {  
118     char decoded[strlen(name) + 1];  
119     urldecode2(decoded, name);  
120     size_t srclen = strlen(decoded), dstlen = MD_NAME_LEN;  
121     u16enc(decoded, mode->name, &srclen, &dstlen);  
122 }
```

Here is the call graph for this function:



5.32.1.7 cmd_profileid()

```
void cmd_profileid (
    usbdevice * kb,
    usbmode * mode,
    int dummy1,
    int dummy2,
    const char * id )
```

Definition at line 167 of file profile.c.

References `usbprofile::id`, `usbid::modified`, `usbdevice::profile`, and `setid()`.

```
167                                     {
168     usbprofile* profile = kb->profile;
169     int newmodified;
170     if(!setid(&profile->id, id) && sscanf(id, "%08x", &newmodified) == 1)
171         memcpy(profile->id.modified, &newmodified, sizeof(newmodified));
172
173 }
```

Here is the call graph for this function:



5.32.1.8 cmd_profilename()

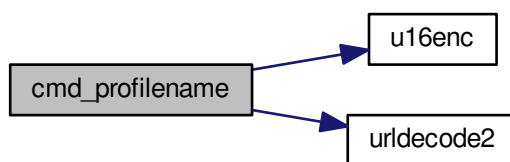
```
void cmd_profilename (
    usbdevice * kb,
    usbmode * dummy1,
    int dummy2,
    int dummy3,
    const char * name )
```

Definition at line 124 of file profile.c.

References `usbprofile::name`, `PR_NAME_LEN`, `usbdevice::profile`, `u16enc()`, and `urldecode2()`.

```
124                                     {
125     usbprofile* profile = kb->profile;
126     char decoded[strlen(name) + 1];
127     urldecode2(decoded, name);
128     size_t srclen = strlen(decoded), dstlen = PR_NAME_LEN;
129     u16enc(decoded, profile->name, &srclen, &dstlen);
130 }
```

Here is the call graph for this function:



5.32.1.9 freemode()

```
static void freemode (  
    usbmode * mode ) [static]
```

Definition at line 198 of file profile.c.

References `usbmode::bind`, and `freebind()`.

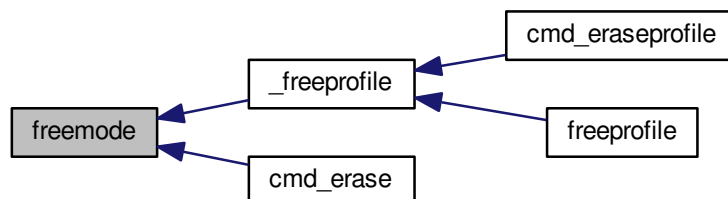
Referenced by `_freeprofile()`, and `cmd_erase()`.

```
198 {  
199     freebind(&mode->bind);  
200     memset(mode, 0, sizeof(*mode));  
201 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



5.32.1.10 freeprofile()

```
void freeprofile (
    usbdevice * kb )
```

Definition at line 228 of file profile.c.

References `_freeprofile()`, and `usbdevice::hw`.

```
228                                     {
229     _freeprofile(kb);
230     // Also free HW profile
231     free(kb->hw);
232     kb->hw = 0;
233 }
```

Here is the call graph for this function:



5.32.1.11 gethwmodename()

```
char* gethwmodename (
    hwprofile * profile,
    int index )
```

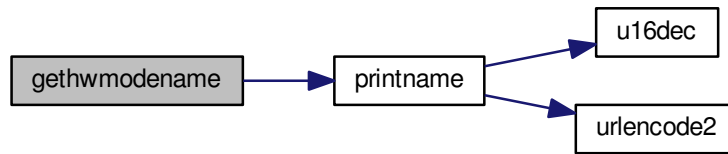
Definition at line 152 of file profile.c.

References `MD_NAME_LEN`, `hwprofile::name`, and `printname()`.

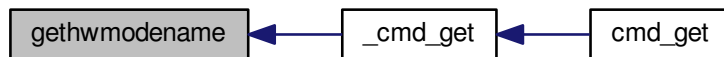
Referenced by `_cmd_get()`.

```
152                                     {
153     return printname(profile->name[index + 1], MD_NAME_LEN);
154 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



5.32.1.12 gethwprofilename()

```
char* gethwprofilename (  
    hwprofile * profile )
```

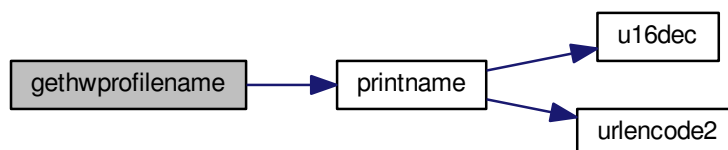
Definition at line 156 of file profile.c.

References `MD_NAME_LEN`, `hwprofile::name`, and `printname()`.

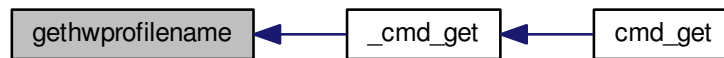
Referenced by `_cmd_get()`.

```
156 {  
157     return printname(profile->name[0], MD_NAME_LEN);  
158 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



5.32.1.13 `getid()`

```
char* getid (
    usbid * id )
```

Definition at line 79 of file `profile.c`.

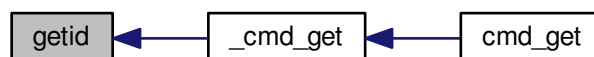
References `usbid::guid`.

Referenced by `_cmd_get()`.

```

79      {
80      int32_t data1;
81      int16_t data2, data3, data4a;
82      char data4b[6];
83      memcpy(&data1, id->guid + 0x0, 4);
84      memcpy(&data2, id->guid + 0x4, 2);
85      memcpy(&data3, id->guid + 0x6, 2);
86      memcpy(&data4a, id->guid + 0x8, 2);
87      memcpy(data4b, id->guid + 0xA, 6);
88      char* guid = malloc(39);
89      snprintf(guid, 39, "{%08X-%04hX-%04hX-%04hX-%02hhX%02hhX%02hhX%02hhX%02hhX%02hhX}",
90              data1, data2, data3, data4a, data4b[0], data4b[1], data4b[2], data4b[3], data4b[4], data4b[5])
91      ;
92      return guid;
93  }
```

Here is the caller graph for this function:



5.32.1.14 getmodename()

```
char* getmodename (  
    usbmode * mode )
```

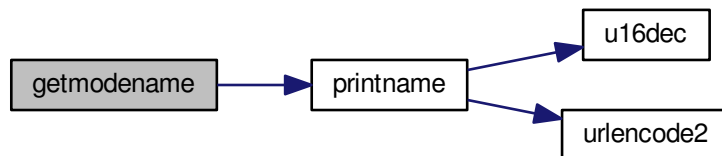
Definition at line 144 of file profile.c.

References MD_NAME_LEN, usbmode::name, and printname().

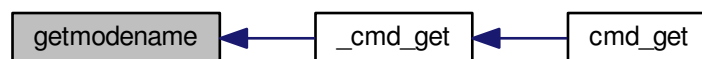
Referenced by _cmd_get().

```
144  
145     return printname(mode->name, MD_NAME_LEN);  
146 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



5.32.1.15 getprofilename()

```
char* getprofilename (  
    usbprofile * profile )
```

Definition at line 148 of file profile.c.

References usbprofile::name, PR_NAME_LEN, and printname().

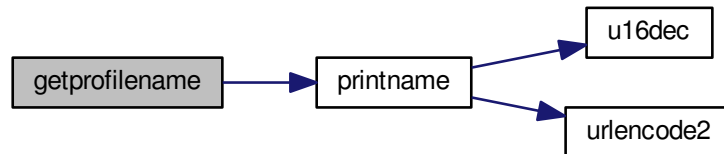
Referenced by _cmd_get().

```

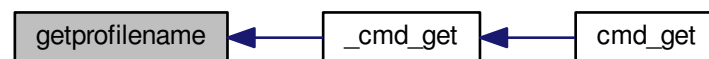
148     {
149     return printname(profile->name, PR_NAME_LEN);
150 }

```

Here is the call graph for this function:



Here is the caller graph for this function:



5.32.1.16 hwtonative()

```

void hwtonative (
    usbprofile * profile,
    hwprofile * hw,
    int modecount )

```

Definition at line 235 of file profile.c.

References `usbmode::dpi`, `hwprofile::dpi`, `dpiset::forceupdate`, `lighting::forceupdate`, `usbmode::id`, `usbprofile::id`, `hwprofile::id`, `usbprofile::lastdpi`, `usbprofile::lastlight`, `usbmode::light`, `hwprofile::light`, `MD_NAME_LEN`, `usbprofile::mode`, `usbmode::name`, `usbprofile::name`, `hwprofile::name`, and `PR_NAME_LEN`.

Referenced by `cmd_hwload_kb()`, and `cmd_hwload_mouse()`.

```

235     {
236     // Copy the profile name and ID
237     memcpy(profile->name, hw->name[0], PR_NAME_LEN * 2);
238     memcpy(&profile->id, hw->id, sizeof(usbid));
239     // Copy the mode settings
240     for(int i = 0; i < modecount; i++){
241         usbmode* mode = profile->mode + i;
242         memcpy(mode->name, hw->name[i + 1], MD_NAME_LEN * 2);

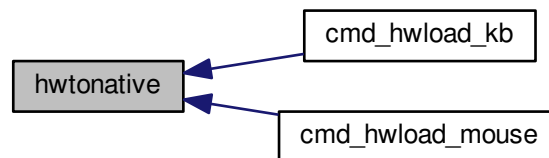
```

```

243     memcpy(&mode->id, hw->id + i + 1, sizeof(usbid));
244     memcpy(&mode->light, hw->light + i, sizeof(lighting));
245     memcpy(&mode->dpi, hw->dpi + i, sizeof(dpiset));
246     // Set a force update on the light/DPI since they've been overwritten
247     mode->light.forceupdate = mode->dpi.forceupdate = 1;
248 }
249 profile->lastlight.forceupdate = profile->lastdpi.
    forceupdate = 1;
250 }

```

Here is the caller graph for this function:



5.32.1.17 initmode()

```

static void initmode (
    usbmode * mode ) [static]

```

Definition at line 175 of file profile.c.

References `usbmode::bind`, `usbmode::dpi`, `dpiset::forceupdate`, `lighting::forceupdate`, `initbind()`, and `usbmode::light`.

Referenced by `allocprofile()`, and `cmd_erase()`.

```

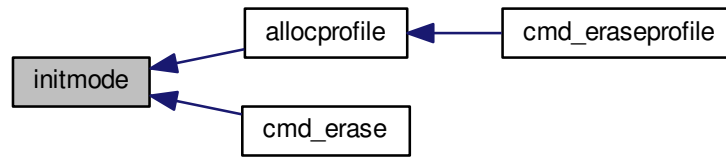
175     {
176     memset(mode, 0, sizeof(*mode));
177     mode->light.forceupdate = 1;
178     mode->dpi.forceupdate = 1;
179     initbind(&mode->bind);
180 }

```

Here is the call graph for this function:



Here is the caller graph for this function:



5.32.1.18 loadprofile()

```
int loadprofile (
    usbdevice * kb )
```

Definition at line 192 of file profile.c.

References `hwloadprofile`.

```
192 {
193     if(hwloadprofile(kb, 1))
194         return -1;
195     return 0;
196 }
```

5.32.1.19 nativetohw()

```
void nativetohw (
    usbprofile * profile,
    hwprofile * hw,
    int modecount )
```

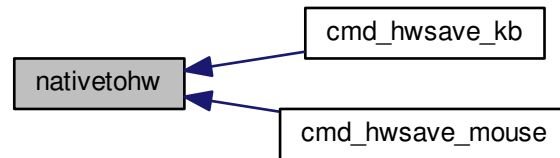
Definition at line 252 of file profile.c.

References `usbmode::dpi`, `hwprofile::dpi`, `usbmode::id`, `usbprofile::id`, `hwprofile::id`, `usbmode::light`, `hwprofile::light`, `MD_NAME_LEN`, `usbprofile::mode`, `usbmode::name`, `usbprofile::name`, `hwprofile::name`, and `PR_NAME_LEN`.

Referenced by `cmd_hwsave_kb()`, and `cmd_hwsave_mouse()`.

```
252 {
253     // Copy name and ID
254     memcpy(hw->name[0], profile->name, PR_NAME_LEN * 2);
255     memcpy(hw->id, &profile->id, sizeof(usbid));
256     // Copy the mode settings
257     for(int i = 0; i < modecount; i++){
258         usbmode* mode = profile->mode + i;
259         memcpy(hw->name[i + 1], mode->name, MD_NAME_LEN * 2);
260         memcpy(hw->id + i + 1, &mode->id, sizeof(usbid));
261         memcpy(hw->light + i, &mode->light, sizeof(lighting));
262         memcpy(hw->dpi + i, &mode->dpi, sizeof(dpi));
263     }
264 }
```

Here is the caller graph for this function:



5.32.1.20 printname()

```
char* printname (  
    ushort * name,  
    int length )
```

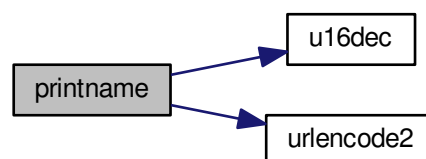
Definition at line 132 of file profile.c.

References `u16dec()`, and `urlencode2()`.

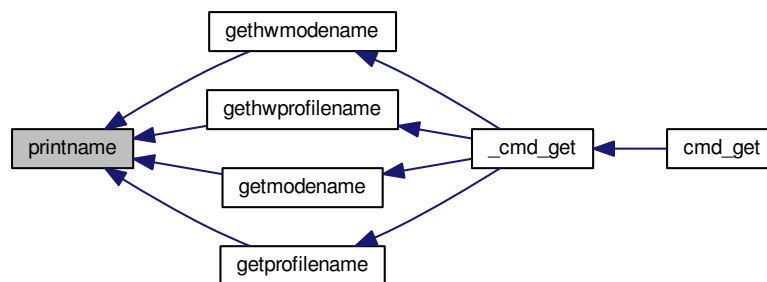
Referenced by `gethwmodename()`, `gethwprofilename()`, `getmodename()`, and `getprofilename()`.

```
132                                     {  
133     // Convert the name to UTF-8  
134     char* buffer = calloc(1, length * 4 - 3);  
135     size_t srclen = length, dstlen = length * 4 - 4;  
136     u16dec(name, buffer, &srclen, &dstlen);  
137     // URL-encode it  
138     char* buffer2 = malloc(strlen(buffer) * 3 + 1);  
139     urlencode2(buffer2, buffer);  
140     free(buffer);  
141     return buffer2;  
142 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



5.32.1.21 `setid()`

```
int setid (
    usbid * id,
    const char * guid )
```

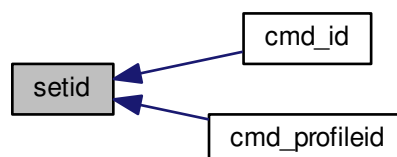
Definition at line 64 of file `profile.c`.

References `usbid::guid`.

Referenced by `cmd_id()`, and `cmd_profileid()`.

```
64     {
65         int32_t data1;
66         int16_t data2, data3, data4a;
67         char data4b[6];
68         if(sscanf(guid, "%08X-%04hX-%04hX-%02hhX%02hhX%02hhX%02hhX%02hhX",
69             &data1, &data2, &data3, &data4a, data4b, data4b + 1, data4b + 2, data4b + 3, data4b + 4,
70             data4b + 5) != 10)
71             return 0;
72         memcpy(id->guid + 0x0, &data1, 4);
73         memcpy(id->guid + 0x4, &data2, 2);
74         memcpy(id->guid + 0x6, &data3, 2);
75         memcpy(id->guid + 0x8, &data4a, 2);
76         memcpy(id->guid + 0xA, data4b, 6);
77         return 1;
78     }
```

Here is the caller graph for this function:



5.32.1.22 u16dec()

```
void u16dec (
    ushort * in,
    char * out,
    size_t * srclen,
    size_t * dstlen )
```

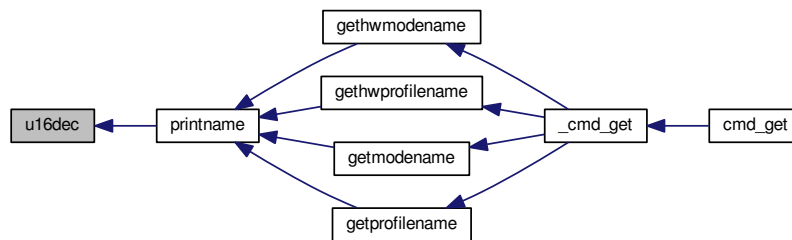
Definition at line 105 of file profile.c.

References [utf16to8](#).

Referenced by [printname\(\)](#).

```
105                                     {
106     if(!utf16to8)
107         utf16to8 = iconv_open("UTF-8", "UTF-16LE");
108     size_t srclen2 = 0, srclenmax = *srclen;
109     for(; srclen2 < srclenmax; srclen2++){
110         if(!in[srclen2])
111             break;
112     }
113     *srclen = srclen2 * 2;
114     iconv(utf16to8, (char**)&in, srclen, &out, dstlen);
115 }
```

Here is the caller graph for this function:



5.32.1.23 u16enc()

```
void u16enc (
    char * in,
    ushort * out,
    size_t * srclen,
    size_t * dstlen )
```

Definition at line 97 of file profile.c.

References [utf8to16](#).

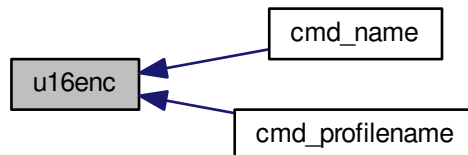
Referenced by [cmd_name\(\)](#), and [cmd_profilename\(\)](#).

```

97                                     {
98     if(!utf8tol6)
99         utf8tol6 = iconv_open("UTF-16LE", "UTF-8");
100     memset(out, 0, *dstlen * 2);
101     *dstlen = *dstlen * 2 - 2;
102     iconv(utf8tol6, &in, srclen, (char**)&out, dstlen);
103 }

```

Here is the caller graph for this function:



5.32.1.24 urldecode2()

```

void urldecode2 (
    char * dst,
    const char * src )

```

Definition at line 8 of file profile.c.

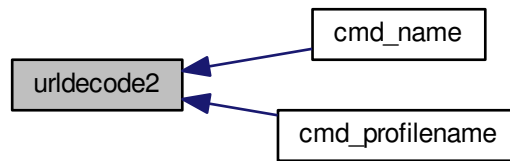
Referenced by cmd_name(), and cmd_profilename().

```

8                                     {
9     char a, b;
10    char s;
11    while((s = *src)){
12        if((s == '%') &&
13            ((a = src[1]) && (b = src[2])) &&
14            (isxdigit(a) && isxdigit(b))){
15            if(a >= 'a')
16                a -= 'a' - 'A';
17            if(a >= 'A')
18                a -= 'A' - 10;
19            else
20                a -= '0';
21            if(b >= 'a')
22                b -= 'a' - 'A';
23            if(b >= 'A')
24                b -= 'A' - 10;
25            else
26                b -= '0';
27            *dst++ = 16 * a + b;
28            src += 3;
29        } else {
30            *dst++ = s;
31            src++;
32        }
33    }
34    *dst = '\0';
35 }

```

Here is the caller graph for this function:



5.32.1.25 urlencode2()

```
void urlencode2 (
    char * dst,
    const char * src )
```

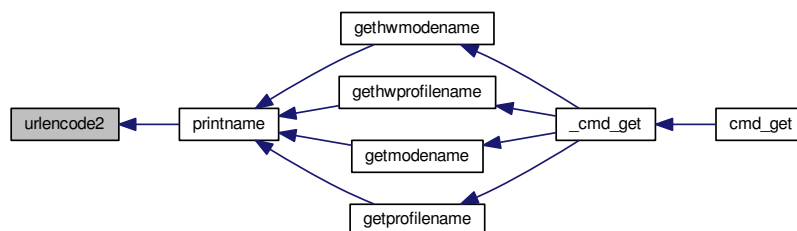
Definition at line 37 of file profile.c.

Referenced by `printname()`.

```

37                                     {
38     char s;
39     while((s = *src++){
40         if(s <= ' ' || s == '/' ||
41            (s >= ':' && s <= '@') ||
42            s == '[' || s == ']' ||
43            s >= 0x7F) {
44         char a = s >> 4, b = s & 0xF;
45         if(a >= 10)
46             a += 'A' - 10;
47         else
48             a += '0';
49         if(b >= 10)
50             b += 'A' - 10;
51         else
52             b += '0';
53         dst[0] = '%';
54         dst[1] = a;
55         dst[2] = b;
56         dst += 3;
57     } else
58         *dst++ = s;
59     }
60     *dst = '\0';
61 }
```

Here is the caller graph for this function:



5.32.2 Variable Documentation

5.32.2.1 utf16to8

```
iconv_t utf16to8 = 0 [static]
```

Definition at line 95 of file profile.c.

Referenced by u16dec().

5.32.2.2 utf8to16

```
iconv_t utf8to16 = 0 [static]
```

Definition at line 95 of file profile.c.

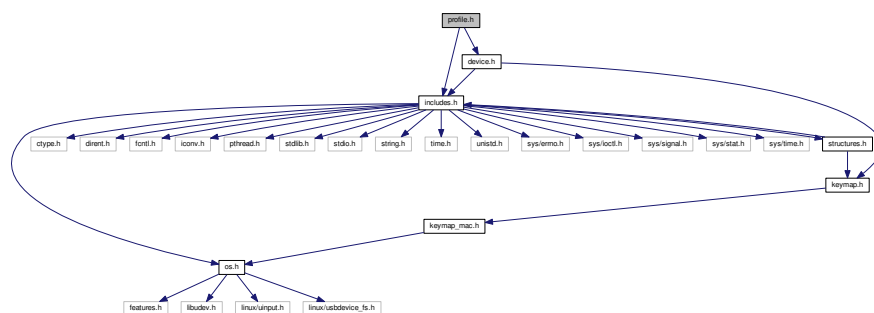
Referenced by u16enc().

5.33 profile.h File Reference

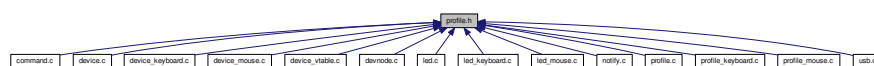
```
#include "includes.h"
```

```
#include "device.h"
```

Include dependency graph for profile.h:



This graph shows which files directly or indirectly include this file:



Macros

- `#define hwloadprofile(kb, apply) (kb)->vtable->hwload(kb, 0, 0, apply, 0)`

Functions

- void `allocprofile (usbdevice *kb)`
- int `loadprofile (usbdevice *kb)`
- void `freeprofile (usbdevice *kb)`
- void `cmd_erase (usbdevice *kb, usbmode *mode, int dummy1, int dummy2, const char *dummy3)`
- void `cmd_eraseprofile (usbdevice *kb, usbmode *dummy1, int dummy2, int dummy3, const char *dummy4)`
- void `cmd_name (usbdevice *kb, usbmode *mode, int dummy1, int dummy2, const char *name)`
- void `cmd_profilename (usbdevice *kb, usbmode *dummy1, int dummy2, int dummy3, const char *name)`
- char * `getmodename (usbmode *mode)`
- char * `getprofilename (usbprofile *profile)`
- char * `gethwmodename (hwprofile *profile, int index)`
- char * `gethwprofilename (hwprofile *profile)`
- int `setid (usbid *id, const char *guid)`
- char * `getid (usbid *id)`
- void `hwtonative (usbprofile *profile, hwprofile *hw, int modecount)`
- void `nativetohw (usbprofile *profile, hwprofile *hw, int modecount)`
- void `cmd_id (usbdevice *kb, usbmode *mode, int dummy1, int dummy2, const char *id)`
- void `cmd_profileid (usbdevice *kb, usbmode *mode, int dummy1, int dummy2, const char *id)`
- int `cmd_hwload_kb (usbdevice *kb, usbmode *dummy1, int dummy2, int apply, const char *dummy3)`
- int `cmd_hwload_mouse (usbdevice *kb, usbmode *dummy1, int dummy2, int apply, const char *dummy3)`
- int `cmd_hwsave_kb (usbdevice *kb, usbmode *dummy1, int dummy2, int dummy3, const char *dummy4)`
- int `cmd_hwsave_mouse (usbdevice *kb, usbmode *dummy1, int dummy2, int dummy3, const char *dummy4)`

5.33.1 Macro Definition Documentation

5.33.1.1 hwloadprofile

```
#define hwloadprofile(  
    kb,  
    apply ) (kb)->vtable->hwload(kb, 0, 0, apply, 0)
```

Definition at line 52 of file profile.h.

Referenced by `_start_dev()`, and `loadprofile()`.

5.33.2 Function Documentation

5.33.2.1 allocprofile()

```
void allocprofile (
    usbdevice * kb )
```

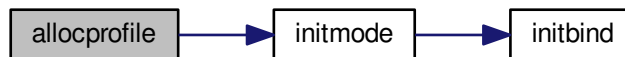
Definition at line 182 of file profile.c.

References `usbprofile::currentmode`, `dpiset::forceupdate`, `lighting::forceupdate`, `initmode()`, `usbprofile::lastdpi`, `usbprofile::lastlight`, `usbprofile::mode`, `MODE_COUNT`, and `usbdevice::profile`.

Referenced by `cmd_eraseprofile()`.

```
182                                     {
183     if(kb->profile)
184         return;
185     usbprofile* profile = kb->profile = calloc(1, sizeof(
186         usbprofile));
187     for(int i = 0; i < MODE_COUNT; i++)
188         initmode(profile->mode + i);
189     profile->currentmode = profile->mode;
190     profile->lastlight.forceupdate = profile->lastdpi.
191     forceupdate = 1;
192 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



5.33.2.2 cmd_erase()

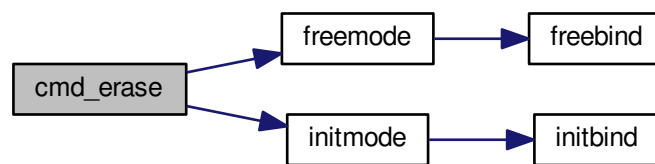
```
void cmd_erase (
    usbdevice * kb,
    usbmode * mode,
    int dummy1,
    int dummy2,
    const char * dummy3 )
```

Definition at line 203 of file profile.c.

References `freemode()`, `imutex`, and `initmode()`.

```
203                                     {
204     pthread_mutex_lock(imutex(kb));
205     freemode(mode);
206     initmode(mode);
207     pthread_mutex_unlock(imutex(kb));
208 }
```

Here is the call graph for this function:



5.33.2.3 cmd_eraseprofile()

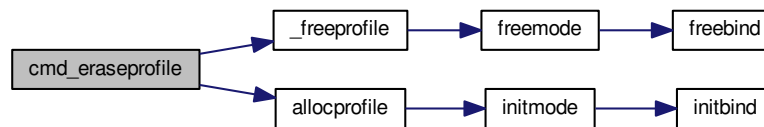
```
void cmd_eraseprofile (
    usbdevice * kb,
    usbmode * dummy1,
    int dummy2,
    int dummy3,
    const char * dummy4 )
```

Definition at line 221 of file profile.c.

References `_freeprofile()`, `allocprofile()`, and `imutex`.

```
221                                     {
222     pthread_mutex_lock(imutex(kb));
223     _freeprofile(kb);
224     allocprofile(kb);
225     pthread_mutex_unlock(imutex(kb));
226 }
```

Here is the call graph for this function:



5.33.2.4 cmd_hwload_kb()

```

int cmd_hwload_kb (
    usbdevice * kb,
    usbmode * dummy1,
    int dummy2,
    int apply,
    const char * dummy3 )

```

Definition at line 16 of file `profile_keyboard.c`.

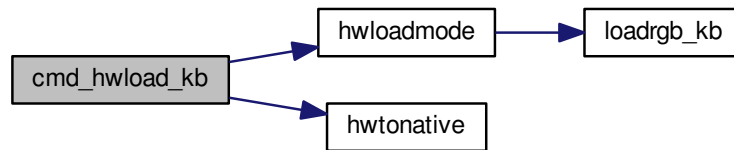
References `DELAY_LONG`, `usbdevice::hw`, `hwloadmode()`, `HWMODE_K70`, `HWMODE_K95`, `hwtonative()`, `hwprofile::id`, `IS_K95`, `MSG_SIZE`, `hwprofile::name`, `PR_NAME_LEN`, `usbdevice::profile`, and `usbrecv`.

```

16                                     {
17     DELAY_LONG(kb);
18     hwprofile* hw = calloc(1, sizeof(hwprofile));
19     // Ask for profile and mode IDs
20     uchar data_pkt[2][MSG_SIZE] = {
21         { 0x0e, 0x15, 0x01, 0 },
22         { 0x0e, 0x16, 0x01, 0 }
23     };
24     uchar in_pkt[MSG_SIZE];
25     int modes = (IS_K95(kb) ? HWMODE_K95 : HWMODE_K70);
26     for(int i = 0; i <= modes; i++){
27         data_pkt[0][3] = i;
28         if(!usbrecv(kb, data_pkt[0], in_pkt)){
29             free(hw);
30             return -1;
31         }
32         memcpy(hw->id + i, in_pkt + 4, sizeof(usbid));
33     }
34     // Ask for profile name
35     if(!usbrecv(kb, data_pkt[1], in_pkt)){
36         free(hw);
37         return -1;
38     }
39     memcpy(hw->name[0], in_pkt + 4, PR_NAME_LEN * 2);
40     // Load modes
41     for(int i = 0; i < modes; i++){
42         if(hwloadmode(kb, hw, i)){
43             free(hw);
44             return -1;
45         }
46     }
47     // Make the profile active (if requested)
48     if(apply)
49         hwtonative(kb->profile, hw, modes);
50     // Free the existing profile (if any)
51     free(kb->hw);
52     kb->hw = hw;
53     DELAY_LONG(kb);
54     return 0;
55 }

```

Here is the call graph for this function:



5.33.2.5 cmd_hwload_mouse()

```

int cmd_hwload_mouse (
    usbdevice * kb,
    usbmode * dummy1,
    int dummy2,
    int apply,
    const char * dummy3 )

```

Definition at line 6 of file `profile_mouse.c`.

References `DELAY_LONG`, `hwprofile::dpi`, `usbdevice::hw`, `hwtonative()`, `hwprofile::id`, `hwprofile::light`, `loaddpi()`, `loadrgb_mouse()`, `MSG_SIZE`, `hwprofile::name`, `PR_NAME_LEN`, `usbdevice::profile`, and `usbrecv`.

```

6
7     DELAY_LONG(kb);
8     hwprofile* hw = calloc(1, sizeof(hwprofile));
9     // Ask for profile and mode IDs
10    uchar data_pkt[2][MSG_SIZE] = {
11        { 0x0e, 0x15, 0x01, 0 },
12        { 0x0e, 0x16, 0x01, 0 }
13    };
14    uchar in_pkt[MSG_SIZE];
15    for(int i = 0; i <= 1; i++){
16        data_pkt[0][3] = i;
17        if(!usbrecv(kb, data_pkt[0], in_pkt)){
18            free(hw);
19            return -1;
20        }
21        memcpy(hw->id + i, in_pkt + 4, sizeof(usbid));
22    }
23    // Ask for profile and mode names
24    for(int i = 0; i <= 1; i++){
25        data_pkt[1][3] = i;
26        if(!usbrecv(kb, data_pkt[1], in_pkt)){
27            free(hw);
28            return -1;
29        }
30        memcpy(hw->name[i], in_pkt + 4, PR_NAME_LEN * 2);
31    }
32
33    // Load the RGB and DPI settings
34    if(loadrgb_mouse(kb, hw->light, 0)
35        || loaddpi(kb, hw->dpi, hw->light)){
36        free(hw);
37        return -1;
38    }
39
40    // Make the profile active (if requested)
41    if(apply)

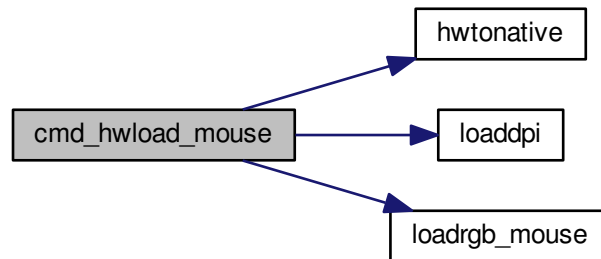
```

```

42     hwtonative(kb->profile, hw, 1);
43     // Free the existing profile (if any)
44     free(kb->hw);
45     kb->hw = hw;
46     DELAY_LONG(kb);
47     return 0;
48 }

```

Here is the call graph for this function:



5.33.2.6 cmd_hwsave_kb()

```

int cmd_hwsave_kb (
    usbdevice * kb,
    usbmode * dummy1,
    int dummy2,
    int dummy3,
    const char * dummy4 )

```

Definition at line 57 of file profile_keyboard.c.

References `DELAY_LONG`, `usbdevice::hw`, `HWMODE_K70`, `HWMODE_K95`, `hwprofile::id`, `IS_K95`, `hwprofile::light`, `MD_NAME_LEN`, `MSG_SIZE`, `hwprofile::name`, `nativetohw()`, `usbdevice::profile`, `savergb_kb()`, and `usb send`.

```

57     {
58     DELAY_LONG(kb);
59     hwprofile* hw = kb->hw;
60     if(!hw)
61         hw = kb->hw = calloc(1, sizeof(hwprofile));
62     int modes = (IS_K95(kb) ? HWMODE_K95 : HWMODE_K70);
63     nativetohw(kb->profile, hw, modes);
64     // Save the profile and mode names
65     uchar data_pkt[2][MSG_SIZE] = {
66         { 0x07, 0x16, 0x01, 0 },
67         { 0x07, 0x15, 0x01, 0 },
68     };
69     // Save the mode names
70     for(int i = 0; i <= modes; i++){
71         data_pkt[0][3] = i;
72         memcpy(data_pkt[0] + 4, hw->name[i], MD_NAME_LEN * 2);
73         if(!usb send(kb, data_pkt[0], 1))
74             return -1;
75     }
76     // Save the IDs

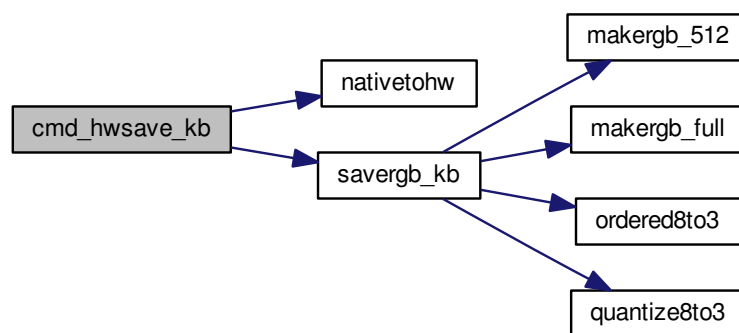
```

```

77     for(int i = 0; i <= modes; i++){
78         data_pkt[1][3] = i;
79         memcpy(data_pkt[1] + 4, hw->id + i, sizeof(usbid));
80         if(!usbdevice::usbdevice::send(kb, data_pkt[1], 1))
81             return -1;
82     }
83     // Save the RGB data
84     for(int i = 0; i < modes; i++){
85         if(savergb_kb(kb, hw->light + i, i))
86             return -1;
87     }
88     DELAY_LONG(kb);
89     return 0;
90 }

```

Here is the call graph for this function:



5.33.2.7 cmd_hwsave_mouse()

```

int cmd_hwsave_mouse (
    usbdevice * kb,
    usbmode * dummy1,
    int dummy2,
    int dummy3,
    const char * dummy4 )

```

Definition at line 50 of file `profile_mouse.c`.

References `DELAY_LONG`, `hwprofile::dpi`, `usbdevice::hw`, `hwprofile::id`, `hwprofile::light`, `MD_NAME_LEN`, `MSG_SIZE`, `hwprofile::name`, `nativetohw()`, `usbdevice::profile`, `savedpi()`, `savergb_mouse()`, and `usbdevice::send`.

```

50     {
51         DELAY_LONG(kb);
52         hwprofile* hw = kb->hw;
53         if(!hw)
54             hw = kb->hw = calloc(1, sizeof(hwprofile));
55         nativetohw(kb->profile, hw, 1);
56         // Save the profile and mode names
57         uchar data_pkt[2][MSG_SIZE] = {
58             { 0x07, 0x16, 0x01, 0 },
59             { 0x07, 0x15, 0x01, 0 },
60         };

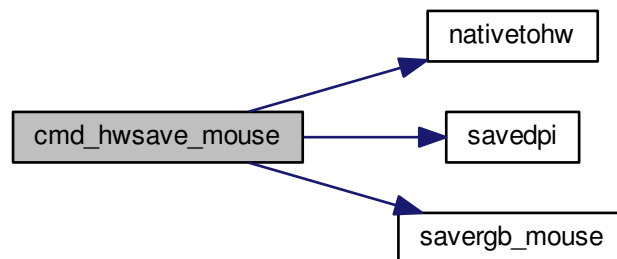
```

```

61     for(int i = 0; i <= 1; i++){
62         data_pkt[0][3] = i;
63         memcpy(data_pkt[0] + 4, hw->name[i], MD_NAME_LEN * 2);
64         if(!usb_send(kb, data_pkt[0], 1))
65             return -1;
66     }
67     // Save the IDs
68     for(int i = 0; i <= 1; i++){
69         data_pkt[1][3] = i;
70         memcpy(data_pkt[1] + 4, hw->id + i, sizeof(usb_id));
71         if(!usb_send(kb, data_pkt[1], 1))
72             return -1;
73     }
74     // Save the RGB data for the non-DPI zones
75     if(savergb_mouse(kb, hw->light, 0))
76         return -1;
77     // Save the DPI data (also saves RGB for those states)
78     if(savedpi(kb, hw->dpi, hw->light))
79         return -1;
80     DELAY_LONG(kb);
81     return 0;
82 }

```

Here is the call graph for this function:



5.33.2.8 cmd_id()

```

void cmd_id (
    usbdevice * kb,
    usbmode * mode,
    int dummy1,
    int dummy2,
    const char * id )

```

Definition at line 160 of file profile.c.

References `usbmode::id`, `usb_id::modified`, and `setid()`.

```

160
161     // ID is either a GUID or an 8-digit hex number
162     int newmodified;
163     if(!setid(&mode->id, id) && sscanf(id, "%08x", &newmodified) == 1)
164         memcpy(mode->id.modified, &newmodified, sizeof(newmodified));
165 }

```

Here is the call graph for this function:



5.33.2.9 cmd_name()

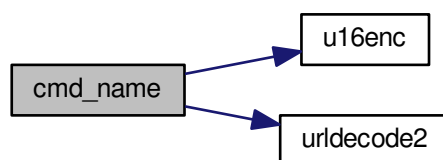
```
void cmd_name (  
    usbdevice * kb,  
    usbmode * mode,  
    int dummy1,  
    int dummy2,  
    const char * name )
```

Definition at line 117 of file profile.c.

References MD_NAME_LEN, usbmode::name, u16enc(), and urldecode2().

```
117                                     {  
118     char decoded[strlen(name) + 1];  
119     urldecode2(decoded, name);  
120     size_t srclen = strlen(decoded), dstlen = MD_NAME_LEN;  
121     u16enc(decoded, mode->name, &srclen, &dstlen);  
122 }
```

Here is the call graph for this function:



5.33.2.10 cmd_profileid()

```
void cmd_profileid (
    usbdevice * kb,
    usbmode * mode,
    int dummy1,
    int dummy2,
    const char * id )
```

Definition at line 167 of file profile.c.

References `usbprofile::id`, `usbid::modified`, `usbdevice::profile`, and `setid()`.

```
167                                     {
168     usbprofile* profile = kb->profile;
169     int newmodified;
170     if(!setid(&profile->id, id) && sscanf(id, "%08x", &newmodified) == 1)
171         memcpy(profile->id.modified, &newmodified, sizeof(newmodified));
172
173 }
```

Here is the call graph for this function:



5.33.2.11 cmd_profilename()

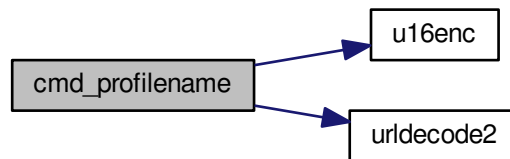
```
void cmd_profilename (
    usbdevice * kb,
    usbmode * dummy1,
    int dummy2,
    int dummy3,
    const char * name )
```

Definition at line 124 of file profile.c.

References `usbprofile::name`, `PR_NAME_LEN`, `usbdevice::profile`, `u16enc()`, and `urldecode2()`.

```
124                                     {
125     usbprofile* profile = kb->profile;
126     char decoded[strlen(name) + 1];
127     urldecode2(decoded, name);
128     size_t srclen = strlen(decoded), dstlen = PR_NAME_LEN;
129     u16enc(decoded, profile->name, &srclen, &dstlen);
130 }
```

Here is the call graph for this function:



5.33.2.12 freeprofile()

```
void freeprofile (  
    usbdevice * kb )
```

Definition at line 228 of file profile.c.

References `_freeprofile()`, and `usbdevice::hw`.

```
228                                     {  
229     _freeprofile(kb);  
230     // Also free HW profile  
231     free(kb->hw);  
232     kb->hw = 0;  
233 }
```

Here is the call graph for this function:



5.33.2.13 gethwmodename()

```
char* gethwmodename (  
    hwprofile * profile,  
    int index )
```

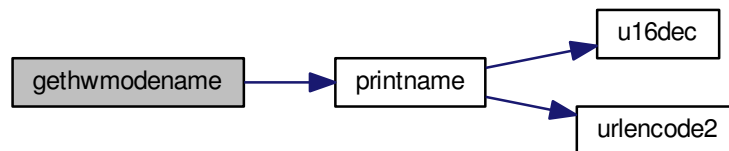
Definition at line 152 of file profile.c.

References MD_NAME_LEN, hwprofile::name, and printname().

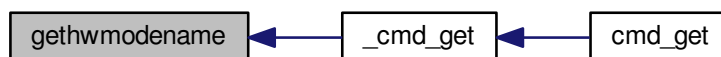
Referenced by _cmd_get().

```
152     {  
153     return printname(profile->name[index + 1], MD_NAME_LEN);  
154 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



5.33.2.14 gethwprofilename()

```
char* gethwprofilename (  
    hwprofile * profile )
```

Definition at line 156 of file profile.c.

References MD_NAME_LEN, hwprofile::name, and printname().

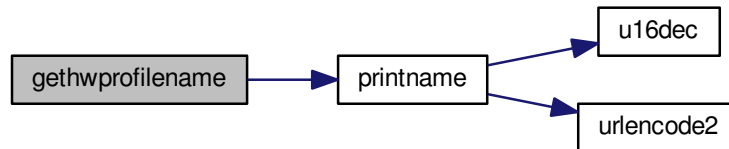
Referenced by _cmd_get().

```

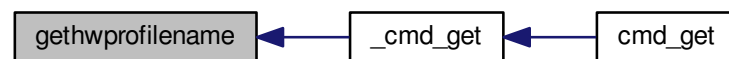
156     {
157     return printname(profile->name[0], MD_NAME_LEN);
158 }

```

Here is the call graph for this function:



Here is the caller graph for this function:



5.33.2.15 getid()

```

char* getid (
    usbid * id )

```

Definition at line 79 of file profile.c.

References `usbid::guid`.

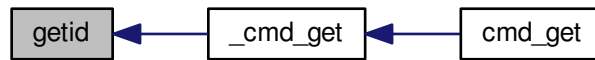
Referenced by `_cmd_get()`.

```

79     {
80     int32_t data1;
81     int16_t data2, data3, data4a;
82     char data4b[6];
83     memcpy(&data1, id->guid + 0x0, 4);
84     memcpy(&data2, id->guid + 0x4, 2);
85     memcpy(&data3, id->guid + 0x6, 2);
86     memcpy(&data4a, id->guid + 0x8, 2);
87     memcpy(data4b, id->guid + 0xA, 6);
88     char* guid = malloc(39);
89     snprintf(guid, 39, "{%08X-%04hX-%04hX-%04hX-%02hhX%02hhX%02hhX%02hhX%02hhX%02hhX}",
90             data1, data2, data3, data4a, data4b[0], data4b[1], data4b[2], data4b[3], data4b[4], data4b[5])
91     ;
92     return guid;
93 }

```

Here is the caller graph for this function:



5.33.2.16 getmodename()

```
char* getmodename (
    usbmode * mode )
```

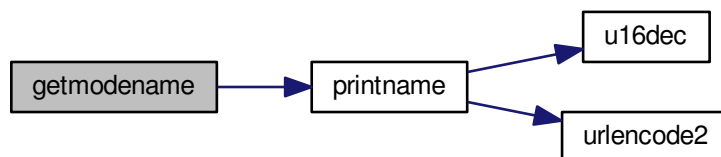
Definition at line 144 of file profile.c.

References MD_NAME_LEN, usbmode::name, and printname().

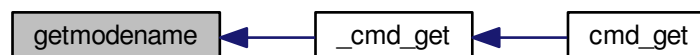
Referenced by `_cmd_get()`.

```
144     {
145     return printname(mode->name, MD_NAME_LEN);
146 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



5.33.2.17 getprofilename()

```
char* getprofilename (
    usbprofile * profile )
```

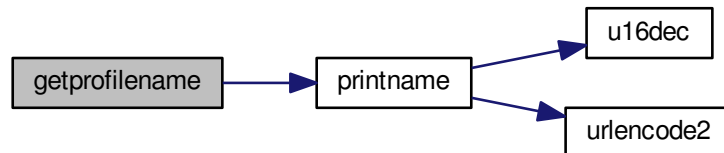
Definition at line 148 of file profile.c.

References usbprofile::name, PR_NAME_LEN, and printname().

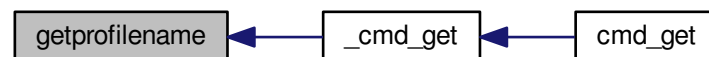
Referenced by _cmd_get().

```
148
149     return printname(profile->name, PR_NAME_LEN);
150 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



5.33.2.18 hwtonative()

```
void hwtonative (
    usbprofile * profile,
    hwprofile * hw,
    int modecount )
```

Definition at line 235 of file profile.c.

References usbmode::dpi, hwprofile::dpi, dpiset::forceupdate, lighting::forceupdate, usbmode::id, usbprofile::id, hwprofile::id, usbprofile::lastdpi, usbprofile::lastlight, usbmode::light, hwprofile::light, MD_NAME_LEN, usbprofile::mode, usbmode::name, usbprofile::name, hwprofile::name, and PR_NAME_LEN.

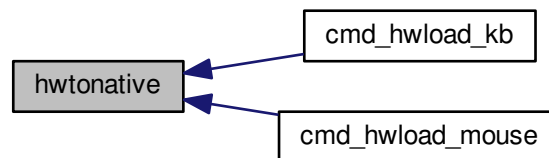
Referenced by `cmd_hwload_kb()`, and `cmd_hwload_mouse()`.

```

235                                     {
236     // Copy the profile name and ID
237     memcpy(profile->name, hw->name[0], PR_NAME_LEN * 2);
238     memcpy(&profile->id, hw->id, sizeof(usbid));
239     // Copy the mode settings
240     for(int i = 0; i < modecount; i++){
241         usbmode* mode = profile->mode + i;
242         memcpy(mode->name, hw->name[i + 1], MD_NAME_LEN * 2);
243         memcpy(&mode->id, hw->id + i + 1, sizeof(usbid));
244         memcpy(&mode->light, hw->light + i, sizeof(lighting));
245         memcpy(&mode->dpi, hw->dpi + i, sizeof(dpiset));
246         // Set a force update on the light/DPI since they've been overwritten
247         mode->light.forceupdate = mode->dpi.forceupdate = 1;
248     }
249     profile->lastlight.forceupdate = profile->lastdpi.
forceupdate = 1;
250 }

```

Here is the caller graph for this function:



5.33.2.19 loadprofile()

```

int loadprofile (
    usbdevice * kb )

```

Definition at line 192 of file profile.c.

References `hwloadprofile`.

```

192                                     {
193     if(hwloadprofile(kb, 1))
194         return -1;
195     return 0;
196 }

```

5.33.2.20 nativetohw()

```
void nativetohw (
    usbprofile * profile,
    hwprofile * hw,
    int modecount )
```

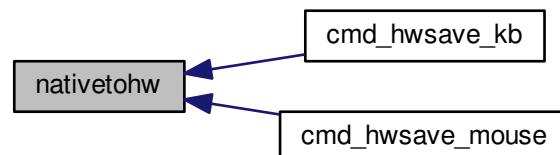
Definition at line 252 of file profile.c.

References `usbmode::dpi`, `hwprofile::dpi`, `usbmode::id`, `usbprofile::id`, `hwprofile::id`, `usbmode::light`, `hwprofile::light`, `MD_NAME_LEN`, `usbprofile::mode`, `usbmode::name`, `usbprofile::name`, `hwprofile::name`, and `PR_NAME_LEN`.

Referenced by `cmd_hwsave_kb()`, and `cmd_hwsave_mouse()`.

```
252                                     {
253     // Copy name and ID
254     memcpy(hw->name[0], profile->name, PR_NAME_LEN * 2);
255     memcpy(hw->id, &profile->id, sizeof(usbid));
256     // Copy the mode settings
257     for(int i = 0; i < modecount; i++){
258         usbmode* mode = profile->mode + i;
259         memcpy(hw->name[i + 1], mode->name, MD_NAME_LEN * 2);
260         memcpy(hw->id + i + 1, &mode->id, sizeof(usbid));
261         memcpy(hw->light + i, &mode->light, sizeof(lighting));
262         memcpy(hw->dpi + i, &mode->dpi, sizeof(dpiset));
263     }
264 }
```

Here is the caller graph for this function:



5.33.2.21 setid()

```
int setid (
    usbid * id,
    const char * guid )
```

Definition at line 64 of file profile.c.

References `usbid::guid`.

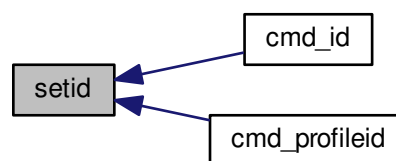
Referenced by `cmd_id()`, and `cmd_profileid()`.

```

64                                     {
65     int32_t data1;
66     int16_t data2, data3, data4a;
67     char data4b[6];
68     if(sscanf(guid, "%08X-%04hX-%04hX-%04hX-%02hhX%02hhX%02hhX%02hhX%02hhX%02hhX",
69         &data1, &data2, &data3, &data4a, data4b, data4b + 1, data4b + 2, data4b + 3, data4b + 4,
data4b + 5) != 10)
70         return 0;
71     memcpy(id->guid + 0x0, &data1, 4);
72     memcpy(id->guid + 0x4, &data2, 2);
73     memcpy(id->guid + 0x6, &data3, 2);
74     memcpy(id->guid + 0x8, &data4a, 2);
75     memcpy(id->guid + 0xA, data4b, 6);
76     return 1;
77 }

```

Here is the caller graph for this function:



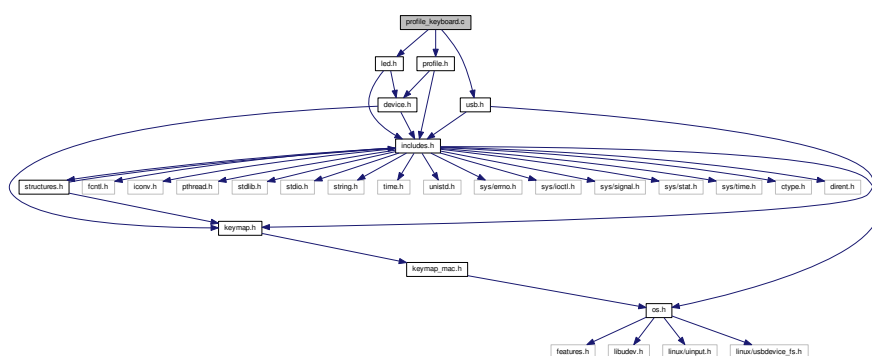
5.34 profile_keyboard.c File Reference

```

#include "profile.h"
#include "usb.h"
#include "led.h"

```

Include dependency graph for profile_keyboard.c:



Functions

- static int [hwloadmode](#) ([usbdevice](#) *kb, [hwprofile](#) *hw, int mode)
- int [cmd_hwload_kb](#) ([usbdevice](#) *kb, [usbmode](#) *dummy1, int dummy2, int apply, const char *dummy3)
- int [cmd_hwsave_kb](#) ([usbdevice](#) *kb, [usbmode](#) *dummy1, int dummy2, int dummy3, const char *dummy4)

5.34.1 Function Documentation

5.34.1.1 cmd_hwload_kb()

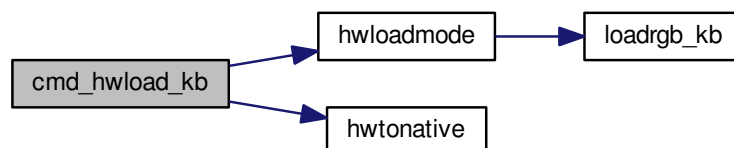
```
int cmd_hwload_kb (
    usbdevice * kb,
    usbmode * dummy1,
    int dummy2,
    int apply,
    const char * dummy3 )
```

Definition at line 16 of file profile_keyboard.c.

References `DELAY_LONG`, `usbdevice::hw`, `hwloadmode()`, `HWMODE_K70`, `HWMODE_K95`, `hwtonative()`, `hwprofile::id`, `IS_K95`, `MSG_SIZE`, `hwprofile::name`, `PR_NAME_LEN`, `usbdevice::profile`, and `usbrecv`.

```
16 {
17     DELAY_LONG(kb);
18     hwprofile* hw = calloc(1, sizeof(hwprofile));
19     // Ask for profile and mode IDs
20     uchar data_pkt[2][MSG_SIZE] = {
21         { 0x0e, 0x15, 0x01, 0 },
22         { 0x0e, 0x16, 0x01, 0 }
23     };
24     uchar in_pkt[MSG_SIZE];
25     int modes = (IS_K95(kb) ? HWMODE_K95 : HWMODE_K70);
26     for(int i = 0; i <= modes; i++){
27         data_pkt[0][3] = i;
28         if(!usbrecv(kb, data_pkt[0], in_pkt)){
29             free(hw);
30             return -1;
31         }
32         memcpy(hw->id + i, in_pkt + 4, sizeof(usbid));
33     }
34     // Ask for profile name
35     if(!usbrecv(kb, data_pkt[1], in_pkt)){
36         free(hw);
37         return -1;
38     }
39     memcpy(hw->name[0], in_pkt + 4, PR_NAME_LEN * 2);
40     // Load modes
41     for(int i = 0; i < modes; i++){
42         if(hwloadmode(kb, hw, i)){
43             free(hw);
44             return -1;
45         }
46     }
47     // Make the profile active (if requested)
48     if(apply)
49         hwtonative(kb->profile, hw, modes);
50     // Free the existing profile (if any)
51     free(kb->hw);
52     kb->hw = hw;
53     DELAY_LONG(kb);
54     return 0;
55 }
```

Here is the call graph for this function:



5.34.1.2 cmd_hwsave_kb()

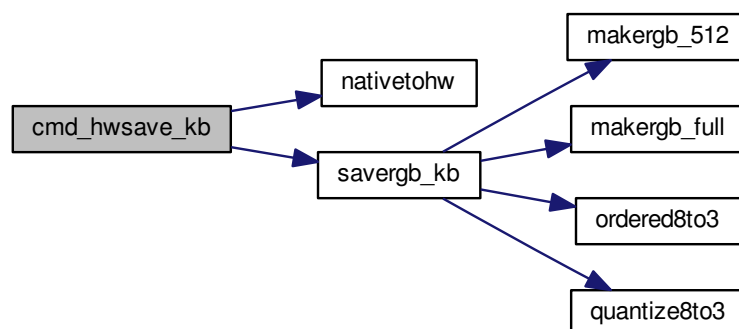
```
int cmd_hwsave_kb (
    usbdevice * kb,
    usbmode * dummy1,
    int dummy2,
    int dummy3,
    const char * dummy4 )
```

Definition at line 57 of file profile_keyboard.c.

References `DELAY_LONG`, `usbdevice::hw`, `HWMODE_K70`, `HWMODE_K95`, `hwprofile::id`, `IS_K95`, `hwprofile::light`, `MD_NAME_LEN`, `MSG_SIZE`, `hwprofile::name`, `nativetohw()`, `usbdevice::profile`, `savergb_kb()`, and `usb send`.

```
57                                     {
58     DELAY_LONG(kb);
59     hwprofile* hw = kb->hw;
60     if(!hw)
61         hw = kb->hw = calloc(1, sizeof(hwprofile));
62     int modes = (IS_K95(kb) ? HWMODE_K95 : HWMODE_K70);
63     nativetohw(kb->profile, hw, modes);
64     // Save the profile and mode names
65     uchar data_pkt[2][MSG_SIZE] = {
66         { 0x07, 0x16, 0x01, 0 },
67         { 0x07, 0x15, 0x01, 0 },
68     };
69     // Save the mode names
70     for(int i = 0; i <= modes; i++){
71         data_pkt[0][3] = i;
72         memcpy(data_pkt[0] + 4, hw->name[i], MD_NAME_LEN * 2);
73         if(!usb send(kb, data_pkt[0], 1))
74             return -1;
75     }
76     // Save the IDs
77     for(int i = 0; i <= modes; i++){
78         data_pkt[1][3] = i;
79         memcpy(data_pkt[1] + 4, hw->id + i, sizeof(usb id));
80         if(!usb send(kb, data_pkt[1], 1))
81             return -1;
82     }
83     // Save the RGB data
84     for(int i = 0; i < modes; i++){
85         if(savergb_kb(kb, hw->light + i, i))
86             return -1;
87     }
88     DELAY_LONG(kb);
89     return 0;
90 }
```

Here is the call graph for this function:



5.34.1.3 hwloadmode()

```
static int hwloadmode (  
    usbdevice * kb,  
    hwprofile * hw,  
    int mode ) [static]
```

Definition at line 5 of file profile_keyboard.c.

References hwprofile::light, loadrgb_kb(), MD_NAME_LEN, MSG_SIZE, hwprofile::name, and usbrecv.

Referenced by cmd_hwload_kb().

```
5                                     {  
6     // Ask for mode's name  
7     uchar data_pkt[MSG_SIZE] = { 0x0e, 0x16, 0x01, mode + 1, 0 };  
8     uchar in_pkt[MSG_SIZE];  
9     if(!usbrecv(kb, data_pkt, in_pkt))  
10        return -1;  
11     memcpy(hw->name[mode + 1], in_pkt + 4, MD_NAME_LEN * 2);  
12     // Load the RGB setting  
13     return loadrgb_kb(kb, hw->light + mode, mode);  
14 }
```

Here is the call graph for this function:



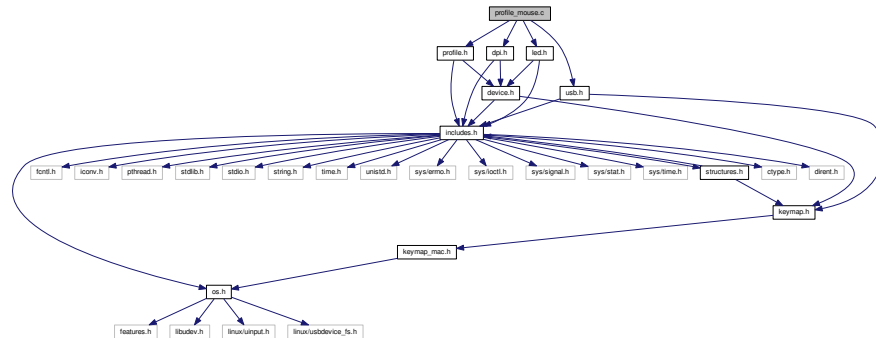
Here is the caller graph for this function:



5.35 profile_mouse.c File Reference

```
#include "dpi.h"
#include "profile.h"
#include "usb.h"
#include "led.h"
```

Include dependency graph for profile_mouse.c:



Functions

- `int cmd_hwload_mouse (usbdevice *kb, usbmode *dummy1, int dummy2, int apply, const char *dummy3)`
- `int cmd_hwsave_mouse (usbdevice *kb, usbmode *dummy1, int dummy2, int dummy3, const char *dummy4)`

5.35.1 Function Documentation

5.35.1.1 cmd_hwload_mouse()

```
int cmd_hwload_mouse (
    usbdevice * kb,
    usbmode * dummy1,
    int dummy2,
    int apply,
    const char * dummy3 )
```

Definition at line 6 of file profile_mouse.c.

References `DELAY_LONG`, `hwprofile::dpi`, `usbdevice::hw`, `hwtonative()`, `hwprofile::id`, `hwprofile::light`, `loaddpi()`, `loadrgb_mouse()`, `MSG_SIZE`, `hwprofile::name`, `PR_NAME_LEN`, `usbdevice::profile`, and `usbrecv`.

```

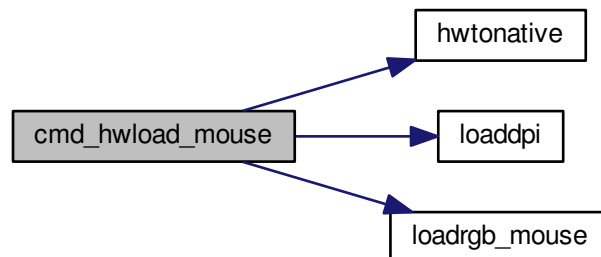
6
7     DELAY_LONG(kb);
8     hwprofile* hw = calloc(1, sizeof(hwprofile));
9     // Ask for profile and mode IDs
10    uchar data_pkt[2][MSG_SIZE] = {
11        { 0x0e, 0x15, 0x01, 0 },
12        { 0x0e, 0x16, 0x01, 0 }
13    };
14    uchar in_pkt[MSG_SIZE];
15    for(int i = 0; i <= 1; i++){
```

```

16     data_pkt[0][3] = i;
17     if(!usbrecv(kb, data_pkt[0], in_pkt)){
18         free(hw);
19         return -1;
20     }
21     memcpy(hw->id + i, in_pkt + 4, sizeof(usbid));
22 }
23 // Ask for profile and mode names
24 for(int i = 0; i <= 1; i++){
25     data_pkt[1][3] = i;
26     if(!usbrecv(kb, data_pkt[1], in_pkt)){
27         free(hw);
28         return -1;
29     }
30     memcpy(hw->name[i], in_pkt + 4, PR_NAME_LEN * 2);
31 }
32
33 // Load the RGB and DPI settings
34 if(loadrgb_mouse(kb, hw->light, 0)
35    || loaddpi(kb, hw->dpi, hw->light)){
36     free(hw);
37     return -1;
38 }
39
40 // Make the profile active (if requested)
41 if(apply)
42     hwtonative(kb->profile, hw, 1);
43 // Free the existing profile (if any)
44 free(kb->hw);
45 kb->hw = hw;
46 DELAY_LONG(kb);
47 return 0;
48 }

```

Here is the call graph for this function:



5.35.1.2 cmd_hwsave_mouse()

```

int cmd_hwsave_mouse (
    usbdevice * kb,
    usbmode * dummy1,
    int dummy2,
    int dummy3,
    const char * dummy4 )

```

Definition at line 50 of file `profile_mouse.c`.

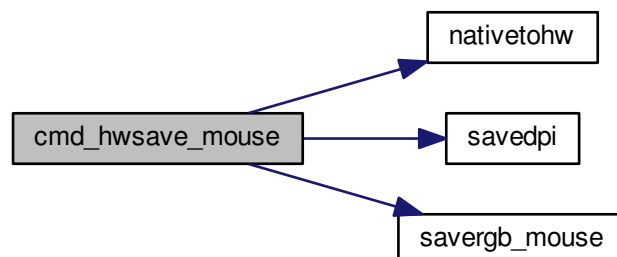
References `DELAY_LONG`, `hwprofile::dpi`, `usbdevice::hw`, `hwprofile::id`, `hwprofile::light`, `MD_NAME_LEN`, `MSG_SIZE`, `hwprofile::name`, `nativetohw()`, `usbdevice::profile`, `savedpi()`, `savergb_mouse()`, and `usbsend`.

```

50                                     {
51     DELAY_LONG(kb);
52     hwprofile* hw = kb->hw;
53     if(!hw)
54         hw = kb->hw = calloc(1, sizeof(hwprofile));
55     nativetohw(kb->profile, hw, 1);
56     // Save the profile and mode names
57     uchar data_pkt[2][MSG_SIZE] = {
58         { 0x07, 0x16, 0x01, 0 },
59         { 0x07, 0x15, 0x01, 0 },
60     };
61     for(int i = 0; i <= 1; i++){
62         data_pkt[0][3] = i;
63         memcpy(data_pkt[0] + 4, hw->name[i], MD_NAME_LEN * 2);
64         if(!usb_send(kb, data_pkt[0], 1))
65             return -1;
66     }
67     // Save the IDs
68     for(int i = 0; i <= 1; i++){
69         data_pkt[1][3] = i;
70         memcpy(data_pkt[1] + 4, hw->id + i, sizeof(usb_id));
71         if(!usb_send(kb, data_pkt[1], 1))
72             return -1;
73     }
74     // Save the RGB data for the non-DPI zones
75     if(savergb_mouse(kb, hw->light, 0))
76         return -1;
77     // Save the DPI data (also saves RGB for those states)
78     if(savedpi(kb, hw->dpi, hw->light))
79         return -1;
80     DELAY_LONG(kb);
81     return 0;
82 }

```

Here is the call graph for this function:

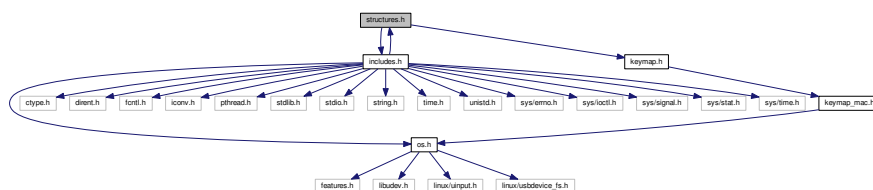


5.36 structures.h File Reference

```
#include "includes.h"
```

```
#include "keymap.h"
```

Include dependency graph for `structures.h`:



- `#define FEAT_COMMON (FEAT_BIND | FEAT_NOTIFY | FEAT_FWVERSION | FEAT_MOUSEACCEL | FEAT_HWLOAD)`
- `#define FEAT_STD_RGB (FEAT_COMMON | FEAT_RGB | FEAT_POLLRATE | FEAT_FWUPDATE)`
- `#define FEAT_STD_NRGB (FEAT_COMMON)`
- `#define FEAT_LMASK (FEAT_ANSI | FEAT_ISO)`
- `#define HAS_FEATURES(kb, feat) (((kb)->features & (feat)) == (feat))`
- `#define HAS_ANY_FEATURE(kb, feat) (!((kb)->features & (feat)))`
- `#define NEEDS_FW_UPDATE(kb) ((kb)->fwversion == 0 && HAS_FEATURES((kb), FEAT_FWUPDATE | FEAT_FWVERSION))`
- `#define SCROLL_ACCELERATED 0`
- `#define SCROLL_MIN 1`
- `#define SCROLL_MAX 10`
- `#define KB_NAME_LEN 34`
- `#define SERIAL_LEN 34`
- `#define MSG_SIZE 64`
- `#define IFACE_MAX 4`

Variables

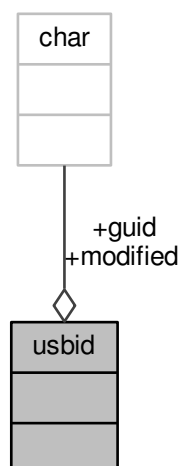
- const union `devcmd vtable_keyboard`
- const union `devcmd vtable_keyboard_nonrgb`
- const union `devcmd vtable_mouse`

5.36.1 Data Structure Documentation

5.36.1.1 struct usbid

Definition at line 8 of file structures.h.

Collaboration diagram for usbid:



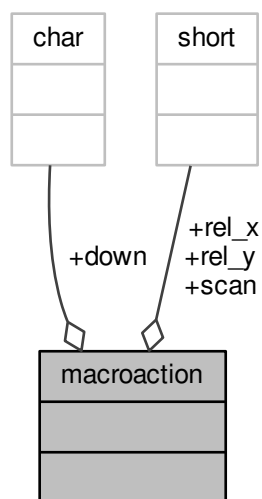
Data Fields

char	guid[16]	
char	modified[4]	

5.36.1.2 struct macroaction

Definition at line 27 of file structures.h.

Collaboration diagram for macroaction:



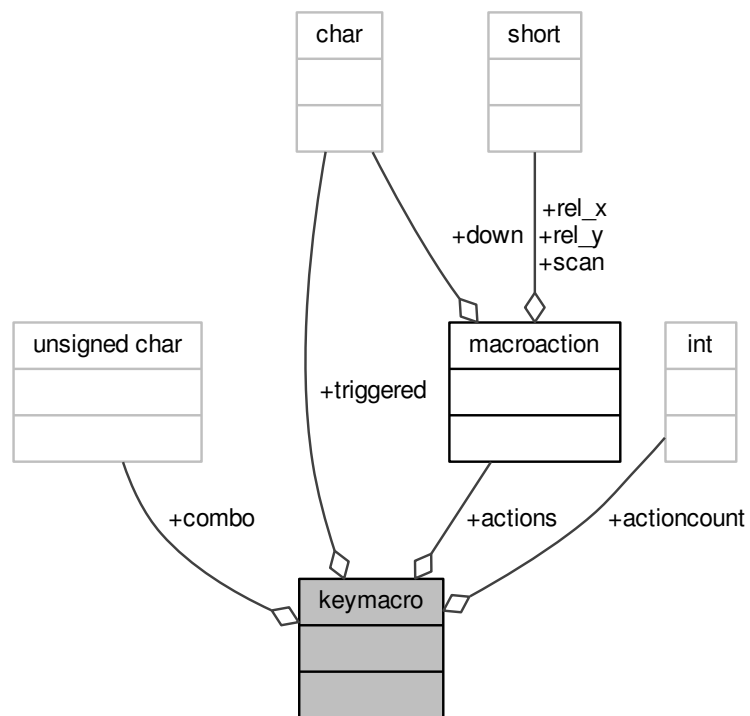
Data Fields

char	down	
short	rel_x	
short	rel_y	
short	scan	

5.36.1.3 struct keymacro

Definition at line 34 of file structures.h.

Collaboration diagram for keymacro:



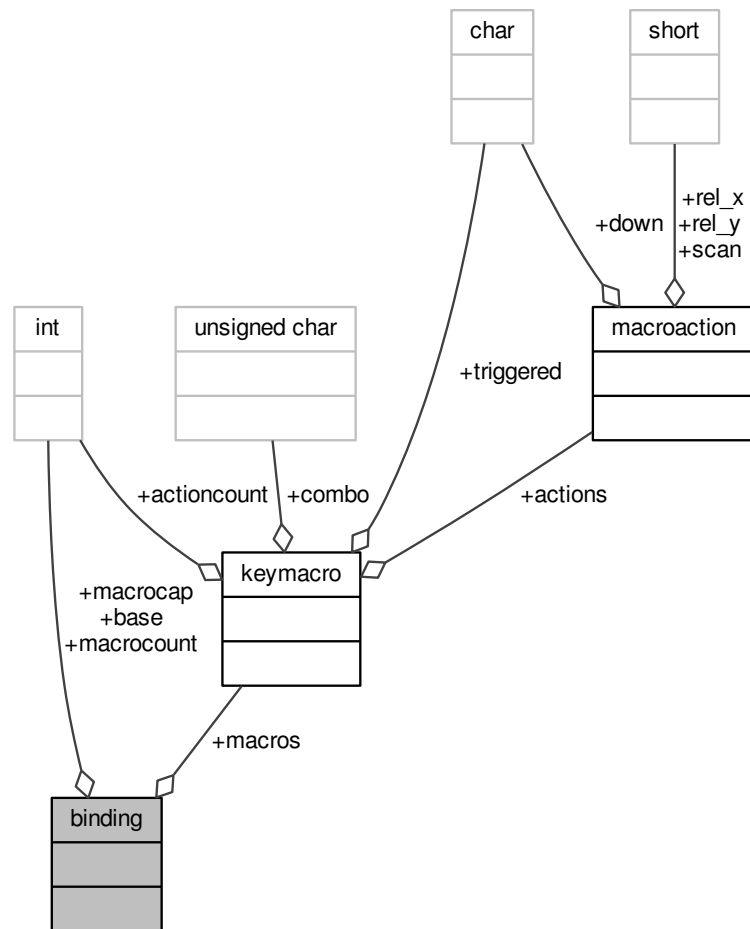
Data Fields

int	actioncount	
macroaction *	actions	
uchar	combo[(((152+3+12)+25)+7)/8]]	
char	triggered	

5.36.1.4 struct binding

Definition at line 42 of file structures.h.

Collaboration diagram for binding:



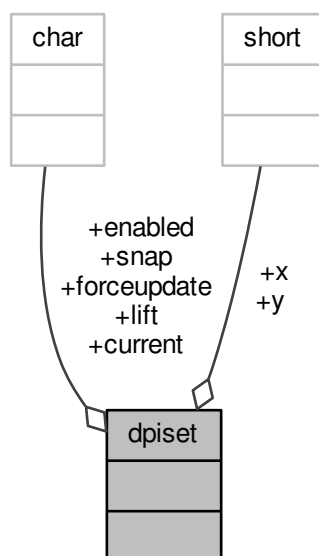
Data Fields

int	base[(((152+3+12)+25))]	
int	macrocap	
int	macrocount	
keymacro *	macros	

5.36.1.5 struct dpiset

Definition at line 56 of file structures.h.

Collaboration diagram for dpiset:



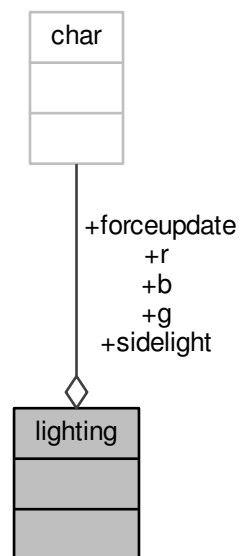
Data Fields

uchar	current	
uchar	enabled	
uchar	forceupdate	
uchar	lift	
uchar	snap	
ushort	x[6]	
ushort	y[6]	

5.36.1.6 struct lighting

Definition at line 72 of file structures.h.

Collaboration diagram for lighting:



Data Fields

<code>uchar</code>	<code>b[152+11]</code>	
<code>uchar</code>	<code>forceupdate</code>	
<code>uchar</code>	<code>g[152+11]</code>	
<code>uchar</code>	<code>r[152+11]</code>	
<code>uchar</code>	<code>sidelight</code>	

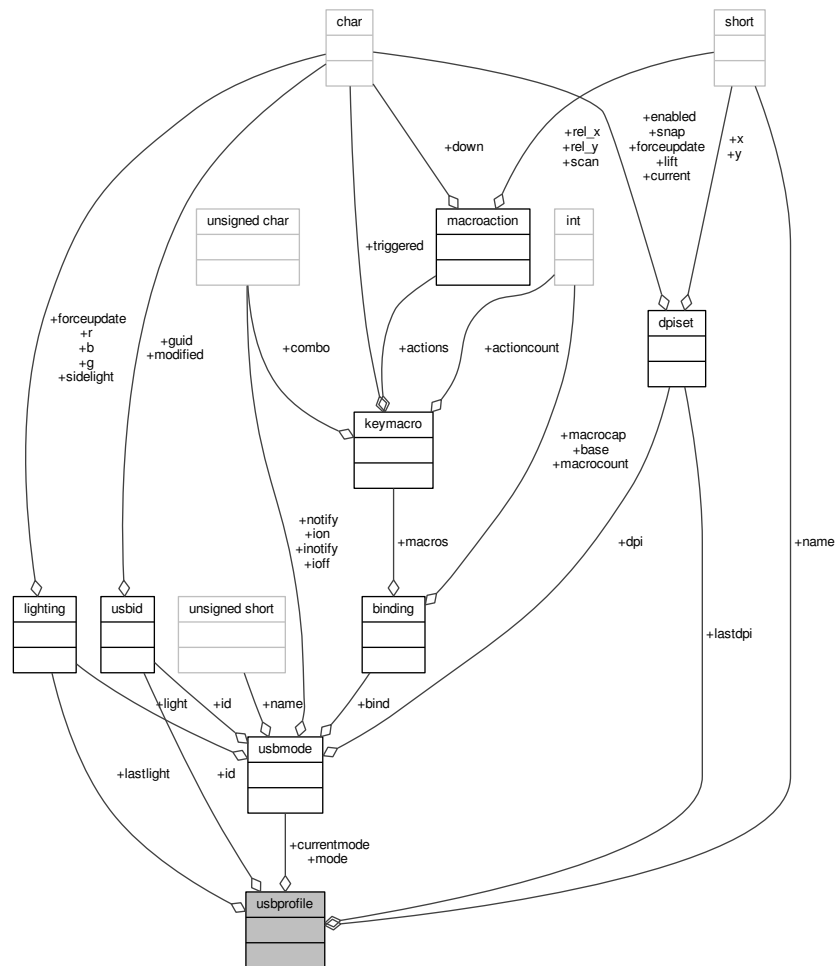
5.36.1.7 struct usbmode

Definition at line 82 of file structures.h.

5.36.1.8 struct usbprofile

Definition at line 100 of file structures.h.

Collaboration diagram for usbprofile:



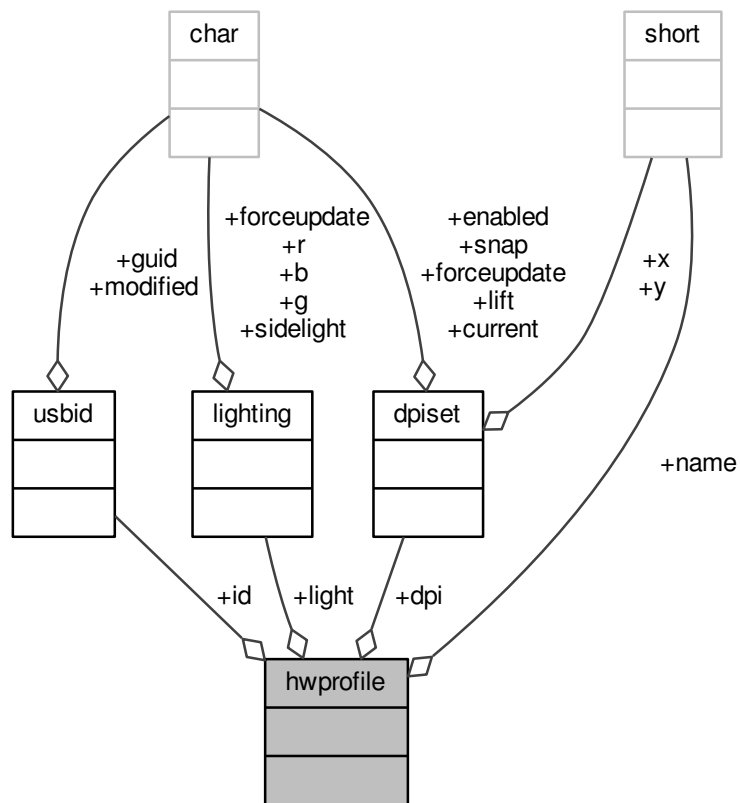
Data Fields

<code>usbmode *</code>	<code>currentmode</code>	
<code>usbid</code>	<code>id</code>	
<code>dpiset</code>	<code>lastdpi</code>	
<code>lighting</code>	<code>lastlight</code>	
<code>usbmode</code>	<code>mode[6]</code>	
<code>ushort</code>	<code>name[16]</code>	

5.36.1.9 struct hwprofile

Definition at line 117 of file structures.h.

Collaboration diagram for hwprofile:



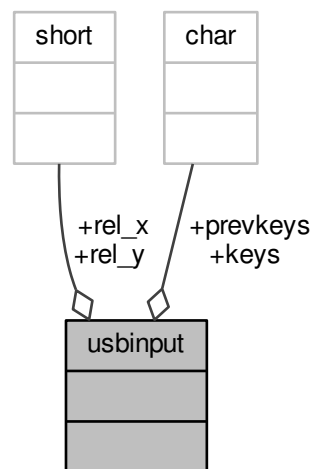
Data Fields

dpi	<code>dpi[3]</code>	
usb	<code>id[3+1]</code>	
lighting	<code>light[3]</code>	
ushort	<code>name[3+1][16]</code>	

5.36.1.10 struct usbinput

Definition at line 128 of file structures.h.

Collaboration diagram for usbinput:



Data Fields

uchar	keys[(((152+3+12)+25)+7)/8]]	
uchar	prevkeys[(((152+3+12)+25)+7)/8]]	
short	rel_x	
short	rel_y	

5.36.1.11 struct usbdevice

Definition at line 177 of file structures.h.

Data Fields

char	serial[34]	
pthread_t	thread	
struct udev_device *	udev	
int	uinput_kb	
int	uinput_mouse	
char	usbdelay	
short	vendor	
const union devcmd *	vtable	

5.36.2 Macro Definition Documentation

5.36.2.1 CLEAR_KEYBIT

```
#define CLEAR_KEYBIT(  
    array,  
    index ) do { (array)[(index) / 8] &= ~(1 << ((index) % 8)); } while(0)
```

Definition at line 16 of file structures.h.

Referenced by `cmd_notify()`, `corsair_mousecopy()`, `hid_kb_translate()`, and `hid_mouse_translate()`.

5.36.2.2 DPI_COUNT

```
#define DPI_COUNT 6
```

Definition at line 53 of file structures.h.

Referenced by `cmd_dpi()`, `cmd_dpisel()`, `loaddpi()`, `printdpi()`, `savedpi()`, and `updatedpi()`.

5.36.2.3 FEAT_ADJRATE

```
#define FEAT_ADJRATE 0x008
```

Definition at line 138 of file structures.h.

Referenced by `_mkdevpath()`, `_setupusb()`, and `_start_dev()`.

5.36.2.4 FEAT_ANSI

```
#define FEAT_ANSI 0x200
```

Definition at line 145 of file structures.h.

Referenced by readcmd().

5.36.2.5 FEAT_BIND

```
#define FEAT_BIND 0x010
```

Definition at line 139 of file structures.h.

Referenced by _mkdevpath(), main(), and readcmd().

5.36.2.6 FEAT_COMMON

```
#define FEAT_COMMON (FEAT_BIND | FEAT_NOTIFY | FEAT_FWVERSION | FEAT_MOUSEACCEL | FEAT_HWLOAD)
```

Definition at line 150 of file structures.h.

5.36.2.7 FEAT_FWUPDATE

```
#define FEAT_FWUPDATE 0x080
```

Definition at line 142 of file structures.h.

Referenced by _mkdevpath(), _start_dev(), and cmd_fwupdate().

5.36.2.8 FEAT_FWVERSION

```
#define FEAT_FWVERSION 0x040
```

Definition at line 141 of file structures.h.

Referenced by _mkdevpath(), and _start_dev().

5.36.2.9 FEAT_HWLOAD

```
#define FEAT_HWLOAD 0x100
```

Definition at line 143 of file structures.h.

Referenced by `_start_dev()`.

5.36.2.10 FEAT_ISO

```
#define FEAT_ISO 0x400
```

Definition at line 146 of file structures.h.

Referenced by `readcmd()`.

5.36.2.11 FEAT_LMASK

```
#define FEAT_LMASK (FEAT_ANSI | FEAT_ISO)
```

Definition at line 153 of file structures.h.

Referenced by `readcmd()`.

5.36.2.12 FEAT_MONOCHROME

```
#define FEAT_MONOCHROME 0x002
```

Definition at line 136 of file structures.h.

Referenced by `_mkdevpath()`, and `_setupusb()`.

5.36.2.13 FEAT_MOUSEACCEL

```
#define FEAT_MOUSEACCEL 0x800
```

Definition at line 147 of file structures.h.

Referenced by `main()`, and `readcmd()`.

5.36.2.14 FEAT_NOTIFY

```
#define FEAT_NOTIFY 0x020
```

Definition at line 140 of file structures.h.

Referenced by `_mkdevpath()`, `main()`, and `readcmd()`.

5.36.2.15 FEAT_POLLRATE

```
#define FEAT_POLLRATE 0x004
```

Definition at line 137 of file structures.h.

Referenced by `_mkdevpath()`, `_start_dev()`, and `getfwversion()`.

5.36.2.16 FEAT_RGB

```
#define FEAT_RGB 0x001
```

Definition at line 135 of file structures.h.

Referenced by `_mkdevpath()`, `_start_dev()`, `os_setupusb()`, `revertusb()`, and `usbunclaim()`.

5.36.2.17 FEAT_STD_NRGB

```
#define FEAT_STD_NRGB (FEAT\_COMMON)
```

Definition at line 152 of file structures.h.

Referenced by `_setupusb()`.

5.36.2.18 FEAT_STD_RGB

```
#define FEAT_STD_RGB (FEAT\_COMMON | FEAT\_RGB | FEAT\_POLLRATE | FEAT\_FWUPDATE)
```

Definition at line 151 of file structures.h.

Referenced by `_setupusb()`.

5.36.2.19 HAS_ANY_FEATURE

```
#define HAS_ANY_FEATURE(  
    kb,  
    feat ) (!((kb)->features & (feat)))
```

Definition at line 157 of file structures.h.

5.36.2.20 HAS_FEATURES

```
#define HAS_FEATURES(  
    kb,  
    feat ) (((kb)->features & (feat)) == (feat))
```

Definition at line 156 of file structures.h.

Referenced by `_mkdevpath()`, `_start_dev()`, `cmd_fwupdate()`, `os_setupusb()`, `readcmd()`, `revertusb()`, and `usbunclaim()`.

5.36.2.21 HWMODE_K70

```
#define HWMODE_K70 1
```

Definition at line 114 of file structures.h.

Referenced by `cmd_hwload_kb()`, and `cmd_hwsave_kb()`.

5.36.2.22 HWMODE_K95

```
#define HWMODE_K95 3
```

Definition at line 115 of file structures.h.

Referenced by `cmd_hwload_kb()`, and `cmd_hwsave_kb()`.

5.36.2.23 HWMODE_MAX

```
#define HWMODE_MAX 3
```

Definition at line 116 of file structures.h.

5.36.2.24 I_CAPS

```
#define I_CAPS 2
```

Definition at line 20 of file structures.h.

Referenced by `_cmd_get()`, `iselect()`, `nprintind()`, and `updateindicators_kb()`.

5.36.2.25 I_NUM

```
#define I_NUM 1
```

Definition at line 19 of file structures.h.

Referenced by `_cmd_get()`, `iselect()`, `nprintind()`, and `updateindicators_kb()`.

5.36.2.26 I_SCROLL

```
#define I_SCROLL 4
```

Definition at line 21 of file structures.h.

Referenced by `_cmd_get()`, `iselect()`, `nprintind()`, and `updateindicators_kb()`.

5.36.2.27 IFACE_MAX

```
#define IFACE_MAX 4
```

Definition at line 176 of file structures.h.

5.36.2.28 KB_NAME_LEN

```
#define KB_NAME_LEN 34
```

Definition at line 173 of file structures.h.

Referenced by `_setupusb()`, and `os_setupusb()`.

5.36.2.29 LIFT_MAX

```
#define LIFT_MAX 5
```

Definition at line 55 of file structures.h.

Referenced by `cmd_lift()`, and `loaddpi()`.

5.36.2.30 LIFT_MIN

```
#define LIFT_MIN 1
```

Definition at line 54 of file structures.h.

Referenced by `cmd_lift()`, and `loaddpi()`.

5.36.2.31 MACRO_MAX

```
#define MACRO_MAX 1024
```

Definition at line 50 of file structures.h.

Referenced by `_cmd_macro()`.

5.36.2.32 MD_NAME_LEN

```
#define MD_NAME_LEN 16
```

Definition at line 81 of file structures.h.

Referenced by `cmd_hwsave_kb()`, `cmd_hwsave_mouse()`, `cmd_name()`, `gethwmodename()`, `gethwprofilename()`, `getmodename()`, `hwloadmode()`, `hwtonative()`, and `nativetohw()`.

5.36.2.33 MODE_COUNT

```
#define MODE_COUNT 6
```

Definition at line 99 of file structures.h.

Referenced by `_freeprofile()`, `allocprofile()`, and `readcmd()`.

5.36.2.34 MSG_SIZE

```
#define MSG_SIZE 64
```

Definition at line 175 of file structures.h.

Referenced by `_usb_send()`, `cmd_hwload_kb()`, `cmd_hwload_mouse()`, `cmd_hwsave_kb()`, `cmd_hwsave_mouse()`, `cmd_pollrate()`, `fwupdate()`, `get_fw_version()`, `hwloadmode()`, `loaddpi()`, `loadrgb_kb()`, `loadrgb_mouse()`, `os_↵inputmain()`, `os_usbrecv()`, `os_usb_send()`, `savedpi()`, `savergb_kb()`, `savergb_mouse()`, `setactive_kb()`, `setactive_↵mouse()`, `updatedpi()`, `updatergb_kb()`, and `updatergb_mouse()`.

5.36.2.35 NEEDS_FW_UPDATE

```
#define NEEDS_FW_UPDATE(  
    kb ) ((kb)->fwversion == 0 && HAS_FEATURES((kb), FEAT_FWUPDATE | FEAT_FWVERSION))
```

Definition at line 160 of file structures.h.

Referenced by `_start_dev()`, `readcmd()`, `revertusb()`, `setactive_kb()`, and `setactive_mouse()`.

5.36.2.36 OUTFIFO_MAX

```
#define OUTFIFO_MAX 10
```

Definition at line 24 of file structures.h.

Referenced by `_mknotifynode()`, `_rmnotifynode()`, `inputupdate_keys()`, `nprintf()`, `readcmd()`, `rmdevpath()`, and `updateindicators_kb()`.

5.36.2.37 PR_NAME_LEN

```
#define PR_NAME_LEN 16
```

Definition at line 98 of file structures.h.

Referenced by `cmd_hwload_kb()`, `cmd_hwload_mouse()`, `cmd_profile_name()`, `get_profile_name()`, `hwtonative()`, and `native_to_hw()`.

5.36.2.38 SCROLL_ACCELERATED

```
#define SCROLL_ACCELERATED 0
```

Definition at line 163 of file structures.h.

Referenced by `readcmd()`.

5.36.2.39 SCROLL_MAX

```
#define SCROLL_MAX 10
```

Definition at line 165 of file structures.h.

Referenced by readcmd().

5.36.2.40 SCROLL_MIN

```
#define SCROLL_MIN 1
```

Definition at line 164 of file structures.h.

Referenced by readcmd().

5.36.2.41 SERIAL_LEN

```
#define SERIAL_LEN 34
```

Definition at line 174 of file structures.h.

Referenced by _setupusb(), and os_setupusb().

5.36.2.42 SET_KEYBIT

```
#define SET_KEYBIT(  
    array,  
    index ) do { (array)[(index) / 8] |= 1 << ((index) % 8); } while(0)
```

Definition at line 15 of file structures.h.

Referenced by _cmd_macro(), cmd_notify(), corsair_mousecopy(), hid_kb_translate(), and hid_mouse_translate().

5.36.3 Variable Documentation

5.36.3.1 vtable_keyboard

```
const union devcmd vtable_keyboard
```

Definition at line 28 of file device_vtable.c.

Referenced by get_vtable().

5.36.3.2 vtable_keyboard_nonrgb

```
const union devcmd vtable_keyboard_nonrgb
```

Definition at line 75 of file device_vtable.c.

Referenced by `get_vtable()`.

5.36.3.3 vtable_mouse

```
const union devcmd vtable_mouse
```

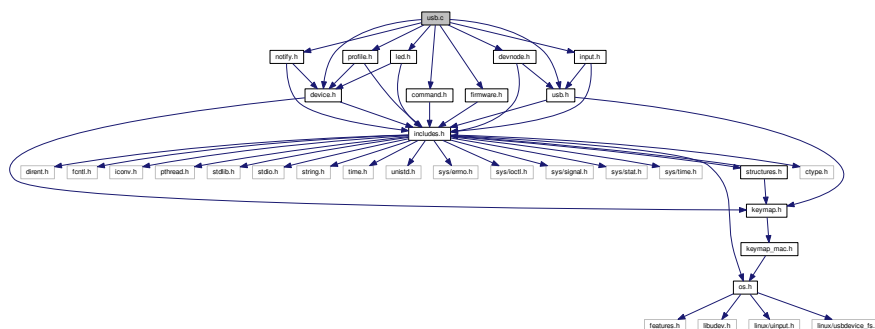
Definition at line 122 of file device_vtable.c.

Referenced by `get_vtable()`.

5.37 usb.c File Reference

```
#include "command.h"
#include "device.h"
#include "devnode.h"
#include "firmware.h"
#include "input.h"
#include "led.h"
#include "notify.h"
#include "profile.h"
#include "usb.h"
```

Include dependency graph for usb.c:



Functions

- const char * [vendor_str](#) (short vendor)
brief.
- const char * [product_str](#) (short product)
brief.
- static const [devcmd](#) * [get_vtable](#) (short vendor, short product)
brief.
- static void * [devmain](#) ([usbdevice](#) *kb)
brief.
- static void * [_setupusb](#) (void *context)
brief.
- void [setupusb](#) ([usbdevice](#) *kb)
- int [revertusb](#) ([usbdevice](#) *kb)
- int [_resetusb](#) ([usbdevice](#) *kb, const char *file, int line)
- int [usb_tryreset](#) ([usbdevice](#) *kb)
- int [_usbSEND](#) ([usbdevice](#) *kb, const [uchar](#) *messages, int count, const char *file, int line)
- int [_usbrecv](#) ([usbdevice](#) *kb, const [uchar](#) *out_msg, [uchar](#) *in_msg, const char *file, int line)
- int [closeusb](#) ([usbdevice](#) *kb)

Variables

- pthread_mutex_t [usbmutex](#) = PTHREAD_MUTEX_INITIALIZER
brief.
- volatile int [reset_stop](#) = 0
brief.
- int [features_mask](#) = -1
brief.
- int [hwload_mode](#)

5.37.1 Function Documentation

5.37.1.1 [_resetusb\(\)](#)

```
int _resetusb (
    usbdevice * kb,
    const char * file,
    int line )
```

[_resetusb](#) Reset a USB device.

First reset the device via [os_resetusb\(\)](#) after a long delay (it may send something to the host). If this worked (retval == 0), give the device another long delay Then perform the initialization via the device specific start() function entry in kb->vtable and if this is successful also, return the result of the device depenten updatetgb() with force=true.

Definition at line 429 of file usb.c.

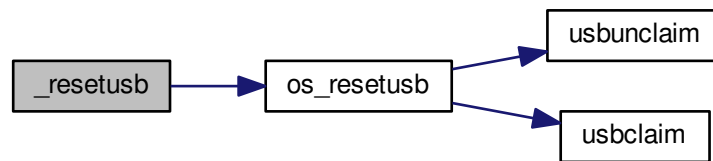
References [usbdevice::active](#), [DELAY_LONG](#), [os_resetusb\(\)](#), and [usbdevice::vtable](#).

```

429                                     {
430     // Perform a USB reset
431     DELAY_LONG(kb);
432     int res = os_resetusb(kb, file, line);
433     if(res)
434         return res;
435     DELAY_LONG(kb);
436     // Re-initialize the device
437     if(kb->vtable->start(kb, kb->active) != 0)
438         return -1;
439     if(kb->vtable->updatergb(kb, 1) != 0)
440         return -1;
441     return 0;
442 }

```

Here is the call graph for this function:



5.37.1.2 _setupusb()

```

static void* _setupusb (
    void * context ) [static]

```

`_setupusb` A horrible function for setting up an usb device

Parameters

<i>context</i>	As <code>_setupusb()</code> is called as a new thread, the <code>kb*</code> is transferred as <code>void*</code>
----------------	--

Returns

a `pthread_t*` 0, here casted as `void*`. Retval is always null

The basic structure of the function is somewhat habituated. It is more like an assembler routine than a structured program. This is not really bad, but just getting used to.

After every action, which can be practically fault-prone, the routine goes into the same error handling: It goes via `goto` to one of two exit labels. The difference is whether or not an unlock has to be performed on the `imutex` variable. In both cases, `closeusb()` is called, then an unlock is performed on the `dmutex`.

The only case where this error handling is not performed is the correct return of the call to `devmain()`. Here simply the return value of `devmain()` is passed to the caller.

In either case, the routine terminates with a `void* 0` because either `devmain()` has returned constant null or the routine itself returns zero.

The basic idea of this routine is the following:

First some initialization of kb standard structured and local vars is done.

- **kb** is set to the pointer given from start environment
- local vars **vendor** and **product** are set to the values from the corresponding fields of kb
- local var **vt** and the **kb->vtable** are both set to the retval of `get_vtable()`
- **kb->features** are set depending on the type of hardware connected:
 - set either to standard non rgb (all common flags like binding, notify, FW, hardware-loading etc) or in case of RGB-device set to standard + RGB, pollrate-change and fw-update
 - exclude all features which are disabled via `feature_mask` (set by daemon CLI parameters)
 - if it is a mouse, add `adjust-rate`
 - if it is a monochrome device, set the flag for RGB-protocol, but single color
- the standard delay time is initialized in `kb->usbdelay`
- A fixed 100ms wait is the start. **Although the `DELAY_LONG` macro is given a parameter, it is ignored. Occasionally refactor it.**
- The first relevant point is the operating system-specific opening of the interface in `os_setupusb()`. As a result, some parameters should be set in kb (name, serial, fwversion, epcount = number of usb endpoints), and all endpoints should be claimed with `usbclaim()`. Claiming is the only point where `os_setupusb()` can produce an error (-1, otherwise 0).
- The following two statements deal with possible errors when setting the kb values in the current routine: If the version or the name was not read correctly, they are set to default values:
 - serial is set to "`<vendor>: <product> -NoID`"
 - the name is set to "`<vendor> <product>`".
- Then the user level input subsystem is activated via `os_openinput()`. There are two file descriptors, one for the mouse and one for the keyboard. **As mentioned in `structures.h`, not the just opened FD numbers are stored under `kb->uinput_kb` or `kb->uinput_mouse`, but the values increased by 1!** The reason is, if the open fails or not open has been done until now, that struct member is set to 0, not to -1 or other negative value. So all usage of this `kb->handle` must be something like "`kb->handle - 1`", as you can find it in the code.
- The next action is to create a separate thread, which gets as parameter kb and starts with `os_inputmain()`. The thread is immediately detached so that it can return its resource completely independently if it should terminate.
- The same happens with `os_setupindicators()`, which initially initializes all LED variables in kb to off and then starts the `_ledthread()` thread with kb as parameter and then detaches it. Here again only the generation of the thread can fail.
- Via an entry in the vable (allocprofile, identical for all three vtable types), `allocprofile()` is called in `profile.c`. With a valid parameter kb, a `usbprofile` structure is allocated and stored as a `kb->profile`. Then `initmode()` is called for each of the initializable modes (`MODE_COUNT`, currently 6). This procedure creates the memory space for the mode information, initializes the range to 0, and then sets the `light.forceupdate` and `dpi.forceupdate` to true. This forces an update later in the initialization of the device.
The first mode is set as the current mode and two force flags are set (this seems to be mode-intersecting flags for light and update).

Warning

There is no error handling for the `allocprofile()` and `initmode()` procedures. However, since they allocate storage areas, the subsequent assignments and initializations can run in a SEGV.

- Not completely understandable is why now via the vtable the function `updateindicators()` is called. But this actually happens in the just started thread `_ledthread()`. Either the initialization is wrong und must done here with force or the overview is lost, what happens when...
Regardless: For a mouse nothing happens here, for a keyboard `updateindicators_kb()` is called via the entry in `kb->vtable`. The first parameter is `kb` again, the second is constant 1 (means force = true). This causes the LED status to be sent after a 5ms delay via `os_sendindicators()` (ioctl with a `usbdevfs_ctrltransfer`). The notification is sent to all currently open notification channels then.
`Setupindicators()` and with it `updateindicators_kb()` can fail.
- From this point - if an error is detected - the error label is addressed by goto statement, which first performs an unlock on the `imutex`. This is interesting because the next statement is exactly this: An unlock on the `imutex`.
- Via vtable the `kb->start()` function is called next. This is the same for a mouse and an RGB keyboard: `start↔_dev()`, for a non RGB keyboard it is `start_kb_nrgb()`.
First parameter is as always `kb`, second is 0 (makeactive = false).
 - In `start_kb_nrgb()` set the keyboard into a so-called software mode (`NK95_HWOFF`) via ioctl with `usbdevfs_ctrltransfer` in function `_nk95cmd()`, which will in turn is called via macro `nk95cmd()` via `start_kb_nrgb()`.
Then two dummy values (active and pollrate) are set in the `kb` structure and ready.
 - `start_dev()` does a bit more - because this function is for both mouse and keyboard. `start_dev()` calls - after setting an extended timeout parameter - `_start_dev()`. Both are located in `device.c`.
 - First, `_start_dev()` attempts to determine the firmware version of the device, but only if two conditions are met: `hwload-mode` is not null (then hw-loading is disabled) and the device has the `FEAT_HWLOAD` feature. Then the firmware and the poll rate are fetched via `getfwversion()`.
If `hwload_mode` is set to "load only once" (`==1`), then the `HWLOAD` feature is masked, so that no further reading can take place.
 - Now check if device needs a firmware update. If so, set it up and leave the function without error.
 - Else load the hardware profile from device if the `hw-pointer` is not set and hw-loading is possible and allowed.
Return error if `mode == 2` (load always) and loading got an error. Else mask the `HWLOAD` feature, because `hwload` must be 1 and the error could be a repeated hw-reading.
Puh, that is real Horror code. It seems to be not faulty, but completely unreadable.
 - Finally, the second parameter of `_startdev()` is used to check whether the device is to be activated. Depending on the parameter, the active or the idle-member in the correspondig vtable is called. These are device-dependent again:

Device	active	idle
RGB Keyboard	<code>cmd_active_kb()</code> means: start the device with a lot of kb-specific initializers (software controlled mode)	<code>cmd_idle_kb()</code> set the device with a lot of kb-specific initializers into the hardware controlled mode)
non RGB Keyboard	<code>cmd_io_none()</code> means: Do nothing	<code>cmd_io_none()</code> means: Do nothing
Mouse	<code>cmd_active_mouse()</code> similar to <code>cmd_active_kb()</code>	<code>cmd_idle_mouse</code> similar to <code>cmd↔_idle_kb()</code>

- If either `start()` succeeded or the next following `usb_tryreset()`, it goes on, otherwise again a hard abort occurs.
- Next, go to `mkdevpath()`. After securing the EUID (effective UID) especially for macOS, work starts really in `_mkdevpath()`. Create - no matter how many devices were registered - either the `ckb0/` files **version**, **pid** and **connected** or the **cmd** command fifo, the first notification fifo **notify0**, **model** and **serial** as well as the **features** of the device and the **pollrate**.

- If all this is done and no error has occurred, a debug info is printed ("Setup finished for ckbx") `updateconnected()` writes the new device into the text file under `ckb0/` and `devmain()` is called.

`devmain()`'s return value is returned by `_setupusb()` when we terminate.

- The remaining code lines are the two exit labels as described above

Definition at line 217 of file `usb.c`.

References `ckb_info`, `closeusb()`, `DELAY_LONG`, `devmain()`, `devpath`, `dmutex`, `FEAT_ADJRATE`, `FEAT_MONOCHROME`, `FEAT_STD_NRGB`, `FEAT_STD_RGB`, `usbdevice::features`, `features_mask`, `get_vtable()`, `imutex`, `INDEX_OF`, `usbdevice::inputthread`, `IS_MONOCHROME`, `IS_MOUSE`, `IS_RGB`, `KB_NAME_LEN`, `keyboard`, `mkdevpath()`, `usbdevice::name`, `os_inputmain()`, `os_inputopen()`, `os_setupindicators()`, `os_setupusb()`, `usbdevice::product`, `product_str()`, `usbdevice::serial`, `SERIAL_LEN`, `updateconnected()`, `USB_DELAY_DEFAULT`, `usb_tryreset()`, `usbdevice::usbdelay`, `usbdevice::vendor`, `vendor_str()`, and `usbdevice::vtable`.

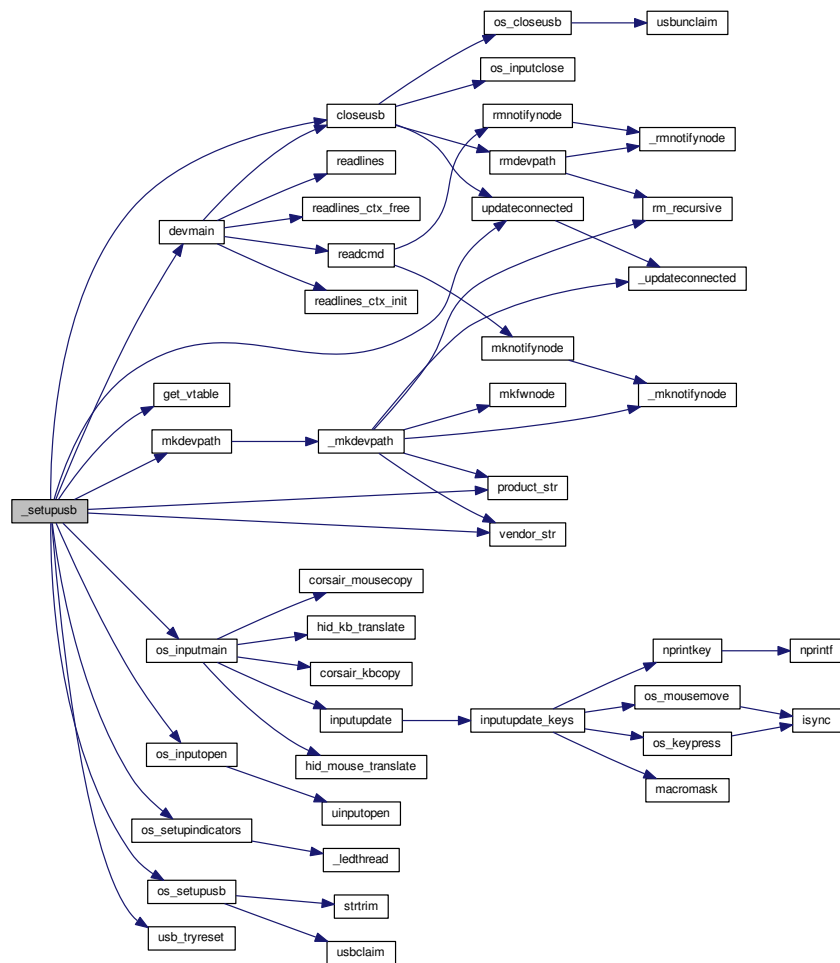
Referenced by `setupusb()`.

```

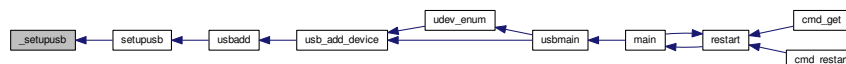
217                                     {
230     usbdevice* kb = context;
231     // Set standard fields
232     short vendor = kb->vendor, product = kb->product;
233     const devcmd* vt = kb->vtable = get_vtable(vendor, product);
234     kb->features = (IS_RGB(vendor, product) ? FEAT_STD_RGB :
FEAT_STD_NRGB) & features_mask;
235     if(IS_MOUSE(vendor, product)) kb->features |= FEAT_ADJRATE;
236     if(IS_MONOCHROME(vendor, product)) kb->features |=
FEAT_MONOCHROME;
237     kb->usbdelay = USB_DELAY_DEFAULT;
238
239     // Perform OS-specific setup
240     DELAY_LONG(kb);
241
242     if(os_setupusb(kb))
243         goto fail;
244
245     // Make up a device name and serial if they weren't assigned
246     if(!kb->serial[0])
247         snprintf(kb->serial, SERIAL_LEN, "%04x:%04x-NoID", kb->
vendor, kb->product);
248     if(!kb->name[0])
249         snprintf(kb->name, KB_NAME_LEN, "%s %s", vendor_str(kb->
vendor), product_str(kb->product));
250
251     // Set up an input device for key events
252     if(os_inputopen(kb))
253         goto fail;
254     if(pthread_create(&kb->inputthread, 0, os_inputmain, kb))
255         goto fail;
256     pthread_detach(kb->inputthread);
257     if(os_setupindicators(kb))
258         goto fail;
259
260     // Set up device
261     vt->allocprofile(kb);
262     vt->updateindicators(kb, 1);
263     pthread_mutex_unlock(imutex(kb));
264     if(vt->start(kb, 0) && usb_tryreset(kb))
265         goto fail_noinput;
266     // Make /dev path
267     if(mkdevpath(kb))
268         goto fail_noinput;
269     // Finished. Enter main loop
270     int index = INDEX_OF(kb, keyboard);
271     ckb_info("Setup finished for %s%d\n", devpath, index);
272     updateconnected();
273     return devmain(kb);
274 fail:
275     pthread_mutex_unlock(imutex(kb));
276     fail_noinput:
277     closeusb(kb);
278     pthread_mutex_unlock(dmutex(kb));
279     return 0;
280 }

```

Here is the call graph for this function:



Here is the caller graph for this function:



5.37.1.3 _usbrecv()

```

int _usbrecv (
    usbdevice * kb,
    const uchar * out_msg,
    uchar * in_msg,

```

```
const char * file,
int line )
```

`_usbrecv` Request data from a USB device by first sending an output packet and then reading the response.

To fully understand this, you need to know about usb: All control is at the usb host (the CPU). If the device wants to communicate something to the host, it must wait for the host to ask. The usb protocol defines the cycles and periods in which actions are to be taken.

So in order to receive a data packet from the device, the host must first send a send request.

This is done by `_usbrecv()` in the first block by sending the MSG_SIZE large data block from `out_msg` via `os_↵_usbrecv()` as it is a machine depending implementation. The usb target device is as always determined over kb.

For `os_usbrecv()` to know that it is a receive request, the `is_recv` parameter is set to true (1). With this, `os_usbrecv()` generates a control package for the hardware, not a data packet.

If sending of the control package is not successful, a maximum of 5 times the transmission is repeated (including the first attempt). If a non-cancelable error is signaled or the drive is stopped via `reset_stop`, `_usbrecv()` immediately returns 0.

After this, the function waits for the requested response from the device using `os_usbrecv()`.

`os_usbrecv()` returns 0, -1 or something else.

Zero signals a serious error which is not treatable and `_usbrecv()` also returns 0.

-1 means that it is a treatable error - a timeout for example - and therefore the next transfer attempt is started after a long pause (DELAY_LONG) if not `reset_stop` or the wrong `hwload_mode` require a termination with a return value of 0.

After 5 attempts, `_usbrecv()` returns and returns 0 as well as an error message.

When data is received, the number of received bytes is returned. This should always be MSG_SIZE, but `os_↵_usbrecv()` can also return less. It should not be more, because then there would be an unhandled buffer overflow, but it could be less. This would be signaled in `os_usbrecv()` with a message.

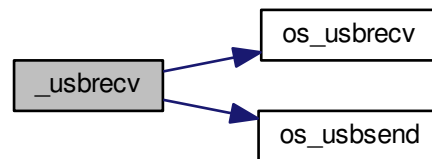
The buffers behind `out_msg` and `in_msg` are MSG_SIZE at least (currently 64 Bytes). More is ok but useless, less brings unpredictable behavior.

Definition at line 602 of file usb.c.

References `ckb_err_fn`, `DELAY_LONG`, `DELAY_MEDIUM`, `DELAY_SHORT`, `hwload_mode`, `os_usbrecv()`, `os_↵_usbrecv()`, and `reset_stop`.

```
602                                     {
603     // Try a maximum of 3 times
604     for(int try = 0; try < 5; try++){
605         // Send the output message
606         DELAY_SHORT(kb);
607         int res = os_usbrecv(kb, out_msg, 1, file, line);
608         if(res == 0)
609             return 0;
610         else if(res == -1){
611             // Retry on temporary failure
612             if(reset_stop)
613                 return 0;
614             DELAY_LONG(kb);
615             continue;
616         }
617         // Wait for the response
618         DELAY_MEDIUM(kb);
619         res = os_usbrecv(kb, in_msg, file, line);
620         if(res == 0)
621             return 0;
622         else if(res != -1)
623             return res;
624         if(reset_stop || hwload_mode != 2)
625             return 0;
626         DELAY_LONG(kb);
627     }
628     // Give up
629     ckb_err_fn("Too many send/recv failures. Dropping.\n", file, line);
630     return 0;
631 }
```

Here is the call graph for this function:



5.37.1.4 _usbsend()

```

int _usbsend (
    usbdevice * kb,
    const uchar * messages,
    int count,
    const char * file,
    int line )
  
```

_usbsend send a logical message completely to the given device

Todo A lot of different conditions are combined in this code. Don't think, it is good in every combination...

The main task of _usbsend () is to transfer the complete logical message from the buffer beginning with *messages* to *count * MSG_SIZE*.

According to usb 2.0 specification, a USB transmits a maximum of 64 byte user data packets. For the transmission of longer messages we need a segmentation. And that is exactly what happens here.

The message is given one by one to [os_usbsend\(\)](#) in MSG_SIZE (= 64) byte large bites.

Attention

This means that the buffer given as argument must be $n * \text{MSG_SIZE}$ Byte long.

An essential constant parameter which is relevant for [os_usbsend\(\)](#) only is `is_rcv = 0`, which means sending.

Now it gets a little complicated again:

- If [os_usbsend\(\)](#) returns 0, only zero bytes could be sent in one of the packets, or it was an error (-1 from the systemcall), but not a timeout. How many Bytes were sent in total from earlier calls does not seem to matter, [_usbsend\(\)](#) returns a total of 0.
- Returns [os_usbsend\(\)](#) -1, first check if **reset_stop** is set globally or (incomprehensible) `hwload_mode` is not set to "always". In either case, [_usbsend\(\)](#) returns 0, otherwise it is assumed to be a temporary transfer error and it simply retransmits the physical packet after a long delay.

- If the return value of `os_usbsend()` was neither 0 nor -1, it specifies the number of bytes transferred. Here is an information hiding conflict with `os_usbsend()` (at least in the Linux version): If `os_usbsend()` can not transfer the entire packet, errors are thrown and the number of bytes sent is returned. `_usbsend()` interprets this as well and remembers the total number of bytes transferred in the local variable **total_sent**. Subsequently, however, transmission is continued with the next complete `MSG_SIZE` block and not with the first of the possibly missing bytes.

Todo Check whether this is the same in the macOS variant. It is not dramatic, but if errors occur, it can certainly irritate the devices completely if they receive incomplete data streams. Do we have errors with the messages "Wrote YY bytes (expected 64)" in the system logs? If not, we do not need to look any further.

When the last packet is transferred, `_usbsend()` returns the effectively counted set of bytes (from **total_sent**). This at least gives the caller the opportunity to check whether something has been lost in the middle.

A bit strange is the structure of the program: Handling the **count** `MSG_SIZE` blocks to be transferred is done in the outer for (...) loop. Repeating the transfer with a treatable error is managed by the inner while(1) loop. This must be considered when reading the code; The "break" on successful block transfer leaves the inner while, not the for (...).

Definition at line 535 of file usb.c.

References `DELAY_LONG`, `DELAY_SHORT`, `hwload_mode`, `MSG_SIZE`, `os_usbsend()`, and `reset_stop`.

```

535                                     {
536     int total_sent = 0;
537     for(int i = 0; i < count; i++){
538         // Send each message via the OS function
539         while(1){
540             DELAY_SHORT(kb);
541             int res = os_usbsend(kb, messages + i * MSG_SIZE, 0, file, line);
542             if(res == 0)
543                 return 0;
544             else if(res != -1){
545                 total_sent += res;
546                 break;
547             }
548             // Stop immediately if the program is shutting down or hardware load is set to tryonce
549             if(reset_stop || hwload_mode != 2)
550                 return 0;
551             // Retry as long as the result is temporary failure
552             DELAY_LONG(kb);
553         }
554     }
555     return total_sent;
556 }
```

Here is the call graph for this function:



5.37.1.5 closeusb()

```
int closeusb (
    usbdevice * kb )
```

closeusb Close a USB device and remove device entry.

An imutex lock ensures first of all, that no communication is currently running from the viewpoint of the driver to the user input device (ie the virtual driver with which characters or mouse movements are sent from the daemon to the operating system as inputs).

If the **kb** has an acceptable value = 0, the index of the device is looked for and with this index [os_inputclose\(\)](#) is called. After this no more characters can be sent to the operating system.

Then the connection to the usb device is capped by [os_closeusb\(\)](#).

Todo What is not yet comprehensible is the call to [updateconnected\(\)](#) BEFORE [os_closeusb\(\)](#). Should that be in the other sequence? Or is [updateconnected\(\)](#) not displaying the connected usb devices, but the representation which uinput devices are loaded? Questions about questions ...

If there is no valid **handle**, only [updateconnected\(\)](#) is called. We are probably trying to disconnect a connection under construction. Not clear.

The cmd pipe as well as all open notify pipes are deleted via [rmdevpath\(\)](#).

This means that nothing can happen to the input path - so the device-specific imutex is unlocked again and remains unlocked.

Also the dmutex is unlocked now, but only to join the thread, which was originally taken under **kb->thread** (which started with [_setupusb\(\)](#)) with [pthread_join\(\)](#) again. Because of the closed devices that thread would have to quit sometime

See also

the hack note with [rmdevpath\(\)](#)

As soon as the thread is caught, the dmutex is locked again, which is what I do not understand yet: What other thread can do usb communication now?

If the vtable exists for the given kb (why not? It seems to have race conditions here!!), via the vtable the actually device-specific, but still everywhere identical [freeprofile\(\)](#) is called. This frees areas that are no longer needed. Then the **usbdevice** structure in its array is set to zero completely.

Error handling is rather unusual in [closeusb\(\)](#); Everything works (no matter what the called functions return), and [closeusb\(\)](#) always returns zero (success).

Definition at line 676 of file usb.c.

References [ckb_info](#), [devpath](#), [dmutex](#), [usbdevice::handle](#), [imutex](#), [INDEX_OF](#), [keyboard](#), [os_closeusb\(\)](#), [os_inputclose\(\)](#), [rmdevpath\(\)](#), [usbdevice::thread](#), [updateconnected\(\)](#), and [usbdevice::vtable](#).

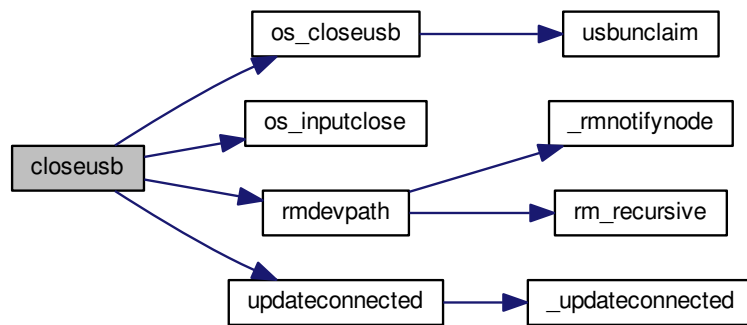
Referenced by [_setupusb\(\)](#), [devmain\(\)](#), [quitWithLock\(\)](#), and [usb_rm_device\(\)](#).

```

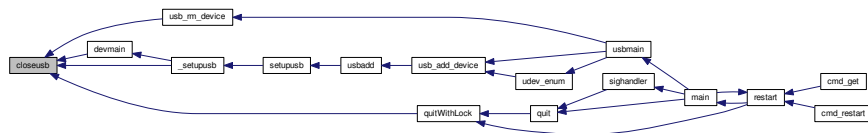
676         {
677     pthread_mutex_lock(imutex(kb));
678     if(kb->handle){
679         int index = INDEX_OF(kb, keyboard);
680         ckb_info("Disconnecting %s%d\n", devpath, index);
681         os_inputclose(kb);
682         updateconnected();
683         // Close USB device
684         os_closeusb(kb);
685     } else
686         updateconnected();
687     rmdevpath(kb);
688
689     // Wait for thread to close
690     pthread_mutex_unlock(imutex(kb));
691     pthread_mutex_unlock(dmutex(kb));
692     pthread_join(kb->thread, 0);
693     pthread_mutex_lock(dmutex(kb));
694
695     // Delete the profile and the control path
696     if(!kb->vtable)
697         return 0;
698     kb->vtable->freeprofile(kb);
699     memset(kb, 0, sizeof(usbdevice));
700     return 0;
701 }

```

Here is the call graph for this function:



Here is the caller graph for this function:



5.37.1.6 devmain()

```

static void* devmain (
    usbdevice * kb ) [static]

```

`devmain` is called by `_setupusb`

Parameters

<i>kb</i>	the pointer to the device. Even if it has the name kb, it is valid also for a mouse (the whole driver seems to be implemented first for a keyboard).
-----------	--

Returns

always a nullptr

Synchronization

The syncing via mutexes is interesting:

1. *imutex* (the Input mutex)

This one is locked in `setupusb()`. That function does only two things: Locking the mutex and trying to start a thread at `_setupusb()`. `_setupusb()` unlocks *imutex* after getting some buffers and initializing internal structures from the indicators (this function often gets problems with error messages like "unable to read indicators" or "Timeout bla blubb").

Warning

have a look at `updateindicators()` later.

if creating the thread is not successful, the *imutex* remains blocked. Have a look at `setupusb()` later.

2. *dmutex* (the Device mutex)

This one is very interesting, because it is handled in `devmain()`. It seems that it is locked only in `_ledthread()`, which is a thread created in `os_setupindicators()`. `os_setupindicators()` again is called in `_setupusb()` long before calling `devmain()`. So this mutex is locked when we start the function as the old comment says. Before reading from the FIFO and direct afterwards an unlock..lock sequence is implemented here. Even if only the function `readlines()` should be surrounded by the unlock..lock, the variable definition of the line pointer is also included here. Not nice, but does not bother either. Probably the Unlock..lock is needed so that now another process can change the control structure *linectx* while we wait in `readlines()`.

Todo Hope to find the need for dmutex usage later.

Should this function be declared as `pthread_t*` function, because of the definition of `pthread-create`?
But `void*` works also...

Attention

dmutex should still be locked when this is called

First a `readlines_ctx` buffer structure is initialized by `readlines_ctx_init()`.

After some setup functions, beginning in `_setupusb()` which has called `devmain()`, we read the command input-
Fifo designated to that device in an endless loop. This loop has two possible exits (plus reaction to signals, not mentioned here).

If the reading via `readlines()` is successful (we might have read multiple lines), the interpretation is done by `readcmd()` iff the connection to the device is still available (checked via `IS_CONNECTED(kb)`). This is true if the kb-structure has a handle and an event pointer both != Null). If not, the loop is left (the first exit point).

if nothing is in the line buffer (some magic interrupt?), continue in the endless while without any reaction.

Todo `readcmd()` gets a **line**, not **lines**. Have a look on that later.

Is the condition `IS_CONNECTED` valid? What functions change the condition for the macro?

If interpretation and communication with the usb device got errors, they are signalled by `readcmd()` (non zero retcode). In this case the usb device is closed via `closeusb()` and the endless loop is left (the second exit point).

After leaving the endless loop the readlines-ctx structure and its buffers are freed by `readlines_ctx_free()`.

Definition at line 135 of file usb.c.

References `ckb_warn`, `closeusb()`, `dmutex`, `usbdevice::infifo`, `IS_CONNECTED`, `usbdevice::name`, `readcmd()`, `readlines()`, `readlines_ctx_free()`, and `readlines_ctx_init()`.

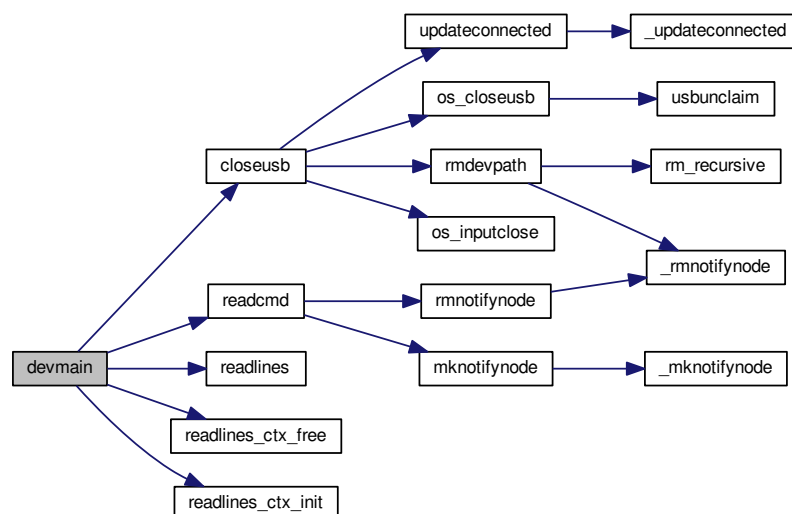
Referenced by `_setupusb()`.

```

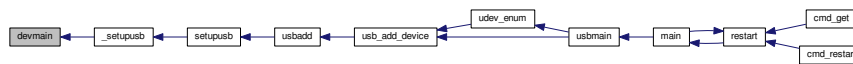
135
137     int kbfifo = kb->infifo - 1;
140     readlines_ctx_t linectx;
141     readlines_ctx_init(&linectx);
146     while(1){
153         pthread_mutex_unlock(dmutex(kb));
154         // Read from FIFO
155         const char* line;
156         int lines = readlines(kbfifo, linectx, &line);
157         pthread_mutex_lock(dmutex(kb));
158         // End thread when the handle is removed
159         if(!IS_CONNECTED(kb)) {
160             ckb_warn("devmain: not connected: %s\n", kb->name);
161             break;
162         }
166         if(lines){
169             if(readcmd(kb, line)){
175                 // USB transfer failed; destroy device
176                 ckb_warn("devmain: readcmd got error, %s\n", kb->name);
177                 closeusb(kb);
178                 break;
179             }
180         }
181     }
182     pthread_mutex_unlock(dmutex(kb));
185     readlines_ctx_free(linectx);
186     return 0;
187 }

```

Here is the call graph for this function:



Here is the caller graph for this function:



5.37.1.7 get_vtable()

```
static const devcmd* get_vtable (
    short vendor,
    short product ) [static]
```

get_vtable returns the correct vtable pointer

Parameters

<i>vendor</i>	short usb vendor ID
<i>product</i>	short usb product ID

Returns

Depending on the type and model, the corresponding vtable pointer is returned (see below)

At present, we have three different vtables:

- `vtable_mouse` is used for all mouse types. This may be wrong with some newer mice?
- `vtable_keyboard` is used for all RGB Keyboards.
- `vtable_keyboard_nonrgb` for all the rest.

Todo Is the last point really a good decision and always correct?

Definition at line 102 of file `usb.c`.

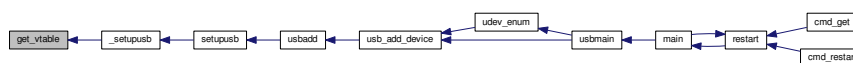
References `IS_MOUSE`, `IS_RGB`, `vtable_keyboard`, `vtable_keyboard_nonrgb`, and `vtable_mouse`.

Referenced by `_setupusb()`.

```

102
103     return IS_MOUSE(vendor, product) ? &vtable_mouse :
      IS_RGB(vendor, product) ? &vtable_keyboard : &
      vtable_keyboard_nonrgb;
104 }
```

Here is the caller graph for this function:



5.37.1.8 product_str()

```
const char* product_str (
    short product )
```

product_str returns a condensed view on what type of device we have.

At present, various models and their properties are known from corsair products. Some models differ in principle (mice and keyboards), others differ in the way they function (for example, RGB and non RGB), but they are very similar.

Here, only the first point is taken into consideration and we return a unified model string. If the model is not known with its number, *product_str* returns an empty string.

The model numbers and corresponding strings with the numbers in hex-string are defined in [usb.h](#)

At present, this function is used to initialize `kb->name` and to give information in debug strings.

Attention

The combinations below have to fit to the combinations in the macros mentioned above. So if you add a device with a new number, change both.

Todo There are macros defined in [usb.h](#) to detect all the combinations below. the only difference is the parameter: The macros need the *kb**, *product_str()* needs the *product ID*

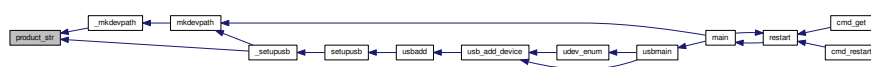
Definition at line 70 of file `usb.c`.

References `P_HARPOON`, `P_K65`, `P_K65_LUX`, `P_K65_NRGB`, `P_K65_RFIRE`, `P_K70`, `P_K70_LUX`, `P_K70_LUX_NRGB`, `P_K70_NRGB`, `P_K70_RFIRE`, `P_K70_RFIRE_NRGB`, `P_K95`, `P_K95_NRGB`, `P_K95_PLATINUM`, `P_M65`, `P_M65_PRO`, `P_SABRE_L`, `P_SABRE_N`, `P_SABRE_O`, `P_SABRE_O2`, `P_SCIMITAR`, `P_SCIMITAR_PRO`, `P_STRAFE`, and `P_STRAFE_NRGB`.

Referenced by `_mkdevpath()`, and `_setupusb()`.

```
70
71     if (product == P_K95 || product == P_K95_NRGB || product ==
72         P_K95_PLATINUM)
73         return "k95";
74     if (product == P_K70 || product == P_K70_NRGB || product ==
75         P_K70_LUX || product == P_K70_LUX_NRGB || product ==
76         P_K70_RFIRE || product == P_K70_RFIRE_NRGB)
77         return "k70";
78     if (product == P_K65 || product == P_K65_NRGB || product ==
79         P_K65_LUX || product == P_K65_RFIRE)
80         return "k65";
81     if (product == P_STRAFE || product == P_STRAFE_NRGB)
82         return "strafe";
83     if (product == P_M65 || product == P_M65_PRO)
84         return "m65";
85     if (product == P_SABRE_O || product == P_SABRE_L || product ==
86         P_SABRE_N || product == P_SABRE_O2 || product == P_HARPOON)
87         return "sabre";
88     if (product == P_SCIMITAR || product == P_SCIMITAR_PRO)
89         return "scimitar";
90     return "";
```

Here is the caller graph for this function:



5.37.1.9 revertusb()

```
int revertusb (
    usbdevice * kb )
```

revertusb sets a given device to inactive (hardware controlled) mode if not a fw-upgrade is indicated

First is checked, whether a firmware-upgrade is indicated for the device. If so, `revertusb()` returns 0.

Todo Why is this useful? Are there problems seen with deactivating a device with older fw-version??? Why isn't this an error indicating reason and we return success (0)?

Anyway, the following steps are similar to some other procs, dealing with low level usb handling:

- If we do not have an RGB device, a simple setting to Hardware-mode (NK95_HWON) is sent to the device via `nk95cmd()`.

Todo The return value of `nk95cmd()` is ignored (but sending the ioctl may produce an error and `_nk95_cmd` will indicate this), instead `revertusb()` returns success in any case.

- If we have an RGB device, `setactive()` is called with second param active = false. That function will have a look on differences between keyboards and mice. More precisely `setactive()` is just a macro to call via the `kb->vtable` entries either the `active()` or the `idle()` function where the `vtable` points to. `setactive()` may return error indications. If so, `revertusb()` returns -1, otherwise 0 in any other case.

Definition at line 410 of file `usb.c`.

References `FEAT_RGB`, `HAS_FEATURES`, `NEEDS_FW_UPDATE`, `NK95_HWON`, `nk95cmd`, and `setactive`.

Referenced by `quitWithLock()`.

```
410     {
411         if (NEEDS_FW_UPDATE(kb))
412             return 0;
413         if (!HAS_FEATURES(kb, FEAT_RGB)) {
414             nk95cmd(kb, NK95_HWON);
415             return 0;
416         }
417         if (setactive(kb, 0))
418             return -1;
419         return 0;
420     }
```

Here is the caller graph for this function:



5.37.1.10 setupusb()

```
void setupusb (
    usbdevice * kb )
```

setupusb starts a thread with kb as parameter and `_setupusb()` as entrypoint.

Set up a USB device after its handle is open. Spawns a new thread `_setupusb()` with standard parameter kb. dmutex must be locked prior to calling this function. The function will unlock it when finished. In kb->thread the thread id is mentioned, because `closeusb()` needs this info for joining that thread again.

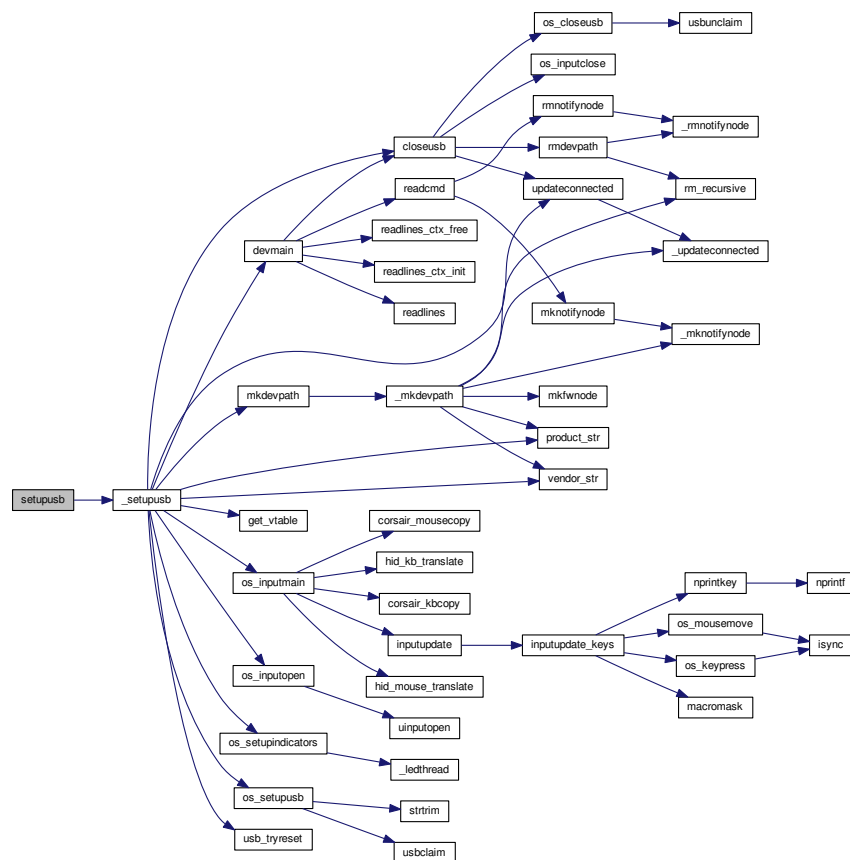
Definition at line 389 of file usb.c.

References `_setupusb()`, `ckb_err`, `imutex`, and `usbdevice::thread`.

Referenced by `usbadd()`.

```
389 {
390     pthread_mutex_lock(&imutex(kb));
391     if(pthread_create(&kb->thread, 0, _setupusb, kb))
392         ckb_err("Failed to create USB thread\n");
393 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



5.37.1.11 usb_tryreset()

```
int usb_tryreset (
    usbdevice * kb )
```

usb_tryreset does what the name means: Try to reset the usb via [resetusb\(\)](#)

This function is called if an usb command ran into an error in case of one of the following two situations:

- When setting up a new usb device and the start() function got an error (

See also

[_setupusb\(\)](#)

- If upgrading to a new firmware gets an error (

See also

[cmd_fwupdate\(\)](#).

The previous action which got the error will NOT be re-attempted.

In an endless loop [usb_tryreset\(\)](#) tries to reset the given usb device via the macro [resetusb\(\)](#).

This macro calls [_resetusb\(\)](#) with debugging information.

[_resetusb\(\)](#) sends a command via the operating system dependent function [os_resetusb\(\)](#) and - if successful - reinitializes the device. [os_resetusb\(\)](#) returns -2 to indicate a broken device and all structures should be removed for it.

In that case, the loop is terminated, an error message is produced and [usb_tryreset\(\)](#) returns -1.

In case [resetusb\(\)](#) has success, the endless loop is left via a return 0 (success).

If the return value from [resetusb\(\)](#) is -1, the loop is continued with the next try.

If the global variable **reset_stop** is set directly when the function is called or after each try, [usb_tryreset\(\)](#) stops working and returns -1.

Todo Why does [usb_tryreset\(\)](#) hide the information returned from [resetusb\(\)](#)? Isn't it needed by the callers?

Definition at line 468 of file usb.c.

References [ckb_err](#), [ckb_info](#), [hwlload_mode](#), [reset_stop](#), and [resetusb](#).

Referenced by [_setupusb\(\)](#), and [cmd_fwupdate\(\)](#).

```

468                                     {
469     if(reset_stop)
470         return -1;
471     ckb_info("Attempting reset...\n");
472     while(1){
473         int res = resetusb(kb);
474         if(!res){
475             ckb_info("Reset success\n");
476             return 0;
477         }
478         if(res == -2 || reset_stop)
479             break;
480     }
481     ckb_err("Reset failed. Disconnecting.\n");
482     return -1;
483 }

```

Here is the caller graph for this function:



5.37.1.12 vendor_str()

```

const char* vendor_str (
    short vendor )

```

uncomment the following Define to see USB packets sent to the device

vendor_str returns "corsair" iff the given *vendor* argument is equal to *V_CORSAIR* (0x1bc) else it returns ""

Attention

There is also a string defined *V_CORSAIR_STR*, which returns the device number as string in hex "1b1c".

Definition at line 43 of file usb.c.

References *V_CORSAIR*.

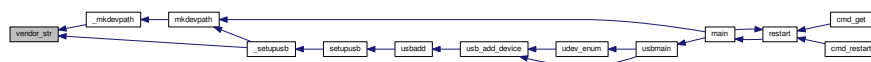
Referenced by *_mkdevpath()*, and *_setupusb()*.

```

43                                     {
44     if (vendor == V_CORSAIR)
45         return "corsair";
46     return "";
47 }

```

Here is the caller graph for this function:



5.37.2 Variable Documentation

5.37.2.1 features_mask

```
int features_mask = -1
```

features_mask Mask of features to exclude from all devices

That bit mask ist set to enable all (-1). When interpreting the input parameters, some of these bits can be cleared. At the moment binding, notifying and mouse-acceleration can be disabled via command line. Have a look at [main\(\)](#) in [main.c](#) for details.

Definition at line 35 of file usb.c.

Referenced by [_setupusb\(\)](#), and [main\(\)](#).

5.37.2.2 hwload_mode

```
int hwload_mode
```

Definition at line 7 of file device.c.

Referenced by [_start_dev\(\)](#), [_usbrecv\(\)](#), [_usbsend\(\)](#), and [usb_tryreset\(\)](#).

5.37.2.3 reset_stop

```
volatile int reset_stop = 0
```

reset_stop is boolean: Reset stopper for when the program shuts down.

Is set only by [quit\(\)](#) to true (1) to inform several usb_* functions to end their loops and tries.

Definition at line 25 of file usb.c.

Referenced by [_usbrecv\(\)](#), [_usbsend\(\)](#), [quitWithLock\(\)](#), and [usb_tryreset\(\)](#).

5.37.2.4 usbmutex

```
pthread_mutex_t usbmutex = PTHREAD_MUTEX_INITIALIZER
```

usbmutex is a never referenced mutex!

Todo We should have a look why this mutex is never used.

Definition at line 17 of file usb.c.

- #define P_K70_RFIRE_NRGB 0x1b3a
 - #define P_K70_RFIRE_NRGB_STR "1b3a"
 - #define IS_K70(kb) ((kb)->vendor == V_CORSAIR && ((kb)->product == P_K70 || (kb)->product == P_K70_NRGB || (kb)->product == P_K70_RFIRE || (kb)->product == P_K70_RFIRE_NRGB || (kb)->product == P_K70_LUX || (kb)->product == P_K70_LUX_NRGB))
 - #define P_K95 0x1b11
 - #define P_K95_STR "1b11"
 - #define P_K95_NRGB 0x1b08
 - #define P_K95_NRGB_STR "1b08"
 - #define P_K95_PLATINUM 0x1b2d
 - #define P_K95_PLATINUM_STR "1b2d"
 - #define IS_K95(kb) ((kb)->vendor == V_CORSAIR && ((kb)->product == P_K95 || (kb)->product == P_K95_NRGB || (kb)->product == P_K95_PLATINUM))
 - #define P_STRAFE 0x1b20
 - #define P_STRAFE_STR "1b20"
 - #define P_STRAFE_NRGB 0x1b15
 - #define P_STRAFE_NRGB_STR "1b15"
 - #define IS_STRAFE(kb) ((kb)->vendor == V_CORSAIR && ((kb)->product == P_STRAFE || (kb)->product == P_STRAFE_NRGB))
 - #define P_M65 0x1b12
 - #define P_M65_STR "1b12"
 - #define P_M65_PRO 0x1b2e
 - #define P_M65_PRO_STR "1b2e"
 - #define IS_M65(kb) ((kb)->vendor == V_CORSAIR && ((kb)->product == P_M65 || (kb)->product == P_M65_PRO))
 - #define P_SABRE_O 0x1b14 /* optical */
 - #define P_SABRE_O_STR "1b14"
 - #define P_SABRE_L 0x1b19 /* laser */
 - #define P_SABRE_L_STR "1b19"
 - #define P_SABRE_N 0x1b2f /* new? */
 - #define P_SABRE_N_STR "1b2f"
 - #define P_SABRE_O2 0x1b32 /* Observed on a CH-9000111-EU model SABRE */
 - #define P_SABRE_O2_STR "1b32"
 - #define P_HARPOON 0x1b3c /* Harpoon test */
 - #define P_HARPOON_STR "1b3c"
 - #define IS_SABRE(kb) ((kb)->vendor == V_CORSAIR && ((kb)->product == P_SABRE_O || (kb)->product == P_SABRE_L || (kb)->product == P_SABRE_N || (kb)->product == P_SABRE_O2 || (kb)->product == P_HARPOON))
 - #define P_SCIMITAR 0x1b1e
 - #define P_SCIMITAR_STR "1b1e"
 - #define P_SCIMITAR_PRO 0x1b3e
 - #define P_SCIMITAR_PRO_STR "1b3e"
 - #define IS_SCIMITAR(kb) ((kb)->vendor == V_CORSAIR && ((kb)->product == P_SCIMITAR || (kb)->product == P_SCIMITAR_PRO))
 - #define IS_RGB(vendor, product) ((vendor) == (V_CORSAIR) && (product) != (P_K65_NRGB) && (product) != (P_K70_NRGB) && (product) != (P_K95_NRGB))
- RGB vs non-RGB test (note: non-RGB Strafe is still considered "RGB" in that it shares the same protocol. The difference is denoted with the "monochrome" feature).*
- #define IS_MONOCHROME(vendor, product) ((vendor) == (V_CORSAIR) && (product) == (P_STRAFE_NRGB))
- The difference between non RGB and monochrome is, that monochrome has lights, but just in one color. nonRGB has no lights. Change this if new **monochrome** devices are added.*
- #define IS_RGB_DEV(kb) IS_RGB((kb)->vendor, (kb)->product)
- For calling with a usbdevice*, vendor and product are extracted and IS_RGB() is returned.*
- #define IS_MONOCHROME_DEV(kb) IS_MONOCHROME((kb)->vendor, (kb)->product)

- For calling with a `usbdevice*`, `vendor` and `product` are extracted and `IS_MONOCHROME()` is returned.
- `#define IS_FULLRANGE(kb) (IS_RGB((kb)->vendor, (kb)->product) && (kb)->product != P_K65 && (kb)->product != P_K70 && (kb)->product != P_K95)`
Full color range (16.8M) vs partial color range (512)
 - `#define IS_MOUSE(vendor, product) ((vendor) == (V_CORSAIR) && ((product) == (P_M65) || (product) == (P_M65_PRO) || (product) == (P_SABRE_O) || (product) == (P_SABRE_L) || (product) == (P_SABRE_N) || (product) == (P_SCIMITAR) || (product) == (P_SCIMITAR_PRO) || (product) == (P_SABRE_O2)))`
Mouse vs keyboard test.
 - `#define IS_MOUSE_DEV(kb) IS_MOUSE((kb)->vendor, (kb)->product)`
For calling with a `usbdevice*`, `vendor` and `product` are extracted and `IS_MOUSE()` is returned.
 - `#define DELAY_SHORT(kb) usleep((int)(kb)->usbdelay * 1000)`
USB delays for when the keyboards get picky about timing That was the original comment, but it is used anytime. The short delay is used before any send or receive.
 - `#define DELAY_MEDIUM(kb) usleep((int)(kb)->usbdelay * 10000)`
the medium delay is used after sending a command before waiting for the answer.
 - `#define DELAY_LONG(kb) usleep(100000)`
The longest delay takes place where something went wrong (eg when resetting the device)
 - `#define USB_DELAY_DEFAULT 5`
This constant is used to initialize `kb->usbdelay`. It is used in many places (see macros above) but often also overwritten to the fixed value of 10. Pure Hacker code.
 - `#define resetusb(kb) _resetusb(kb, __FILE_NOPATH__, __LINE__)`
`resetusb()` is just a macro to call `_resetusb()` with debuggin constants (file, lineno)
 - `#define usbsend(kb, messages, count) _usbsend(kb, messages, count, __FILE_NOPATH__, __LINE__)`
`usbsend` macro is used to wrap `_usbsend()` with debugging information (file and lineno)
 - `#define usbrecv(kb, out_msg, in_msg) _usbrecv(kb, out_msg, in_msg, __FILE_NOPATH__, __LINE__)`
`usbrecv` macro is used to wrap `_usbrecv()` with debugging information (file and lineno)
 - `#define nk95cmd(kb, command) _nk95cmd(kb, (command) >> 16 & 0xFF, (command) & 0xFFFF, __FILE_NOPATH__, __LINE__)`
`nk95cmd()` macro is used to wrap `_nk95cmd()` with debugging information (file and lineno). the command structure is different:
Just the bits 23..16 are used as bits 7..0 for `bRequest`
Bits 15..0 are used as `wValue`
 - `#define NK95_HWOFF 0x020030`
Hardware-specific commands for the K95 nonRGB,.
 - `#define NK95_HWON 0x020001`
Hardware playback on.
 - `#define NK95_M1 0x140001`
Switch to mode 1.
 - `#define NK95_M2 0x140002`
Switch to mode 2.
 - `#define NK95_M3 0x140003`
Switch to mode 3.

Functions

- `const char * vendor_str` (short vendor)
uncomment the following Define to see USB packets sent to the device
- `const char * product_str` (short product)
`product_str` returns a condensed view on what type of device we have.
- `int usbmain ()`
Start the USB main loop. Returns program exit code when finished.
- `void usbkill ()`

- Stop the USB system.*
- void [setupusb](#) ([usbdevice](#) *kb)
setupusb starts a thread with kb as parameter and [_setupusb\(\)](#) as entrypoint.
 - int [os_setupusb](#) ([usbdevice](#) *kb)
os_setupusb OS-specific setup for a specific usb device.
 - void * [os_inputmain](#) (void *context)
os_inputmain is run in a separate thread and will be detached from the main thread, so it needs to clean up its own resources.
 - int [revertusb](#) ([usbdevice](#) *kb)
revertusb sets a given device to inactive (hardware controlled) mode if not a fw-upgrade is indicated
 - int [closeusb](#) ([usbdevice](#) *kb)
closeusb Close a USB device and remove device entry.
 - void [os_closeusb](#) ([usbdevice](#) *kb)
os_closeusb unclaim it, destroy the udev device and clear data structures at kb
 - int [_resetusb](#) ([usbdevice](#) *kb, const char *file, int line)
_resetusb Reset a USB device.
 - int [os_resetusb](#) ([usbdevice](#) *kb, const char *file, int line)
os_resetusb is the os specific implementation for resetting usb
 - int [_usbSEND](#) ([usbdevice](#) *kb, const [uchar](#) *messages, int count, const char *file, int line)
_usbSEND send a logical message completely to the given device
 - int [_usbRECV](#) ([usbdevice](#) *kb, const [uchar](#) *out_msg, [uchar](#) *in_msg, const char *file, int line)
_usbRECV Request data from a USB device by first sending an output packet and then reading the response.
 - int [os_usbSEND](#) ([usbdevice](#) *kb, const [uchar](#) *out_msg, int is_recv, const char *file, int line)
os_usbSEND sends a data packet (MSG_SIZE = 64) Bytes long
 - int [os_usbRECV](#) ([usbdevice](#) *kb, [uchar](#) *in_msg, const char *file, int line)
os_usbRECV receives a max MSGSIZE long buffer from usb device
 - void [os_sendindicators](#) ([usbdevice](#) *kb)
os_sendindicators update the indicators for the special keys (Numlock, Capslock and what else?)
 - int [_nk95cmd](#) ([usbdevice](#) *kb, [uchar](#) bRequest, [ushort](#) wValue, const char *file, int line)
_nk95cmd If we control a non RGB keyboard, set the keyboard via ioctl with usbdevfs_ctrltransfer
 - int [usb_tryreset](#) ([usbdevice](#) *kb)
usb_tryreset does what the name means: Try to reset the usb via [resetusb\(\)](#)

5.38.1 Detailed Description

Vendor/product codes

The list of defines in the first part of the file describes the various types of equipment from Corsair and summarizes them according to specific characteristics.

Each device type is described with two defines:

- On the one hand the device ID with which the device can be recognized on the USB as a short
- and on the other hand the same representation as a string, but without leading "0x".

First entry-pair is the Provider ID (vendorID) from Corsair.

Block No.	contains	Devices are bundled via
1	The first block contains the K65-like keyboards, regardless of their properties (RGB, ...).	In summary, they can be queried using the macro IS_K65() .

Block No.	contains	Devices are bundled via
2	the K70-like Keyboards with all their configuration types	summarized by IS_K70() .
3	the K95 series keyboards	collected with the macro IS_K95() .
4	strafe keyboards	IS_STRAFE()
5	M65 mice with and without RGB	IS_M65()
6	The SABRE and HARPOON mice. Maybe this will be divided into two different blocks later because of different number of special keys	IS_SABRE()
7	The Scimitar mouse devices	IS_SCIMITAR()

5.38.2 Macro Definition Documentation

5.38.2.1 DELAY_LONG

```
#define DELAY_LONG(  
    kb ) usleep(100000)
```

Definition at line 153 of file usb.h.

Referenced by [_resetusb\(\)](#), [_setupusb\(\)](#), [_usbrecv\(\)](#), [_usbseend\(\)](#), [cmd_hwload_kb\(\)](#), [cmd_hwload_mouse\(\)](#), [cmd_hwsave_kb\(\)](#), and [cmd_hwsave_mouse\(\)](#).

5.38.2.2 DELAY_MEDIUM

```
#define DELAY_MEDIUM(  
    kb ) usleep((int)(kb)->usbdelay * 10000)
```

Definition at line 150 of file usb.h.

Referenced by [_usbrecv\(\)](#), and [setactive_kb\(\)](#).

5.38.2.3 DELAY_SHORT

```
#define DELAY_SHORT(  
    kb ) usleep((int)(kb)->usbdelay * 1000)
```

Definition at line 147 of file usb.h.

Referenced by [_usbrecv\(\)](#), [_usbseend\(\)](#), and [updateindicators_kb\(\)](#).

5.38.2.4 IS_FULLRANGE

```
#define IS_FULLRANGE(  
    kb ) (IS_RGB((kb)->vendor, (kb)->product) && (kb)->product != P_K65 && (kb)->product  
    != P_K70 && (kb)->product != P_K95)
```

Definition at line 136 of file usb.h.

Referenced by readcmd(), and updatergb_kb().

5.38.2.5 IS_K65

```
#define IS_K65(  
    kb ) ((kb)->vendor == V_CORSAIR && ((kb)->product == P_K65 || (kb)->product ==  
    P_K65_NRGB || (kb)->product == P_K65_LUX || (kb)->product == P_K65_RFIRE))
```

Definition at line 49 of file usb.h.

Referenced by has_key().

5.38.2.6 IS_K70

```
#define IS_K70(  
    kb ) ((kb)->vendor == V_CORSAIR && ((kb)->product == P_K70 || (kb)->product ==  
    P_K70_NRGB || (kb)->product == P_K70_RFIRE || (kb)->product == P_K70_RFIRE_NRGB || (kb)->product  
    == P_K70_LUX || (kb)->product == P_K70_LUX_NRGB))
```

Definition at line 63 of file usb.h.

5.38.2.7 IS_K95

```
#define IS_K95(  
    kb ) ((kb)->vendor == V_CORSAIR && ((kb)->product == P_K95 || (kb)->product ==  
    P_K95_NRGB || (kb)->product == P_K95_PLATINUM))
```

Definition at line 71 of file usb.h.

Referenced by cmd_hwload_kb(), cmd_hwsave_kb(), and has_key().

5.38.2.8 IS_M65

```
#define IS_M65(  
    kb ) ((kb)->vendor == V_CORSAIR && ((kb)->product == P_M65 || (kb)->product ==  
P_M65_PRO))
```

Definition at line 83 of file usb.h.

Referenced by isblack().

5.38.2.9 IS_MONOCHROME

```
#define IS_MONOCHROME(  
    vendor,  
    product ) ((vendor) == (V_CORSAIR) && (product) == (P_STRAFE_NRGB))
```

Definition at line 127 of file usb.h.

Referenced by _setupusb().

5.38.2.10 IS_MONOCHROME_DEV

```
#define IS_MONOCHROME_DEV(  
    kb ) IS_MONOCHROME((kb)->vendor, (kb)->product)
```

Definition at line 133 of file usb.h.

5.38.2.11 IS_MOUSE

```
#define IS_MOUSE(  
    vendor,  
    product ) ((vendor) == (V_CORSAIR) && ((product) == (P_M65) || (product) == (P↵  
_M65_PRO) || (product) == (P_SABRE_O) || (product) == (P_SABRE_L) || (product) == (P_SABRE_N) ||  
(product) == (P_SCIMITAR) || (product) == (P_SCIMITAR_PRO) || (product) == (P_SABRE_O2)))
```

Definition at line 139 of file usb.h.

Referenced by _setupusb(), get_vtable(), has_key(), and os_inputmain().

5.38.2.12 IS_MOUSE_DEV

```
#define IS_MOUSE_DEV(  
    kb ) IS_MOUSE( (kb)->vendor, (kb)->product)
```

Definition at line 142 of file usb.h.

Referenced by readcmd().

5.38.2.13 IS_RGB

```
#define IS_RGB(  
    vendor,  
    product ) ((vendor) == (V_CORSAIR) && (product) != (P_K65_NRGB) && (product) !=  
(P_K70_NRGB) && (product) != (P_K95_NRGB))
```

Definition at line 122 of file usb.h.

Referenced by _setupusb(), get_vtable(), and os_inputmain().

5.38.2.14 IS_RGB_DEV

```
#define IS_RGB_DEV(  
    kb ) IS_RGB( (kb)->vendor, (kb)->product)
```

Definition at line 130 of file usb.h.

5.38.2.15 IS_SABRE

```
#define IS_SABRE(  
    kb ) ((kb)->vendor == V_CORSAIR && ((kb)->product == P_SABRE_O || (kb)->product  
== P_SABRE_L || (kb)->product == P_SABRE_N || (kb)->product == P_SABRE_O2 || (kb)->product ==  
P_HARPOON))
```

Definition at line 95 of file usb.h.

Referenced by has_key(), loadrgb_mouse(), and savergb_mouse().

5.38.2.16 IS_SCIMITAR

```
#define IS_SCIMITAR(  
    kb ) ((kb)->vendor == V_CORSAIR && ((kb)->product == P_SCIMITAR || (kb)->product  
== P_SCIMITAR_PRO))
```

Definition at line 101 of file usb.h.

Referenced by `has_key()`, `loadrgb_mouse()`, and `savergb_mouse()`.

5.38.2.17 IS_STRAFE

```
#define IS_STRAFE(  
    kb ) ((kb)->vendor == V_CORSAIR && ((kb)->product == P_STRAFE || (kb)->product  
== P_STRAFE_NRGB))
```

Definition at line 77 of file usb.h.

Referenced by `savergb_kb()`.

5.38.2.18 NK95_HWOFF

```
#define NK95_HWOFF 0x020030
```

See also

[usb2.0 documentation for details](#). Set Hardware playback off

Definition at line 299 of file usb.h.

Referenced by `start_kb_nrgb()`.

5.38.2.19 NK95_HWON

```
#define NK95_HWON 0x020001
```

Definition at line 302 of file usb.h.

Referenced by `revertusb()`.

5.38.2.20 NK95_M1

```
#define NK95_M1 0x140001
```

Definition at line 305 of file usb.h.

Referenced by `setmodeindex_nrgb()`.

5.38.2.21 NK95_M2

```
#define NK95_M2 0x140002
```

Definition at line 308 of file usb.h.

Referenced by `setmodeindex_nrgb()`.

5.38.2.22 NK95_M3

```
#define NK95_M3 0x140003
```

Definition at line 311 of file usb.h.

Referenced by `setmodeindex_nrgb()`.

5.38.2.23 nk95cmd

```
#define nk95cmd(  
    kb,  
    command ) \_nk95cmd(kb, (command) >> 16 & 0xFF, (command) & 0xFFFF, __FILE_NOPA↵  
    TH__, __LINE__)
```

Definition at line 294 of file usb.h.

Referenced by `revertusb()`, `setmodeindex_nrgb()`, and `start_kb_nrgb()`.

5.38.2.24 P_HARPOON

```
#define P_HARPOON 0x1b3c /* Harpoon test */
```

Definition at line 93 of file usb.h.

Referenced by `product_str()`.

5.38.2.25 P_HARPOON_STR

```
#define P_HARPOON_STR "1b3c"
```

Definition at line 94 of file usb.h.

5.38.2.26 P_K65

```
#define P_K65 0x1b17
```

Definition at line 41 of file usb.h.

Referenced by `product_str()`.

5.38.2.27 P_K65_LUX

```
#define P_K65_LUX 0x1b37
```

Definition at line 45 of file usb.h.

Referenced by `product_str()`.

5.38.2.28 P_K65_LUX_STR

```
#define P_K65_LUX_STR "1b37"
```

Definition at line 46 of file usb.h.

5.38.2.29 P_K65_NRGB

```
#define P_K65_NRGB 0x1b07
```

Definition at line 43 of file usb.h.

Referenced by `product_str()`.

5.38.2.30 P_K65_NRGB_STR

```
#define P_K65_NRGB_STR "1b07"
```

Definition at line 44 of file usb.h.

5.38.2.31 P_K65_RFIRE

```
#define P_K65_RFIRE 0x1b39
```

Definition at line 47 of file usb.h.

Referenced by product_str().

5.38.2.32 P_K65_RFIRE_STR

```
#define P_K65_RFIRE_STR "1b39"
```

Definition at line 48 of file usb.h.

5.38.2.33 P_K65_STR

```
#define P_K65_STR "1b17"
```

Definition at line 42 of file usb.h.

5.38.2.34 P_K70

```
#define P_K70 0x1b13
```

Definition at line 51 of file usb.h.

Referenced by product_str().

5.38.2.35 P_K70_LUX

```
#define P_K70_LUX 0x1b33
```

Definition at line 55 of file usb.h.

Referenced by product_str().

5.38.2.36 P_K70_LUX_NRGB

```
#define P_K70_LUX_NRGB 0x1b36
```

Definition at line 57 of file usb.h.

Referenced by product_str().

5.38.2.37 P_K70_LUX_NRGB_STR

```
#define P_K70_LUX_NRGB_STR "1b36"
```

Definition at line 58 of file usb.h.

5.38.2.38 P_K70_LUX_STR

```
#define P_K70_LUX_STR "1b33"
```

Definition at line 56 of file usb.h.

5.38.2.39 P_K70_NRGB

```
#define P_K70_NRGB 0x1b09
```

Definition at line 53 of file usb.h.

Referenced by product_str().

5.38.2.40 P_K70_NRGB_STR

```
#define P_K70_NRGB_STR "1b09"
```

Definition at line 54 of file usb.h.

5.38.2.41 P_K70_RFIRE

```
#define P_K70_RFIRE 0x1b38
```

Definition at line 59 of file usb.h.

Referenced by product_str().

5.38.2.42 P_K70_RFIRE_NRGB

```
#define P_K70_RFIRE_NRGB 0x1b3a
```

Definition at line 61 of file usb.h.

Referenced by product_str().

5.38.2.43 P_K70_RFIRE_NRGB_STR

```
#define P_K70_RFIRE_NRGB_STR "1b3a"
```

Definition at line 62 of file usb.h.

5.38.2.44 P_K70_RFIRE_STR

```
#define P_K70_RFIRE_STR "1b38"
```

Definition at line 60 of file usb.h.

5.38.2.45 P_K70_STR

```
#define P_K70_STR "1b13"
```

Definition at line 52 of file usb.h.

5.38.2.46 P_K95

```
#define P_K95 0x1b11
```

Definition at line 65 of file usb.h.

Referenced by product_str().

5.38.2.47 P_K95_NRGB

```
#define P_K95_NRGB 0x1b08
```

Definition at line 67 of file usb.h.

Referenced by _nk95cmd(), and product_str().

5.38.2.48 P_K95_NRGB_STR

```
#define P_K95_NRGB_STR "1b08"
```

Definition at line 68 of file usb.h.

5.38.2.49 P_K95_PLATINUM

```
#define P_K95_PLATINUM 0x1b2d
```

Definition at line 69 of file usb.h.

Referenced by product_str().

5.38.2.50 P_K95_PLATINUM_STR

```
#define P_K95_PLATINUM_STR "1b2d"
```

Definition at line 70 of file usb.h.

5.38.2.51 P_K95_STR

```
#define P_K95_STR "1b11"
```

Definition at line 66 of file usb.h.

5.38.2.52 P_M65

```
#define P_M65 0x1b12
```

Definition at line 79 of file usb.h.

Referenced by product_str().

5.38.2.53 P_M65_PRO

```
#define P_M65_PRO 0x1b2e
```

Definition at line 81 of file usb.h.

Referenced by product_str().

5.38.2.54 P_M65_PRO_STR

```
#define P_M65_PRO_STR "1b2e"
```

Definition at line 82 of file usb.h.

5.38.2.55 P_M65_STR

```
#define P_M65_STR "1b12"
```

Definition at line 80 of file usb.h.

5.38.2.56 P_SABRE_L

```
#define P_SABRE_L 0x1b19 /* laser */
```

Definition at line 87 of file usb.h.

Referenced by product_str().

5.38.2.57 P_SABRE_L_STR

```
#define P_SABRE_L_STR "1b19"
```

Definition at line 88 of file usb.h.

5.38.2.58 P_SABRE_N

```
#define P_SABRE_N 0x1b2f /* new? */
```

Definition at line 89 of file usb.h.

Referenced by product_str().

5.38.2.59 P_SABRE_N_STR

```
#define P_SABRE_N_STR "1b2f"
```

Definition at line 90 of file usb.h.

5.38.2.60 P_SABRE_O

```
#define P_SABRE_O 0x1b14 /* optical */
```

Definition at line 85 of file usb.h.

Referenced by product_str().

5.38.2.61 P_SABRE_O2

```
#define P_SABRE_O2 0x1b32 /* Observed on a CH-9000111-EU model SABRE */
```

Definition at line 91 of file usb.h.

Referenced by product_str().

5.38.2.62 P_SABRE_O2_STR

```
#define P_SABRE_O2_STR "1b32"
```

Definition at line 92 of file usb.h.

5.38.2.63 P_SABRE_O_STR

```
#define P_SABRE_O_STR "1b14"
```

Definition at line 86 of file usb.h.

5.38.2.64 P_SCIMITAR

```
#define P_SCIMITAR 0x1b1e
```

Definition at line 97 of file usb.h.

Referenced by product_str().

5.38.2.65 P_SCIMITAR_PRO

```
#define P_SCIMITAR_PRO 0x1b3e
```

Definition at line 99 of file usb.h.

Referenced by product_str().

5.38.2.66 P_SCIMITAR_PRO_STR

```
#define P_SCIMITAR_PRO_STR "1b3e"
```

Definition at line 100 of file usb.h.

5.38.2.67 P_SCIMITAR_STR

```
#define P_SCIMITAR_STR "1b1e"
```

Definition at line 98 of file usb.h.

5.38.2.68 P_STRAFE

```
#define P_STRAFE 0x1b20
```

Definition at line 73 of file usb.h.

Referenced by product_str().

5.38.2.69 P_STRAFE_NRGB

```
#define P_STRAFE_NRGB 0x1b15
```

Definition at line 75 of file usb.h.

Referenced by product_str().

5.38.2.70 P_STRAFE_NRGB_STR

```
#define P_STRAFE_NRGB_STR "1b15"
```

Definition at line 76 of file usb.h.

5.38.2.71 P_STRAFE_STR

```
#define P_STRAFE_STR "1b20"
```

Definition at line 74 of file usb.h.

5.38.2.72 resetusb

```
#define resetusb(  
    kb ) \_resetusb(kb, \_\_FILE\_\_, \_\_LINE\_\_)
```

Definition at line 212 of file usb.h.

Referenced by `usb_tryreset()`.

5.38.2.73 USB_DELAY_DEFAULT

```
#define USB_DELAY_DEFAULT 5
```

Definition at line 158 of file usb.h.

Referenced by `_setupusb()`, and `start_dev()`.

5.38.2.74 usbrecv

```
#define usbrecv(  
    kb,  
    out_msg,  
    in_msg ) \_usbrecv(kb, out_msg, in_msg, \_\_FILE\_\_, \_\_LINE\_\_)
```

Parameters

<i>kb</i>	THE usbdevice*
<i>[IN]</i>	out_msg What information does the caller want from the device?
<i>[O↔UT]</i>	in_msg Here comes the answer; The names represent the usb view, not the view of this function! So INput from usb is OUTput of this function.

Definition at line 254 of file usb.h.

Referenced by `cmd_hwload_kb()`, `cmd_hwload_mouse()`, `getfwversion()`, `hwloadmode()`, `loaddpi()`, `loadrgb_kb()`, and `loadrgb_mouse()`.

5.38.2.75 `usb send`

```
#define usb send(  
    kb,  
    messages,  
    count ) _usb send(kb, messages, count, __FILE__, __LINE__)
```

Parameters

<i>kb</i>	THE usbdevice*
<i>[IN]</i>	messages a Pointer to the first byte of the logical message
<i>[IN]</i>	count how many MSG_SIZE buffers is the logical message long?

Definition at line 237 of file usb.h.

Referenced by `cmd_hwsave_kb()`, `cmd_hwsave_mouse()`, `cmd_pollrate()`, `fwupdate()`, `loadrgb_kb()`, `savedpi()`, `savergb_kb()`, `savergb_mouse()`, `setactive_kb()`, `setactive_mouse()`, `updatedpi()`, `updatergb_kb()`, and `updatergb_↵_mouse()`.

5.38.2.76 `V_CORSAIR`

```
#define V_CORSAIR 0x1b1c
```

Warning

When adding new devices please update `src/ckb/fwupgradedialog.cpp` as well.
It should contain the same vendor/product IDs for any devices supporting firmware updates.
In the same way, all other corresponding files have to be supplemented or modified: Currently known for this are [usb_linux.c](#) and [usb_mac.c](#)

Definition at line 38 of file usb.h.

Referenced by `usb_add_device()`, and `vendor_str()`.

5.38.2.77 `V_CORSAIR_STR`

```
#define V_CORSAIR_STR "1b1c"
```

Definition at line 39 of file usb.h.

Referenced by `udev_enum()`, and `usb_add_device()`.

5.38.3 Function Documentation

5.38.3.1 _nk95cmd()

```
int _nk95cmd (
    usbdevice * kb,
    uchar bRequest,
    ushort wValue,
    const char * file,
    int line )
```

Parameters

<i>kb</i>	THE usbdevice*
<i>bRequest</i>	the byte array with the usb request
<i>wValue</i>	a usb wValue
<i>file</i>	for error message
<i>line</i>	for error message

Returns

1 (true) on failure, 0 (false) on success.

To send control packets to a non RGB non color K95 Keyboard, use this function. Normally it is called via the `nk95cmd()` macro.

If it is the wrong device for which the function is called, 0 is returned and nothing done. Otherwise a `usbdevfs_↵` `ctrltransfer` structure is filled and an `USBDEVFS_CONTROL` `ioctl()` called.

bRequest↵ Type	bRequest	wValue	EP	size	Timeout	data
0x40	see table below to switch hardware-modus at Keyboard	wValue	device	MSG_SIZE	5ms	the message buffer pointer
Host to Device, Type=Vendor, Recipient=Device	bRequest parameter	given wValue Parameter	device 0	0 data to write	5000	null

If a 0 or a negative error number is returned by the `ioctl`, an error message is shown depending on the `errno` or "No data written" if `retval` was 0. In either case 1 is returned to indicate the error. If the `ioctl` returned a value > 0 , 0 is returned to indicate no error.

Currently the following combinations for `bRequest` and `wValue` are used:

Device	what it might to do	constant	bRequest	wValue
non RGB Keyboard	set HW-modus on (leave the ckb driver)	HWON	0x0002	0x0030

Device	what it might to do	constant	bRequest	wValue
non RGB Keyboard	set HW-modus off (initialize the ckb driver)	HWOFF	0x0002	0x0001
non RGB Keyboard	set light modus M1 in single-color keyboards	NK95_M1	0x0014	0x0001
non RGB Keyboard	set light modus M2 in single-color keyboards	NK95_M2	0x0014	0x0002
non RGB Keyboard	set light modus M3 in single-color keyboards	NK95_M3	0x0014	0x0003

See also

[usb.h](#)

Definition at line 181 of file usb_linux.c.

References [ckb_err_fn](#), [usbdevice::handle](#), [P_K95_NRGB](#), and [usbdevice::product](#).

```

181                                     {
182     if(kb->product != P_K95_NRGB)
183         return 0;
184     struct usbdevfs_ctrltransfer transfer = { 0x40, bRequest, wValue, 0, 0, 5000, 0 };
185     int res = ioctl(kb->handle - 1, USBDEVFS_CONTROL, &transfer);
186     if(res <= 0){
187         ckb_err_fn("%s\n", file, line, res ? strerror(errno) : "No data written");
188         return 1;
189     }
190     return 0;
191 }
```

5.38.3.2 _resetusb()

```

int _resetusb (
    usbdevice * kb,
    const char * file,
    int line )
```

Parameters

<i>kb</i>	THE usbdevice*
<i>file</i>	filename for error messages
<i>line</i>	line where it is called for error messages

Returns

Returns 0 on success, -1 if device should be removed

[_resetusb](#) Reset a USB device.

First reset the device via [os_resetusb\(\)](#) after a long delay (it may send something to the host). If this worked (retval == 0), give the device another long delay Then perform the initialization via the device specific start() function entry in kb->vtable and if this is successful also, return the result of the device depenten updatetrgb() with force=true.

Definition at line 429 of file usb.c.

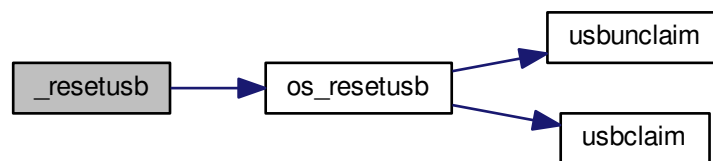
References [usbdevice::active](#), [DELAY_LONG](#), [os_resetusb\(\)](#), and [usbdevice::vtable](#).

```

429                                     {
430     // Perform a USB reset
431     DELAY_LONG(kb);
432     int res = os_resetusb(kb, file, line);
433     if(res)
434         return res;
435     DELAY_LONG(kb);
436     // Re-initialize the device
437     if(kb->vtable->start(kb, kb->active) != 0)
438         return -1;
439     if(kb->vtable->updatergb(kb, 1) != 0)
440         return -1;
441     return 0;
442 }

```

Here is the call graph for this function:



5.38.3.3 _usbrecv()

```

int _usbrecv (
    usbdevice * kb,
    const uchar * out_msg,
    uchar * in_msg,
    const char * file,
    int line )

```

Parameters

	<i>kb</i>	THE usbdevice*
	[IN]	out_msg What information does the caller want from the device?
	[OUT]	in_msg Here comes the answer; The names represent the usb view, not the view of this function! So INput from usb is OUTput of this function.
	[IN]	file for debugging
	[IN]	line for debugging
in	<i>reset_stop</i>	global variable is read

Returns

number of bytes read or zero on failure.

`_usbrecv` Request data from a USB device by first sending an output packet and then reading the response.

To fully understand this, you need to know about usb: All control is at the usb host (the CPU). If the device wants to communicate something to the host, it must wait for the host to ask. The usb protocol defines the cycles and periods in which actions are to be taken.

So in order to receive a data packet from the device, the host must first send a send request.

This is done by `_usbrecv()` in the first block by sending the MSG_SIZE large data block from `out_msg` via `os_↔_usbrecv()` as it is a machine depending implementation. The usb target device is as always determined over `kb`.

For `os_usbrecv()` to know that it is a receive request, the `is_recv` parameter is set to true (1). With this, `os_usbrecv()` generates a control package for the hardware, not a data packet.

If sending of the control package is not successful, a maximum of 5 times the transmission is repeated (including the first attempt). If a non-cancelable error is signaled or the drive is stopped via `reset_stop`, `_usbrecv()` immediately returns 0.

After this, the function waits for the requested response from the device using `os_usbrecv()`.

`os_usbrecv()` returns 0, -1 or something else.

Zero signals a serious error which is not treatable and `_usbrecv()` also returns 0.

-1 means that it is a treatable error - a timeout for example - and therefore the next transfer attempt is started after a long pause (`DELAY_LONG`) if not `reset_stop` or the wrong `hwload_mode` require a termination with a return value of 0.

After 5 attempts, `_usbrecv()` returns and returns 0 as well as an error message.

When data is received, the number of received bytes is returned. This should always be MSG_SIZE, but `os_↔_usbrecv()` can also return less. It should not be more, because then there would be an unhandled buffer overflow, but it could be less. This would be signaled in `os_usbrecv()` with a message.

The buffers behind `out_msg` and `in_msg` are MSG_SIZE at least (currently 64 Bytes). More is ok but useless, less brings unpredictable behavior.

Definition at line 602 of file `usb.c`.

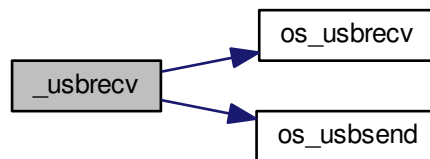
References `ckb_err_fn`, `DELAY_LONG`, `DELAY_MEDIUM`, `DELAY_SHORT`, `hwload_mode`, `os_usbrecv()`, `os_↔_usbrecv()`, and `reset_stop`.

```

602                                     {
603     // Try a maximum of 3 times
604     for(int try = 0; try < 5; try++){
605         // Send the output message
606         DELAY_SHORT(kb);
607         int res = os_usbrecv(kb, out_msg, 1, file, line);
608         if(res == 0)
609             return 0;
610         else if(res == -1){
611             // Retry on temporary failure
612             if(reset_stop)
613                 return 0;
614             DELAY_LONG(kb);
615             continue;
616         }
617         // Wait for the response
618         DELAY_MEDIUM(kb);
619         res = os_usbrecv(kb, in_msg, file, line);
620         if(res == 0)
621             return 0;
622         else if(res != -1)
623             return res;
624         if(reset_stop || hwload_mode != 2)
625             return 0;
626         DELAY_LONG(kb);
627     }
628     // Give up
629     ckb_err_fn("Too many send/recv failures. Dropping.\n", file, line);
630     return 0;
631 }

```

Here is the call graph for this function:



5.38.3.4 _usbsend()

```

int _usbsend (
    usbdevice * kb,
    const uchar * messages,
    int count,
    const char * file,
    int line )
  
```

Parameters

	<i>kb</i>	THE usbdevice*
	[IN]	messages a Pointer to the first byte of the logical message
	[IN]	count how many MSG_SIZE buffers is the logical message long?
	[IN]	file for debugging
	[IN]	line for debugging
in	<i>reset_stop</i>	global variable is read

Returns

number of Bytes sent (ideal == count * MSG_SIZE);
 0 if a block could not be sent and it was not a timeout OR **reset_stop** was required or **hwload_mode** is not set to "always"

_usbsend send a logical message completely to the given device

Todo A lot of different conditions are combined in this code. Don't think, it is good in every combination...

The main task of _usbsend () is to transfer the complete logical message from the buffer beginning with *messages* to **count * MSG_SIZE**.

According to usb 2.0 specification, a USB transmits a maximum of 64 byte user data packets. For the transmission of longer messages we need a segmentation. And that is exactly what happens here.

The message is given one by one to [os_usbsend\(\)](#) in MSG_SIZE (= 64) byte large bites.

Attention

This means that the buffer given as argument must be $n * \text{MSG_SIZE}$ Byte long.

An essential constant parameter which is relevant for `os_usbsend()` only is `is_rcv = 0`, which means sending.

Now it gets a little complicated again:

- If `os_usbsend()` returns 0, only zero bytes could be sent in one of the packets, or it was an error (-1 from the systemcall), but not a timeout. How many Bytes were sent in total from earlier calls does not seem to matter, `_usbsend()` returns a total of 0.
- Returns `os_usbsend()` -1, first check if **reset_stop** is set globally or (incomprehensible) `hwload_mode` is not set to "always". In either case, `_usbsend()` returns 0, otherwise it is assumed to be a temporary transfer error and it simply retransmits the physical packet after a long delay.
- If the return value of `os_usbsend()` was neither 0 nor -1, it specifies the number of bytes transferred. Here is an information hiding conflict with `os_usbsend()` (at least in the Linux version): If `os_usbsend()` can not transfer the entire packet, errors are thrown and the number of bytes sent is returned. `_usbsend()` interprets this as well and remembers the total number of bytes transferred in the local variable **total_sent**. Subsequently, however, transmission is continued with the next complete `MSG_SIZE` block and not with the first of the possibly missing bytes.

Todo Check whether this is the same in the macOS variant. It is not dramatic, but if errors occur, it can certainly irritate the devices completely if they receive incomplete data streams. Do we have errors with the messages "Wrote YY bytes (expected 64)" in the system logs? If not, we do not need to look any further.

When the last packet is transferred, `_usbsend()` returns the effectively counted set of bytes (from **total_sent**). This at least gives the caller the opportunity to check whether something has been lost in the middle.

A bit strange is the structure of the program: Handling the **count** `MSG_SIZE` blocks to be transferred is done in the outer for (...) loop. Repeating the transfer with a treatable error is managed by the inner while(1) loop. This must be considered when reading the code; The "break" on successful block transfer leaves the inner while, not the for (...).

Definition at line 535 of file `usb.c`.

References `DELAY_LONG`, `DELAY_SHORT`, `hwload_mode`, `MSG_SIZE`, `os_usbsend()`, and `reset_stop`.

```

535                                     {
536     int total_sent = 0;
537     for(int i = 0; i < count; i++){
538         // Send each message via the OS function
539         while(1){
540             DELAY_SHORT(kb);
541             int res = os_usbsend(kb, messages + i * MSG_SIZE, 0, file, line);
542             if(res == 0)
543                 return 0;
544             else if(res != -1){
545                 total_sent += res;
546                 break;
547             }
548             // Stop immediately if the program is shutting down or hardware load is set to tryonce
549             if(reset_stop || hwload_mode != 2)
550                 return 0;
551             // Retry as long as the result is temporary failure
552             DELAY_LONG(kb);
553         }
554     }
555     return total_sent;
556 }
```

Here is the call graph for this function:



5.38.3.5 closeusb()

```
int closeusb (
    usbdevice * kb )
```

Parameters

$[IN, OUT \leftrightarrow UT]$	kb
--------------------------------	----

Returns

Returns 0 (everytime. No error handling is done!)

closeusb Close a USB device and remove device entry.

An imutex lock ensures first of all, that no communication is currently running from the viewpoint of the driver to the user input device (ie the virtual driver with which characters or mouse movements are sent from the daemon to the operating system as inputs).

If the **kb** has an acceptable value = 0, the index of the device is looked for and with this index [os_inputclose\(\)](#) is called. After this no more characters can be sent to the operating system.

Then the connection to the usb device is capped by [os_closeusb\(\)](#).

Todo What is not yet comprehensible is the call to [updateconnected\(\)](#) BEFORE [os_closeusb\(\)](#). Should that be in the other sequence? Or is [updateconnected\(\)](#) not displaying the connected usb devices, but the representation which uinput devices are loaded? Questions about questions ...

If there is no valid **handle**, only [updateconnected\(\)](#) is called. We are probably trying to disconnect a connection under construction. Not clear.

The cmd pipe as well as all open notify pipes are deleted via [rmdevpath\(\)](#).

This means that nothing can happen to the input path - so the device-specific imutex is unlocked again and remains unlocked.

Also the dmutex is unlocked now, but only to join the thread, which was originally taken under **kb->thread** (which started with [_setupusb\(\)](#)) with [pthread_join\(\)](#) again. Because of the closed devices that thread would have to quit sometime

See also

the hack note with [rmdevpath\(\)](#)

As soon as the thread is caught, the dmutex is locked again, which is what I do not understand yet: What other thread can do usb communication now?

If the vtable exists for the given kb (why not? It seems to have race conditions here!!), via the vtable the actually device-specific, but still everywhere identical [freeprofile\(\)](#) is called. This frees areas that are no longer needed. Then the **usbdevice** structure in its array is set to zero completely.

Error handling is rather unusual in [closeusb\(\)](#); Everything works (no matter what the called functions return), and [closeusb\(\)](#) always returns zero (success).

Definition at line 676 of file usb.c.

References [ckb_info](#), [devpath](#), [dmutex](#), [usbdevice::handle](#), [imutex](#), [INDEX_OF](#), [keyboard](#), [os_closeusb\(\)](#), [os_inputclose\(\)](#), [rmdevpath\(\)](#), [usbdevice::thread](#), [updateconnected\(\)](#), and [usbdevice::vtable](#).

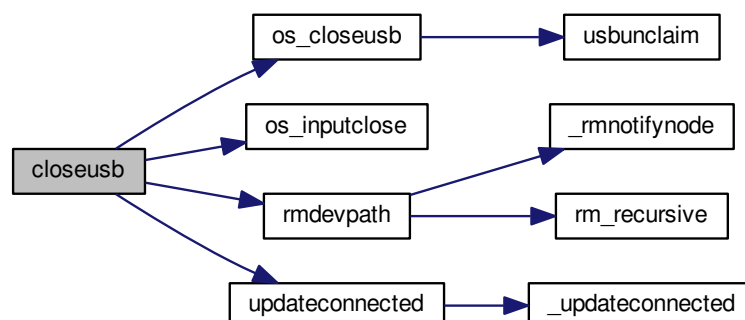
Referenced by [_setupusb\(\)](#), [devmain\(\)](#), [quitWithLock\(\)](#), and [usb_rm_device\(\)](#).

```

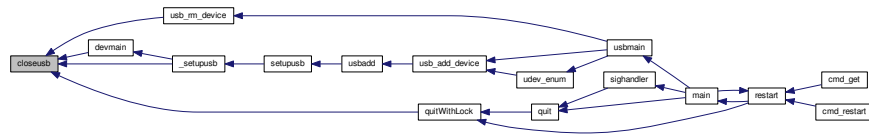
676     {
677     pthread_mutex_lock(&imutex(kb));
678     if(kb->handle){
679         int index = INDEX_OF(kb, keyboard);
680         ckb_info("Disconnecting %s%d\n", devpath, index);
681         os_inputclose(kb);
682         updateconnected();
683         // Close USB device
684         os_closeusb(kb);
685     } else
686         updateconnected();
687     rmdevpath(kb);
688
689     // Wait for thread to close
690     pthread_mutex_unlock(&imutex(kb));
691     pthread_mutex_unlock(&dmutex(kb));
692     pthread_join(kb->thread, 0);
693     pthread_mutex_lock(&dmutex(kb));
694
695     // Delete the profile and the control path
696     if(!kb->vtable)
697         return 0;
698     kb->vtable->freeprofile(kb);
699     memset(kb, 0, sizeof(usbdevice));
700     return 0;
701 }

```

Here is the call graph for this function:



Here is the caller graph for this function:



5.38.3.6 os_closeusb()

```
void os_closeusb (
    usbdevice * kb )
```

Parameters

<i>[IN,OUT,UT]</i>	kb THE usbdevice*
--------------------	-------------------

os_closeusb unclaim it, destroy the udev device and clear data structures at kb

os_closeusb is the linux specific implementation for closing an active usb port.

If a valid handle is given in the kb structure, the usb port is unclaimed ([usbunclaim\(\)](#)).

The device is unrefenced via library function [udev_device_unref\(\)](#).

handle, udev and the first char of kbsyspath are cleared to 0 (empty string for kbsyspath).

Definition at line 423 of file [usb_linux.c](#).

References [usbdevice::handle](#), [INDEX_OF](#), [kbsyspath](#), [keyboard](#), [usbdevice::udev](#), and [usbunclaim\(\)](#).

Referenced by [closeusb\(\)](#).

```

423                                     {
424     if (kb->handle) {
425         usbunclaim(kb, 0);
426         close(kb->handle - 1);
427     }
428     if (kb->udev)
429         udev_device_unref(kb->udev);
430     kb->handle = 0;
431     kb->udev = 0;
432     kbsyspath[INDEX_OF(kb, keyboard)][0] = 0;
433 }
```

Here is the call graph for this function:



device	detect with macro combination	endpoint #	buffer-length
each	none	0	8
RGB Mouse	IS_RGB && IS_MOUSE	1	10
RGB Keyboard	IS_RGB && !IS_MOUSE	1	21
RGB Mouse or Keyboard	IS_RGB	2	MSG_SIZE (64)
non RGB Mouse or Keyboard	!IS_RGB	1	4
non RGB Mouse or Keyboard	!IS_RGB	2	15

Now submit all the URBs via `ioctl(USBDEVFS_SUBMITURB)` with type `USBDEVFS_URB_TYPE_INTERRUPT` (the endpoints are defined as type interrupt). Endpoint number is 0x80..0x82 or 0x83, depending on the model.

The userSpaceFS knows the URBs now, so start monitoring input

if the `ioctl` returns something `!= 0`, let's have a deeper look what happened. Broken devices or shutting down the entire system leads to closing the device and finishing this thread.

If just an EPIPE occurred, give the device a `CLEAR_HALT` and resubmit the URB.

A correct `REAPURB` returns a Pointer to the URB which we now have a closer look into. Lock all following actions with `imutex`.

Process the input depending on type of device. Interpret the actual size of the URB buffer

device	detect with macro combination	seems to be endpoint #	actual buffer-length	function called
mouse (RGB and non RGB)	IS_MOUSE	nA	8, 10 or 11	hid_mouse_translate()
mouse (RGB and non RGB)	IS_MOUSE	nA	MSG_SIZE (64)	corsair_mousecopy()
RGB Keyboard	IS_RGB && !IS_MOUSE	1	8 (BIOS Mode)	hid_kb_translate()
RGB Keyboard	IS_RGB && !IS_MOUSE	2	5 or 21, KB inactive!	hid_kb_translate()
RGB Keyboard	IS_RGB && !IS_MOUSE	3?	MSG_SIZE	corsair_kbcopy()
non RGB Keyboard	!IS_RGB && !IS_MOUSE	nA	nA	hid_kb_translate()

The input data is transformed and copied to the `kb` structure. Now give it to the OS and unlock the `imutex` afterwards.

Re-submit the URB for the next run.

If the endless loop is terminated, clean up by discarding the URBs via `ioctl(USBDEVFS_DISCARDURB)`, free the URB buffers and return a null pointer as thread exit code.

Definition at line 226 of file `usb_linux.c`.

References `usbdevice::active`, `ckb_info`, `ckb_warn`, `corsair_kbcopy()`, `corsair_mousecopy()`, `devpath`, `usbdevice::epcount`, `usbdevice::handle`, `hid_kb_translate()`, `hid_mouse_translate()`, `imutex`, `INDEX_OF`, `usbdevice::input`, `inputupdate()`, `IS_MOUSE`, `IS_RGB`, `keyboard`, `usbinput::keys`, `MSG_SIZE`, `usbdevice::product`, `usbinput::rel_x`, `usbinput::rel_y`, and `usbdevice::vendor`.

Referenced by `_setupusb()`.

```

226                                     {
227     usbdevice* kb = context;
228     int fd = kb->handle - 1;
229     short vendor = kb->vendor, product = kb->product;
230     int index = INDEX_OF(kb, keyboard);
231     ckb_info("Starting input thread for %s%d\n", devpath, index);
232
233     int urbcount = IS_RGB(vendor, product) ? (kb->epcount - 1) : kb->
234     epcount;
235
236     struct usbdevfs_urb urbs[urbcount];
237     memset(urbs, 0, sizeof(urbs));
238
239     urbs[0].buffer_length = 8;
240     if(IS_RGB(vendor, product)){
241         if(IS_MOUSE(vendor, product))
242             urbs[1].buffer_length = 10;
243         else
244             urbs[1].buffer_length = 21;
245         urbs[2].buffer_length = MSG_SIZE;
246         if(urbcount != 3)
247             urbs[urbcount - 1].buffer_length = MSG_SIZE;
248     } else {
249         urbs[1].buffer_length = 4;
250         urbs[2].buffer_length = 15;
251     }
252
253     for(int i = 0; i < urbcount; i++){
254         urbs[i].type = USBDEVFS_URB_TYPE_INTERRUPT;
255         urbs[i].endpoint = 0x80 | (i + 1);
256         urbs[i].buffer = malloc(urbs[i].buffer_length);
257         int rv = ioctl(fd, USBDEVFS_SUBMITURB, urbs + i);
258         if (rv < 0) ckb_warn("os_inputmain 1: got retcode %d and errno = %d, %s\n", rv, errno,
259 strerror(errno));
260     }
261
262     while (1) {
263         struct usbdevfs_urb* urb = 0;
264         int rv;
265
266         if ((rv = ioctl(fd, USBDEVFS_REAPURB, &urb)) {
267             ckb_warn("os_inputmain 2: got retcode %d and errno = %d, %s\n", rv, errno, strerror(
268 errno));
269             if (errno == ENODEV || errno == ENOENT || errno == ESHUTDOWN)
270                 // Stop the thread if the handle closes
271                 break;
272             else if(errno == EPIPE && urb){
273                 rv = ioctl(fd, USBDEVFS_CLEAR_HALT, &urb->endpoint);
274                 ckb_warn("os_inputmain 3: got retcode %d and errno = %d, %s\n", rv, errno, strerror
275 (errno));
276                 // Re-submit the URB
277                 if(urb)
278                     rv = ioctl(fd, USBDEVFS_SUBMITURB, urb);
279                 if (rv < 0) ckb_warn("os_inputmain 4: got retcode %d and errno = %d, %s\n", rv,
280 errno, strerror(errno));
281                 urb = 0;
282             }
283         }
284
285         if (urb) {
286             pthread_mutex_lock(&mutex(kb));
287             if(IS_MOUSE(vendor, product)){
288                 switch(urb->actual_length){
289                     case 8:
290                     case 10:
291                     case 11:
292                         // HID mouse input
293                         hid_mouse_translate(kb->input.keys, &kb->
294 input.rel_x, &kb->input.rel_y, -(urb->endpoint & 0xF), urb->actual_length, urb->buffer)
295 ;
296                         break;
297                     case MSG_SIZE:
298                         // Corsair mouse input
299                         corsair_mousecopy(kb->input.keys, -(urb->endpoint & 0xF), urb
300 ->buffer);
301                         break;
302                 }
303             } else if(IS_RGB(vendor, product)){
304                 switch(urb->actual_length){
305                     case 8:
306                         // RGB EP 1: 6KRO (BIOS mode) input
307                         hid_kb_translate(kb->input.keys, -1, urb->actual_length, urb->
308 buffer);
309                         break;
310                     case 21:
311                     case 5:
312                         // RGB EP 2: NKRO (non-BIOS) input. Accept only if keyboard is inactive

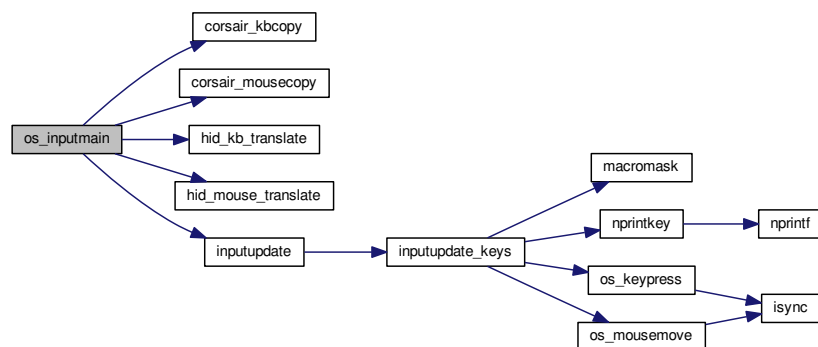
```

```

342         if (!kb->active)
343             hid_kb_translate(kb->input.keys, -2, urb->actual_length,
urb->buffer);
344         break;
345     case MSG_SIZE:
346         // RGB EP 3: Corsair input
347         corsair_kbcopy(kb->input.keys, -(urb->endpoint & 0xF), urb->
buffer);
348         break;
349     }
350 } else {
351     // Non-RGB input
352     hid_kb_translate(kb->input.keys, urb->endpoint & 0xF, urb->
actual_length, urb->buffer);
353 }
354 inputupdate(kb);
355 pthread_mutex_unlock(&mutex(kb));
356 rv = ioctl(fd, USBDEVFS_SUBMITURB, urb);
357 if (rv < 0) ckb_warn("os_inputmain 5: got retcode %d and errno = %d, %s\n", rv, errno,
strerror(errno));
358 urb = 0;
359 }
360 }
361
362 ckb_info("Stopping input thread for %s%d\n", devpath, index);
363 for(int i = 0; i < urbcount; i++){
364     ioctl(fd, USBDEVFS_DISCARDURB, urbs + i);
365     free(urbs[i].buffer);
366 }
367 return 0;
368 }

```

Here is the call graph for this function:



Here is the caller graph for this function:



5.38.3.8 os_resetusb()

```

int os_resetusb (
    usbdevice * kb,
    const char * file,
    int line )

```

Parameters

<i>kb</i>	THE usbdevice*
<i>file</i>	filename for error messages
<i>line</i>	line where it is called for error messages

Returns

Returns 0 on success, -2 if device should be removed and -1 if reset should be tried again

os_resetusb is the os specific implementation for resetting usb

Try to reset an usb device in a linux user space driver.

1. unclaim the device, but do not reconnect the system driver (second param resetting = true)
2. reset the device via USBDEVFS_RESET command
3. claim the device again. Returns 0 on success, -2 if device should be removed and -1 if reset should be tried again

Todo it seems that no one wants to try the reset again. But I've seen it somewhere...

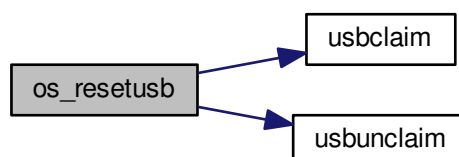
Definition at line 485 of file usb_linux.c.

References ckb_warn, usbdevice::handle, usbdevice::name, TEST_RESET, usbclaim(), and usbunclaim().

Referenced by _resetusb().

```
485                                     {
486     ckb_warn("os_resetusb: resetting %s\n", kb->name);
487     TEST_RESET(usbunclaim(kb, 1));
488     TEST_RESET(ioctl(kb->handle - 1, USBDEVFS_RESET));
489     TEST_RESET(usbclaim(kb));
490     // Success!
491     return 0;
492 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



5.38.3.9 os_sendindicators()

```
void os_sendindicators (
    usbdevice * kb )
```

Parameters

<i>kb</i>	THE usbdevice*
-----------	----------------

os_sendindicators update the indicators for the special keys (Numlock, Capslock and what else?)

os_sendindicators update the indicators for the special keys (Numlock, Capslock and what else?)

Read the data from kb->iLEDs and send them via ioctl() to the keyboard.

bRequest↔ Type	bRequest	wValue	EP	size	Timeout	data
0x21	0x09	0x0200	Interface 0	MSG_SIZE 1 Byte	timeout 0,5ms	the message buffer pointer
Host to Device, Type=Class, Recipient=Interface (why not end- point?)	9 = SEND?	specific	0	1	500	struct* kb- >iLEDs

The ioctl command is USBDEVFS_CONTROL.

Definition at line 206 of file usb_linux.c.

References ckb_err, usbdevice::handle, and usbdevice::iLEDs.

Referenced by updateindicators_kb().

```

206                                     {
207     struct usbdevfs_ctrltransfer transfer = { 0x21, 0x09, 0x0200, 0x00, 1, 500, &kb->
iLEDs };
208     int res = ioctl(kb->handle - 1, USBDEVFS_CONTROL, &transfer);
209     if(res <= 0)
210         ckb_err("%s\n", res ? strerror(errno) : "No data written");
211 }
```

Here is the caller graph for this function:



5.38.3.10 os_setupusb()

```
int os_setupusb (
    usbdevice * kb )
```

Parameters

<i>kb</i>	THE usbdevice*
-----------	----------------

Returns

0 on success, -1 otherwise.

os_setupusb OS-specific setup for a specific usb device.

Perform the operating system-specific opening of the interface in [os_setupusb\(\)](#). As a result, some parameters should be set in kb (name, serial, fwversion, epcount = number of usb endpoints), and all endpoints should be claimed with [usbclaim\(\)](#). Claiming is the only point where [os_setupusb\(\)](#) can produce an error (-1).

- Copy device description and serial
- Copy firmware version (needed to determine USB protocol)
- Do some output about connecting interfaces
- Claim the USB interfaces

Todo in these modules a pullrequest is outstanding

Definition at line 524 of file usb_linux.c.

References [ckb_err](#), [ckb_info](#), [ckb_warn](#), [devpath](#), [usbdevice::epcount](#), [FEAT_RGB](#), [usbdevice::fwversion](#), [HAS_↔FEATURES](#), [INDEX_OF](#), [KB_NAME_LEN](#), [keyboard](#), [usbdevice::name](#), [usbdevice::serial](#), [SERIAL_LEN](#), [strtrim\(\)](#), [usbdevice::udev](#), and [usbclaim\(\)](#).

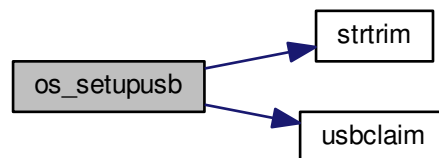
Referenced by [_setupusb\(\)](#).

```

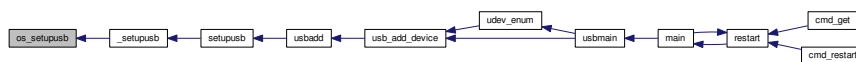
524     {
527     struct udev_device* dev = kb->udev;
528     const char* name = udev_device_get_sysattr_value(dev, "product");
529     if(name)
530         strncpy(kb->name, name, KB_NAME_LEN);
531     strtrim(kb->name);
532     const char* serial = udev_device_get_sysattr_value(dev, "serial");
533     if(serial)
534         strncpy(kb->serial, serial, SERIAL_LEN);
535     strtrim(kb->serial);
538     const char* firmware = udev_device_get_sysattr_value(dev, "bcdDevice");
539     if(firmware)
540         sscanf(firmware, "%hx", &kb->fwversion);
541     else
542         kb->fwversion = 0;
543     int index = INDEX_OF(kb, keyboard);
546     ckb_info("Connecting %s at %s%d\n", kb->name, devpath, index);
547
553     const char* ep_str = udev_device_get_sysattr_value(dev, "bNumInterfaces");
554     kb->epcount = 0;
555     if(ep_str)
556         sscanf(ep_str, "%d", &kb->epcount);
557     if(kb->epcount == 0){
558         // This shouldn't happen, but if it does, assume EP count based on what the device is supposed to
559         have
560         kb->epcount = (HAS_FEATURES(kb, FEAT_RGB) ? 4 : 3);
560         ckb_warn("Unable to read endpoint count from udev, assuming %d...\n", kb->
epcount);
561     }
562     if(usbclaim(kb)){
563         ckb_err("Failed to claim interfaces: %s\n", strerror(errno));
564         return -1;
565     }
566     return 0;
567 }

```

Here is the call graph for this function:



Here is the caller graph for this function:



5.38.3.11 os_usbrecv()

```

int os_usbrecv (
    usbdevice * kb,
    uchar * in_msg,
    const char * file,
    int line )

```

Parameters

<i>kb</i>	THE usbdevice*
<i>in_msg</i>	the buffer to fill with the message received
<i>file</i>	for debugging
<i>line</i>	for debugging

Returns

-1 on timeout, 0 on hard error, number of bytes received otherwise

os_usbrecv does what its name says:

The comment at the beginning of the procedure causes the suspicion that the firmware versionspecific distinction is missing for receiving from usb endpoint 3 or 4. The commented code contains only the reception from EP4, but this may be wrong for a software version 2.0 or higher (see the code for os-usb send ()).

So all the receiving is done via an ioctl() like in os_usb send. The ioctl() is given a struct usbdevfs_ctrltransfer, in which the relevant parameters are entered:

bRequest↔ Type	bRequest	wValue	EP	size	Timeout	data
0xA1	0x01	0x0200	endpoint to be addressed from epcount - 1	MSG_SIZE	5ms	the message buffer pointer
Device to Host, Type=Class, Recipient=Interface	1 = RECEIVE?	specific	Interface #	64	5000	in_msg

The ioctl() returns the number of bytes received. Here is the usual check again:

- If the return value is -1 AND the error is a timeout (ETIMEDOUT), [os_usbrecv\(\)](#) will return -1 to indicate that it is probably a recoverable problem and a retry is recommended.
- For another negative value or other error identifier OR 0 bytes are received, 0 is returned as an identifier for a heavy error.
- In all other cases, the function returns the number of bytes received.

If this is not the entire blocksize (MSG_SIZE bytes), an error message is issued on the standard error channel [warning "Read YY bytes (expected 64)"].

Definition at line 127 of file usb_linux.c.

References ckb_err_fn, ckb_warn_fn, usbdevice::epcount, usbdevice::handle, and MSG_SIZE.

Referenced by _usbrecv().

```

127                                     {
128     int res;
129     // This is what CUE does, but it doesn't seem to work on linux.
130     /*if(kb->fwversion >= 0x130){
131         struct usbdevfs_bulktransfer transfer;
132         memset(&transfer, 0, sizeof(transfer));
133         transfer.ep = 0x84;
134         transfer.len = MSG_SIZE;
135         transfer.timeout = 5000;
136         transfer.data = in_msg;
137         res = ioctl(kb->handle - 1, USBDEVFS_BULK, &transfer);
138     } else {*/
139     struct usbdevfs_ctltransfer transfer = { 0xa1, 0x01, 0x0300, kb->
140     epcount - 1, MSG_SIZE, 5000, in_msg };
141     res = ioctl(kb->handle - 1, USBDEVFS_CONTROL, &transfer);
142     //}
143     if(res <= 0){
144         ckb_err_fn("%s\n", file, line, res ? strerror(errno) : "No data read");
145         if(res == -1 && errno == ETIMEDOUT)
146             return -1;
147         else
148             return 0;
149     } else if(res != MSG_SIZE)
150         ckb_warn_fn("Read %d bytes (expected %d)\n", file, line, res,
151         MSG_SIZE);
152     return res;
153 }

```

Here is the caller graph for this function:



5.38.3.12 os_usbrecv()

```

int os_usbrecv (
    usbdevice * kb,
    const uchar * out_msg,
    int is_recv,
    const char * file,
    int line )

```

Parameters

<i>kb</i>	THE usbdevice*
<i>out_msg</i>	the MSGSIZE char long buffer to send
<i>is_recv</i>	if true, just send an ioctl for further reading packets. If false, send the data at out_msg .
<i>file</i>	for debugging
<i>line</i>	for debugging

Returns

-1 on timeout (try again), 0 on hard error, number of bytes sent otherwise

os_usbseend has two functions:

- if is_recv == false, it tries to send a given MSG_SIZE buffer via the usb interface given with kb.
- otherwise a request is sent via the usb device to initiate the receiving of a message from the remote device.

The functionality for sending distinguishes two cases, depending on the version number of the firmware of the connected device:

If the firmware is less or equal 1.2, the transmission is done via an ioctl(). The ioctl() is given a struct usbdevfs_↔ctrltransfer, in which the relevant parameters are entered:

bRequest↔ Type	bRequest	wValue	EP	size	Timeout	data
0x21	0x09	0x0200	endpoint / IF to be addressed from epcount-1	MSG_SIZE	5000 (=5ms)	the message buffer pointer
Host to Device, Type=Class, Recipient=Interface	9 = Send data?	specific	last or pre-last device #	64	5000	out_msg

The ioctl command is USBDEVFS_CONTROL.

The same constellation is used if the device is requested to send its data (is_recv = true).

For a more recent firmware and is_recv = false, the ioctl command USBDEVFS_CONTROL is not used (this tells the bus to enter the control mode), but the bulk method is used: USBDEVFS_BULK. This is astonishing, because all of the endpoints are type Interrupt, not bulk.

Anyhow, for this purpose a different structure is used for the ioctl() (struct **usbdevfs_bulktransfer**) and this is also initialized differently:

The length and timeout parameters are given the same values as above. The formal parameter out_msg is also passed as a buffer pointer. For the endpoints, the firmware version is differentiated again:

For a firmware version between 1.3 and <2.0 endpoint 4 is used, otherwise (it can only be ≥2.0) endpoint 3 is used.

Todo Since the handling of endpoints has already led to problems elsewhere, this implementation is extremely hardware-dependent and critical!

Eg. the new keyboard K95PLATINUMRGB has a version number significantly less than 2.0 - will it run with this implementation?

The ioctl() - no matter what type - returns the number of bytes sent. Now comes the usual check:

- If the return value is -1 AND the error is a timeout (ETIMEDOUT), **os_usbseend()** will return -1 to indicate that it is probably a recoverable problem and a retry is recommended.
- For another negative value or other error identifier OR 0 bytes sent, 0 is returned as a heavy error identifier.

- In all other cases, the function returns the number of bytes sent.

If this is not the entire blocksize (MSG_SIZE bytes), an error message is issued on the standard error channel [warning "Wrote YY bytes (expected 64)"].

If DEBUG_USB is set during compilation, the number of bytes sent and their representation are logged to the error channel.

Definition at line 66 of file usb_linux.c.

References `ckb_err_fn`, `ckb_warn_fn`, `usbdevice::epcount`, `usbdevice::fwversion`, `usbdevice::handle`, and `MSG_SIZE`.

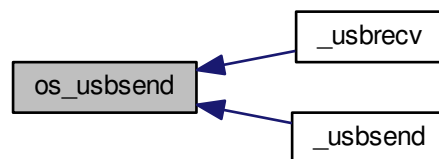
Referenced by `_usbrecv()`, and `_usbsend()`.

```

66                                     {
67     int res;
68     if(kb->fwversion >= 0x120 && !is_recv){
69         struct usbdevfs_bulktransfer transfer;
70         memset(&transfer, 0, sizeof(transfer));
71         transfer.ep = (kb->fwversion >= 0x130 && kb->fwversion < 0x200) ? 4 : 3;
72         transfer.len = MSG_SIZE;
73         transfer.timeout = 5000;
74         transfer.data = (void*)out_msg;
75         res = ioctl(kb->handle - 1, USBDEVFS_BULK, &transfer);
76     } else {
77         struct usbdevfs_ctrltransfer transfer = { 0x21, 0x09, 0x0200, kb->
78         epcount - 1, MSG_SIZE, 5000, (void*)out_msg };
79         res = ioctl(kb->handle - 1, USBDEVFS_CONTROL, &transfer);
80     }
81     if(res <= 0){
82         ckb_err_fn("%s\n", file, line, res ? strerror(errno) : "No data written");
83         if(res == -1 && errno == ETIMEDOUT)
84             return -1;
85         else
86             return 0;
87     } else if(res != MSG_SIZE)
88         ckb_warn_fn("Wrote %d bytes (expected %d)\n", file, line, res,
89         MSG_SIZE);
90 #ifdef DEBUG_USB
91     char converted[MSG_SIZE*3 + 1];
92     for(int i=0;i<MSG_SIZE;i++)
93         sprintf(&converted[i*3], "%02x ", out_msg[i]);
94     ckb_warn_fn("Sent %s\n", file, line, converted);
95 #endif
96     return res;
97 }

```

Here is the caller graph for this function:



5.38.3.13 product_str()

```

const char* product_str (
    short product )

```

Parameters

<i>product</i>	is the <i>short</i> USB device product ID
----------------	---

Returns

string to identify a type of device (see below)

`product_str` returns a condensed view on what type of device we have.

At present, various models and their properties are known from corsair products. Some models differ in principle (mice and keyboards), others differ in the way they function (for example, RGB and non RGB), but they are very similar.

Here, only the first point is taken into consideration and we return a unified model string. If the model is not known with its number, `product_str` returns an empty string.

The model numbers and corresponding strings with the numbers in hex-string are defined in [usb.h](#)

At present, this function is used to initialize `kb->name` and to give information in debug strings.

Attention

The combinations below have to fit to the combinations in the macros mentioned above. So if you add a device with a new number, change both.

Todo There are macros defined in [usb.h](#) to detect all the combinations below. the only difference is the parameter: The macros need the `kb*`, `product_str()` needs the *product ID*

Definition at line 70 of file `usb.c`.

References `P_HARPOON`, `P_K65`, `P_K65_LUX`, `P_K65_NRGB`, `P_K65_RFIRE`, `P_K70`, `P_K70_LUX`, `P_K70_LUX_NRGB`, `P_K70_NRGB`, `P_K70_RFIRE`, `P_K70_RFIRE_NRGB`, `P_K95`, `P_K95_NRGB`, `P_K95_PLATINUM`, `P_M65`, `P_M65_PRO`, `P_SABRE_L`, `P_SABRE_N`, `P_SABRE_O`, `P_SABRE_O2`, `P_SCIMITAR`, `P_SCIMITAR_PRO`, `P_STRAFE`, and `P_STRAFE_NRGB`.

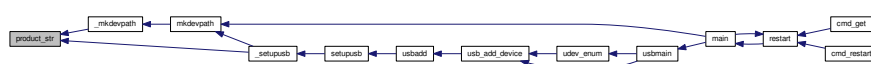
Referenced by `_mkdevpath()`, and `_setupusb()`.

```

70     {
71         if (product == P_K95 || product == P_K95_NRGB || product ==
P_K95_PLATINUM)
72             return "k95";
73         if (product == P_K70 || product == P_K70_NRGB || product ==
P_K70_LUX || product == P_K70_LUX_NRGB || product ==
P_K70_RFIRE || product == P_K70_RFIRE_NRGB)
74             return "k70";
75         if (product == P_K65 || product == P_K65_NRGB || product ==
P_K65_LUX || product == P_K65_RFIRE)
76             return "k65";
77         if (product == P_STRAFE || product == P_STRAFE_NRGB)
78             return "strafe";
79         if (product == P_M65 || product == P_M65_PRO)
80             return "m65";
81         if (product == P_SABRE_O || product == P_SABRE_L || product ==
P_SABRE_N || product == P_SABRE_O2 || product == P_HARPOON)
82             return "sabre";
83         if (product == P_SCIMITAR || product == P_SCIMITAR_PRO)
84             return "scimitar";
85         return "";
86     }

```

Here is the caller graph for this function:



5.38.3.14 revertusb()

```
int revertusb (
    usbdevice * kb )
```

Parameters

<i>kb</i>	THE usbdevice*
-----------	----------------

Returns

0 on success or if device needs firmware upgrade, -1 otherwise

revertusb sets a given device to inactive (hardware controlled) mode if not a fw-upgrade is indicated

First is checked, whether a firmware-upgrade is indicated for the device. If so, [revertusb\(\)](#) returns 0.

Todo Why is this useful? Are there problems seen with deactivating a device with older fw-version??? Why isn't this an error indicating reason and we return success (0)?

Anyway, the following steps are similar to some other procs, dealing with low level usb handling:

- If we do not have an RGB device, a simple setting to Hardware-mode (NK95_HWON) is sent to the device via `nk95cmd()`.

Todo The return value of `nk95cmd()` is ignored (but sending the ioctl may produce an error and `_nk95_cmd` will indicate this), instead `revertusb()` returns success in any case.

- If we have an RGB device, `setactive()` is called with second param `active = false`. That function will have a look on differences between keyboards and mice.
More precisely `setactive()` is just a macro to call via the `kb->vtable` entries either the `active()` or the `idle()` function where the `vtable` points to. `setactive()` may return error indications. If so, `revertusb()` returns -1, otherwise 0 in any other case.

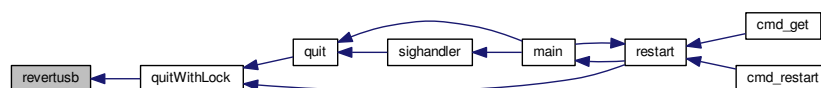
Definition at line 410 of file `usb.c`.

References `FEAT_RGB`, `HAS_FEATURES`, `NEEDS_FW_UPDATE`, `NK95_HWON`, `nk95cmd`, and `setactive`.

Referenced by `quitWithLock()`.

```
410     {
411         if (NEEDS_FW_UPDATE(kb))
412             return 0;
413         if (!HAS_FEATURES(kb, FEAT_RGB)) {
414             nk95cmd(kb, NK95_HWON);
415             return 0;
416         }
417         if (setactive(kb, 0))
418             return -1;
419         return 0;
420     }
```

Here is the caller graph for this function:



5.38.3.15 setupusb()

```
void setupusb (
    usbdevice * kb )
```

Attention

Lock a device's dmutex (see [device.h](#)) before accessing the USB interface.

Parameters

<i>kb</i>	THE usbdevice* used everywhere
<i>[O↔ UT]</i>	kb->thread is used to store the thread ID of the fresh created thread.

setupusb starts a thread with kb as parameter and [_setupusb\(\)](#) as entrypoint.

Set up a USB device after its handle is open. Spawns a new thread [_setupusb\(\)](#) with standard parameter kb. dmutex must be locked prior to calling this function. The function will unlock it when finished. In kb->thread the thread id is mentioned, because [closeusb\(\)](#) needs this info for joining that thread again.

Definition at line 389 of file usb.c.

References [_setupusb\(\)](#), [ckb_err](#), [imutex](#), and [usbdevice::thread](#).

Referenced by [usbadd\(\)](#).

```
389         {
390     pthread_mutex_lock(&imutex(kb));
391     if(pthread_create(&kb->thread, 0, _setupusb, kb))
392         ckb_err("Failed to create USB thread\n");
393 }
```


Returns

0 on success, -1 otherwise

usb_tryreset does what the name means: Try to reset the usb via [resetusb\(\)](#)

This function is called if an usb command ran into an error in case of one of the following two situations:

- When setting up a new usb device and the start() function got an error (

See also

[_setupusb\(\)](#)

- If upgrading to a new firmware gets an error (

See also

[cmd_fwupdate\(\)](#)).

The previous action which got the error will NOT be re-attempted.

In an endless loop [usb_tryreset\(\)](#) tries to reset the given usb device via the macro [resetusb\(\)](#).

This macro calls [_resetusb\(\)](#) with debugging information.

[_resetusb\(\)](#) sends a command via the operating system dependent function [os_resetusb\(\)](#) and - if successful - reinitializes the device. [os_resetusb\(\)](#) returns -2 to indicate a broken device and all structures should be removed for it.

In that case, the loop is terminated, an error message is produced and [usb_tryreset\(\)](#) returns -1.

In case [resetusb\(\)](#) has success, the endless loop is left via a return 0 (success).

If the return value from [resetusb\(\)](#) is -1, the loop is continued with the next try.

If the global variable **reset_stop** is set directly when the function is called or after each try, [usb_tryreset\(\)](#) stops working and returns -1.

Todo Why does [usb_tryreset\(\)](#) hide the information returned from [resetusb\(\)](#)? Isn't it needed by the callers?

Definition at line 468 of file usb.c.

References [ckb_err](#), [ckb_info](#), [hwload_mode](#), [reset_stop](#), and [resetusb](#).

Referenced by [_setupusb\(\)](#), and [cmd_fwupdate\(\)](#).

```

468             {
469     if(reset_stop)
470         return -1;
471     ckb_info("Attempting reset...\n");
472     while(1){
473         int res = resetusb(kb);
474         if(!res){
475             ckb_info("Reset success\n");
476             return 0;
477         }
478         if(res == -2 || reset_stop)
479             break;
480     }
481     ckb_err("Reset failed. Disconnecting.\n");
482     return -1;
483 }
```

Here is the caller graph for this function:



5.38.3.17 usbkill()

```
void usbkill ( )
```

Stop the USB system.

Definition at line 815 of file usb_linux.c.

References `udev`.

Referenced by `quitWithLock()`.

```
815     {
816         udev_unref(udev);
817         udev = 0;
818     }
```

Here is the caller graph for this function:



5.38.3.18 usbmain()

```
int usbmain ( )
```

Start the USB main loop. Returns program exit code when finished.

`usbmain` is called by `main()` after setting up all other stuff.

Returns

0 normally or -1 if fatal error occurs (up to now only if no new devices are available)

First check whether the `uinput` module is loaded by the kernel.

Todo Why isn't missing of `uinput` a fatal error?

Create the `udev` object with `udev_new()` (is a function from `libudev.h`) terminate -1 if error

Enumerate all currently connected devices

Todo lae. here the work has to go on...

Definition at line 752 of file usb_linux.c.

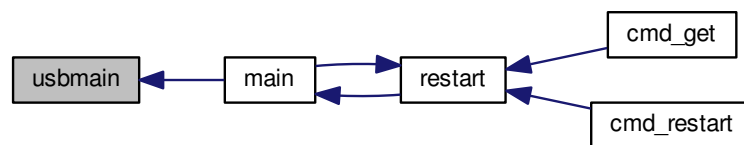
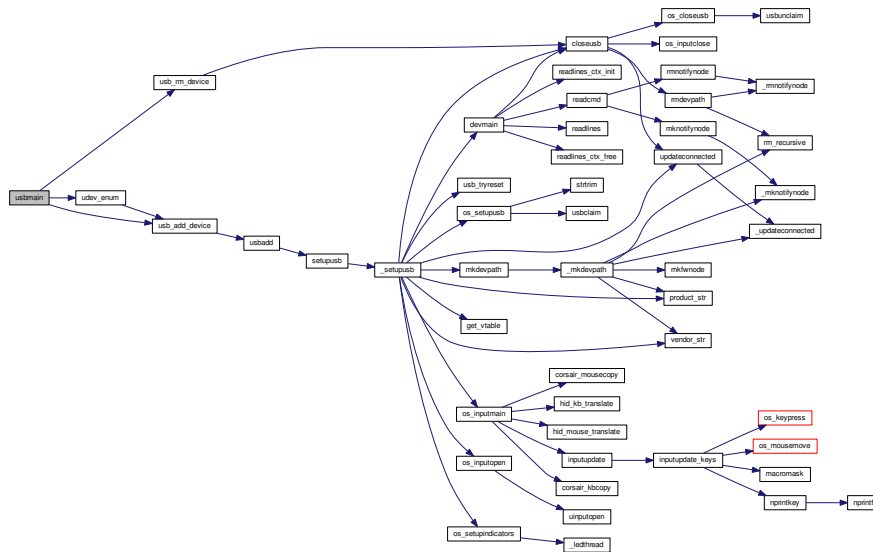
References `ckb_fatal`, `ckb_warn`, `udev`, `udev_enum()`, `usb_add_device()`, and `usb_rm_device()`.

Referenced by `main()`.

```

752     {
753     // Load the uinput module (if it's not loaded already)
754     if(system("modprobe uinput") != 0)
755         ckb_warn("Failed to load uinput module\n");
756
757     if(!(udev = udev_new())) {
758         ckb_fatal("Failed to initialize udev in usbmain(), usb_linux.c\n");
759         return -1;
760     }
761
762     udev_enum();
763
764     // Done scanning. Enter a loop to poll for device updates
765     struct udev_monitor* monitor = udev_monitor_new_from_netlink(udev, "udev");
766     udev_monitor_filter_add_match_subsystem_devtype(monitor, "usb", 0);
767     udev_monitor_enable_receiving(monitor);
768     // Get an fd for the monitor
769     int fd = udev_monitor_get_fd(monitor);
770     fd_set fds;
771     while(udev){
772         FD_ZERO(&fds);
773         FD_SET(fd, &fds);
774         // Block until an event is read
775         if(select(fd + 1, &fds, 0, 0, 0) > 0 && FD_ISSET(fd, &fds)){
776             struct udev_device* dev = udev_monitor_receive_device(monitor);
777             if(!dev)
778                 continue;
779             const char* action = udev_device_get_action(dev);
780             if(!action){
781                 udev_device_unref(dev);
782                 continue;
783             }
784             // Add/remove device
785             if(!strcmp(action, "add")){
786                 int res = usb_add_device(dev);
787                 if(res == 0)
788                     continue;
789                 // If the device matched but the handle wasn't opened correctly, re-enumerate (this
790                 // sometimes solves the problem)
791                 if(res == -1)
792                     udev_enum();
793             } else if(!strcmp(action, "remove")){
794                 usb_rm_device(dev);
795                 udev_device_unref(dev);
796             }
797         }
798         udev_monitor_unref(monitor);
799         return 0;
800     }
801 }

```



```
const char* vendor_str (
    short vendor )
```

Parameters

<i>vendor</i>	<i>short</i> vendor ID
---------------	------------------------

a string: either "" or "corsair"

uncomment the following Define to see USB packets sent to the device

vendor_str returns "corsair" iff the given *vendor* argument is equal to *V_CORSAIR* (0x1bc) else it returns ""

Attention

There is also a string defined *V_CORSAIR_STR*, which returns the device number as string in hex "1b1c".

Definition at line 43 of file usb.c.

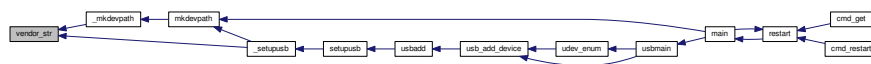
References *V_CORSAIR*.

Referenced by *_mkdevpath()*, and *_setupusb()*.

```

43                                     {
44     if (vendor == V_CORSAIR)
45         return "corsair";
46     return "";
47 }
```

Here is the caller graph for this function:



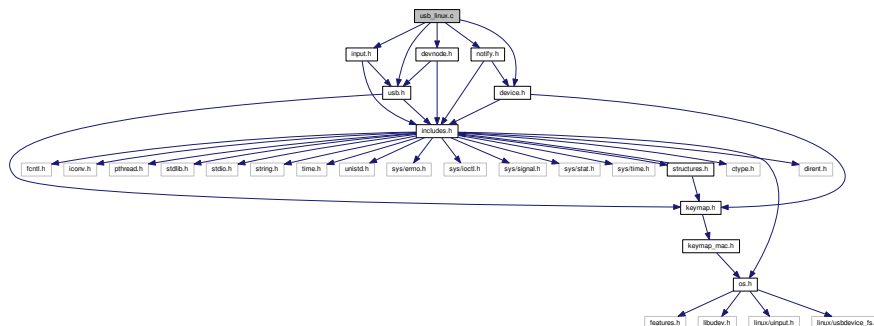
5.39 usb_linux.c File Reference

```

#include "device.h"
#include "devnode.h"
#include "input.h"
#include "notify.h"
#include "usb.h"

```

Include dependency graph for usb_linux.c:



Data Structures

- [struct _model](#)

typedef struct _model Structure to map string representation of vendor / product and its short number representation-
More...

Macros

- #define `TEST_RESET`(op)
TEST_RESET does a "try / catch" for resetting the usb interface.
- #define `N_MODELS` (sizeof(`models`) / sizeof(`_model`))

Functions

- int `os_usbsend` (usbdevice *kb, const uchar *out_msg, int is_recv, const char *file, int line)
os_usbsend sends a data packet (MSG_SIZE = 64) Bytes long
- int `os_usbrecv` (usbdevice *kb, uchar *in_msg, const char *file, int line)
os_usbrecv receives a max MSGSIZE long buffer from usb device
- int `_nk95cmd` (usbdevice *kb, uchar bRequest, ushort wValue, const char *file, int line)
_nk95cmd If we control a non RGB keyboard, set the keyboard via ioctl with usbdevfs_ctrltransfer
- void `os_sendindicators` (usbdevice *kb)
- void * `os_inputmain` (void *context)
os_inputmain This function is run in a separate thread and will be detached from the main thread, so it needs to clean up its own resources.
- static int `usbunclaim` (usbdevice *kb, int resetting)
- void `os_closeusb` (usbdevice *kb)
- static int `usbclaim` (usbdevice *kb)
- int `os_resetusb` (usbdevice *kb, const char *file, int line)
- void `strtrim` (char *string)
- int `os_setupusb` (usbdevice *kb)
- int `usbadd` (struct udev_device *dev, short vendor, short product)
- static int `usb_add_device` (struct udev_device *dev)
Add a udev device.
- static void `usb_rm_device` (struct udev_device *dev)
usb_rm_device find the usb port to remove and close it via `closeusb()`.
- static void `udev_enum` ()
udev_enum use the udev_enumerate_add_match_subsystem() to get all you need but only that.
- int `usbmain` ()
- void `usbkill` ()
usbkill unreference the udev structure via libudev and mark static pointer with 0 as not set.

Variables

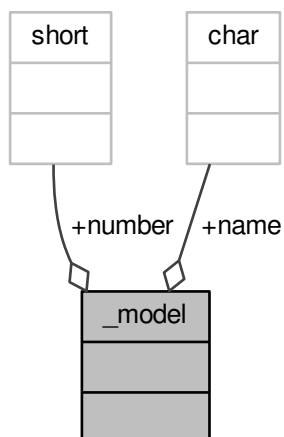
- static char `kbsyspath` [9][FILENAME_MAX]
all open usb devices have their system path names here in this array.
- static struct udev * `udev`
udev struct udev holds the information about the user space devices; used in `usbmain()` to monitor the usb changes.
- pthread_t `usbthread`
struct udev is defined in /usr/include/libudev.h
- pthread_t `udevthread`
- static `_model models` []

5.39.1 Data Structure Documentation

5.39.1.1 struct _model

Definition at line 620 of file usb_linux.c.

Collaboration diagram for _model:



Data Fields

const char *	name	
short	number	

5.39.2 Macro Definition Documentation

5.39.2.1 N_MODELS

```
#define N_MODELS (sizeof(models) / sizeof(_model))
```

Definition at line 659 of file usb_linux.c.

Referenced by usb_add_device().

5.39.2.2 TEST_RESET

```
#define TEST_RESET(
    op )
```

Value:

```
if(op) {
    ckb_err_fn("resetusb failed: %s\n", file, line, strerror(errno)); \
    if(errno == EINTR || errno == EAGAIN) \
        return -1; \
    return -2; \
    /* try again if status code says so */ \
    /* else, remove device */ \
}
```

Definition at line 467 of file usb_linux.c.

Referenced by os_resetusb().

5.39.3 Function Documentation

5.39.3.1 _nk95cmd()

```
int _nk95cmd (
    usbdevice * kb,
    uchar bRequest,
    ushort wValue,
    const char * file,
    int line )
```

To send control packets to a non RGB non color K95 Keyboard, use this function. Normally it is called via the `nk95cmd()` macro.

If it is the wrong device for which the function is called, 0 is returned and nothing done. Otherwise a `usbdevfs_↵` `ctrltransfer` structure is filled and an `USBDEVFS_CONTROL` `ioctl()` called.

bRequest↵ Type	bRequest	wValue	EP	size	Timeout	data
0x40	see table below to switch hardware-modus at Keyboard	wValue	device	MSG_SIZE	5ms	the message buffer pointer
Host to Device, Type=Vendor, Recipient=Device	bRequest parameter	given wValue Parameter	device 0	0 data to write	5000	null

If a 0 or a negative error number is returned by the `ioctl`, an error message is shown depending on the `errno` or "No data written" if `retval` was 0. In either case 1 is returned to indicate the error. If the `ioctl` returned a value > 0 , 0 is returned to indicate no error.

Currently the following combinations for bRequest and wValue are used:

Device	what it might to do	constant	bRequest	wValue
non RGB Keyboard	set HW-modus on (leave the ckb driver)	HWON	0x0002	0x0030
non RGB Keyboard	set HW-modus off (initialize the ckb driver)	HWOFF	0x0002	0x0001
non RGB Keyboard	set light modus M1 in single-color keyboards	NK95_M1	0x0014	0x0001
non RGB Keyboard	set light modus M2 in single-color keyboards	NK95_M2	0x0014	0x0002
non RGB Keyboard	set light modus M3 in single-color keyboards	NK95_M3	0x0014	0x0003

See also

[usb.h](#)

Definition at line 181 of file `usb_linux.c`.

References `ckb_err_fn`, `usbdevice::handle`, `P_K95_NRGB`, and `usbdevice::product`.

```

181                                     {
182     if(kb->product != P_K95_NRGB)
183         return 0;
184     struct usbdevfs_ctrltransfer transfer = { 0x40, bRequest, wValue, 0, 0, 5000, 0 };
185     int res = ioctl(kb->handle - 1, USBDEVFS_CONTROL, &transfer);
186     if(res <= 0){
187         ckb_err_fn("%s\n", file, line, res ? strerror(errno) : "No data written");
188         return 1;
189     }
190     return 0;
191 }
```

5.39.3.2 os_closeusb()

```

void os_closeusb (
    usbdevice * kb )
```

`os_closeusb` unclaim it, destroy the udev device and clear data structures at `kb`

`os_closeusb` is the linux specific implementation for closing an active usb port.

If a valid handle is given in the `kb` structure, the usb port is unclaimed (`usbunclaim()`).

The device is unrefenced via library function `udev_device_unref()`.

`handle`, `udev` and the first char of `kbsyspath` are cleared to 0 (empty string for `kbsyspath`).

Definition at line 423 of file `usb_linux.c`.

References `usbdevice::handle`, `INDEX_OF`, `kbsyspath`, `keyboard`, `usbdevice::udev`, and `usbunclaim()`.

Referenced by `closeusb()`.

```

423                                     {
424     if(kb->handle){
425         usbunclaim(kb, 0);
426         close(kb->handle - 1);
427     }
428     if(kb->udev)
429         udev_device_unref(kb->udev);
430     kb->handle = 0;
431     kb->udev = 0;
432     kbsyspath[INDEX_OF(kb, keyboard)] [0] = 0;
433 }
```


device	detect with macro combination	endpoint #	buffer-length
each	none	0	8
RGB Mouse	IS_RGB && IS_MOUSE	1	10
RGB Keyboard	IS_RGB && !IS_MOUSE	1	21
RGB Mouse or Keyboard	IS_RGB	2	MSG_SIZE (64)
non RGB Mouse or Keyboard	!IS_RGB	1	4
non RGB Mouse or Keyboard	!IS_RGB	2	15

Now submit all the URBs via `ioctl(USBDEVFS_SUBMITURB)` with type `USBDEVFS_URB_TYPE_INTERRUPT` (the endpoints are defined as type interrupt). Endpoint number is 0x80..0x82 or 0x83, depending on the model.

The userSpaceFS knows the URBs now, so start monitoring input

if the `ioctl` returns something `!= 0`, let's have a deeper look what happened. Broken devices or shutting down the entire system leads to closing the device and finishing this thread.

If just an EPIPE occurred, give the device a `CLEAR_HALT` and resubmit the URB.

A correct `REAPURB` returns a Pointer to the URB which we now have a closer look into. Lock all following actions with `imutex`.

Process the input depending on type of device. Interpret the actual size of the URB buffer

device	detect with macro combination	seems to be endpoint #	actual buffer-length	function called
mouse (RGB and non RGB)	IS_MOUSE	nA	8, 10 or 11	<code>hid_mouse_↔translate()</code>
mouse (RGB and non RGB)	IS_MOUSE	nA	MSG_SIZE (64)	<code>corsair_↔mousecopy()</code>
RGB Keyboard	IS_RGB && !IS_↔MOUSE	1	8 (BIOS Mode)	<code>hid_kb_translate()</code>
RGB Keyboard	IS_RGB && !IS_↔MOUSE	2	5 or 21, KB inactive!	<code>hid_kb_translate()</code>
RGB Keyboard	IS_RGB && !IS_↔MOUSE	3?	MSG_SIZE	<code>corsair_kbcopy()</code>
non RGB Keyboard	!IS_RGB && !IS_↔MOUSE	nA	nA	<code>hid_kb_translate()</code>

The input data is transformed and copied to the kb structure. Now give it to the OS and unlock the `imutex` afterwards.

Re-submit the URB for the next run.

If the endless loop is terminated, clean up by discarding the URBs via `ioctl(USBDEVFS_DISCARDURB)`, free the URB buffers and return a null pointer as thread exit code.

Definition at line 226 of file `usb_linux.c`.

References `usbdevice::active`, `ckb_info`, `ckb_warn`, `corsair_kbcopy()`, `corsair_mousecopy()`, `devpath`, `usbdevice_↔::epcount`, `usbdevice::handle`, `hid_kb_translate()`, `hid_mouse_translate()`, `imutex`, `INDEX_OF`, `usbdevice::input`, `inputupdate()`, `IS_MOUSE`, `IS_RGB`, `keyboard`, `usbinput::keys`, `MSG_SIZE`, `usbdevice::product`, `usbinput::rel_↔x`, `usbinput::rel_y`, and `usbdevice::vendor`.

Referenced by `_setupusb()`.

```

226                                     {
227     usbdevice* kb = context;
228     int fd = kb->handle - 1;
229     short vendor = kb->vendor, product = kb->product;
230     int index = INDEX_OF(kb, keyboard);
231     ckb_info("Starting input thread for %s%d\n", devpath, index);
232
233     int urbcount = IS_RGB(vendor, product) ? (kb->epcount - 1) : kb->
234     epcount;
235
236     struct usbdevfs_urb urbs[urbcount];
237     memset(urbs, 0, sizeof(urbs));
238
239     urbs[0].buffer_length = 8;
240     if(IS_RGB(vendor, product)){
241         if(IS_MOUSE(vendor, product))
242             urbs[1].buffer_length = 10;
243         else
244             urbs[1].buffer_length = 21;
245         urbs[2].buffer_length = MSG_SIZE;
246         if(urbcount != 3)
247             urbs[urbcount - 1].buffer_length = MSG_SIZE;
248     } else {
249         urbs[1].buffer_length = 4;
250         urbs[2].buffer_length = 15;
251     }
252
253     for(int i = 0; i < urbcount; i++){
254         urbs[i].type = USBDEVFS_URB_TYPE_INTERRUPT;
255         urbs[i].endpoint = 0x80 | (i + 1);
256         urbs[i].buffer = malloc(urbs[i].buffer_length);
257         int rv = ioctl(fd, USBDEVFS_SUBMITURB, urbs + i);
258         if (rv < 0) ckb_warn("os_inputmain 1: got retcode %d and errno = %d, %s\n", rv, errno,
259 strerror(errno));
260     }
261
262     while (1) {
263         struct usbdevfs_urb* urb = 0;
264         int rv;
265
266         if ((rv = ioctl(fd, USBDEVFS_REAPURB, &urb)) {
267             ckb_warn("os_inputmain 2: got retcode %d and errno = %d, %s\n", rv, errno, strerror(
268 errno));
269             if (errno == ENODEV || errno == ENOENT || errno == ESHUTDOWN)
270                 // Stop the thread if the handle closes
271                 break;
272             else if(errno == EPIPE && urb){
273                 rv = ioctl(fd, USBDEVFS_CLEAR_HALT, &urb->endpoint);
274                 ckb_warn("os_inputmain 3: got retcode %d and errno = %d, %s\n", rv, errno, strerror
275 (errno));
276                 // Re-submit the URB
277                 if(urb)
278                     rv = ioctl(fd, USBDEVFS_SUBMITURB, urb);
279                 if (rv < 0) ckb_warn("os_inputmain 4: got retcode %d and errno = %d, %s\n", rv,
280 errno, strerror(errno));
281                 urb = 0;
282             }
283         }
284
285         if (urb) {
286             pthread_mutex_lock(&mutex(kb));
287             if(IS_MOUSE(vendor, product)){
288                 switch(urb->actual_length){
289                     case 8:
290                     case 10:
291                     case 11:
292                         // HID mouse input
293                         hid_mouse_translate(kb->input.keys, &kb->
294 input.rel_x, &kb->input.rel_y, -(urb->endpoint & 0xF), urb->actual_length, urb->buffer)
295 ;
296                         break;
297                     case MSG_SIZE:
298                         // Corsair mouse input
299                         corsair_mousecopy(kb->input.keys, -(urb->endpoint & 0xF), urb
300 ->buffer);
301                         break;
302                 }
303             } else if(IS_RGB(vendor, product)){
304                 switch(urb->actual_length){
305                     case 8:
306                         // RGB EP 1: 6KRO (BIOS mode) input
307                         hid_kb_translate(kb->input.keys, -1, urb->actual_length, urb->
308 buffer);
309                         break;
310                     case 21:
311                     case 5:
312                         // RGB EP 2: NKRO (non-BIOS) input. Accept only if keyboard is inactive

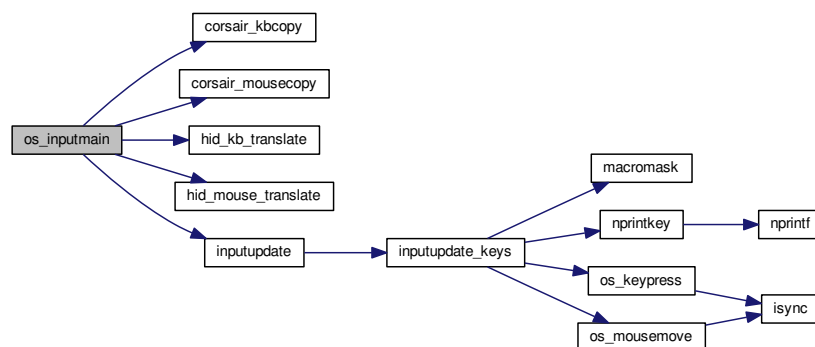
```

```

342         if (!kb->active)
343             hid_kb_translate(kb->input.keys, -2, urb->actual_length,
urb->buffer);
344         break;
345     case MSG_SIZE:
346         // RGB EP 3: Corsair input
347         corsair_kbcopy(kb->input.keys, -(urb->endpoint & 0xF), urb->
buffer);
348         break;
349     }
350 } else {
351     // Non-RGB input
352     hid_kb_translate(kb->input.keys, urb->endpoint & 0xF, urb->
actual_length, urb->buffer);
353 }
354 inputupdate(kb);
355 pthread_mutex_unlock(&imutex(kb));
356 rv = ioctl(fd, USBDEVFS_SUBMITURB, urb);
357 if (rv < 0) ckb_warn("os_inputmain 5: got retcode %d and errno = %d, %s\n", rv, errno,
strerror(errno));
358 urb = 0;
359 }
360 }
361
362 ckb_info("Stopping input thread for %s%d\n", devpath, index);
363 for(int i = 0; i < urbcount; i++){
364     ioctl(fd, USBDEVFS_DISCARDURB, urbs + i);
365     free(urbs[i].buffer);
366 }
367 return 0;
368 }

```

Here is the call graph for this function:



Here is the caller graph for this function:



5.39.3.4 os_resetusb()

```

int os_resetusb (
    usbdevice * kb,

```

```
const char * file,  
int line )
```

os_resetusb is the os specific implementation for resetting usb

Try to reset an usb device in a linux user space driver.

1. unclaim the device, but do not reconnect the system driver (second param resetting = true)
2. reset the device via USBDEVFS_RESET command
3. claim the device again. Returns 0 on success, -2 if device should be removed and -1 if reset should be tried again

Todo it seems that no one wants to try the reset again. But I've seen it somewhere...

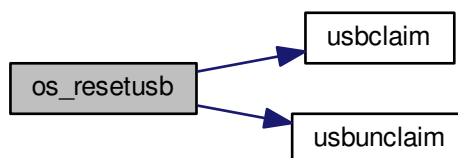
Definition at line 485 of file usb_linux.c.

References `ckb_warn`, `usbdevice::handle`, `usbdevice::name`, `TEST_RESET`, `usbclaim()`, and `usbunclaim()`.

Referenced by `_resetusb()`.

```
485  
486     ckb_warn("os_resetusb: resetting %s\n", kb->name);  
487     TEST_RESET(usbunclaim(kb, 1));  
488     TEST_RESET(ioctl(kb->handle - 1, USBDEVFS_RESET));  
489     TEST_RESET(usbclaim(kb));  
490     // Success!  
491     return 0;  
492 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



5.39.3.5 os_sendindicators()

```
void os_sendindicators (
    usbdevice * kb )
```

os_sendindicators update the indicators for the special keys (Numlock, Capslock and what else?)

os_sendindicators update the indicators for the special keys (Numlock, Capslock and what else?)

Read the data from kb->ileds ans send them via ioctl() to the keyboard.

bRequest↔ Type	bRequest	wValue	EP	size	Timeout	data
0x21	0x09	0x0200	Interface 0	MSG_SIZE 1 Byte	timeout 0,5ms	the message buffer pointer
Host to Device, Type=Class, Recipi- ent=Interface (why not end- point?)	9 = SEND?	specific	0	1	500	struct* kb- >ileds

The ioctl command is USBDEVFS_CONTROL.

Definition at line 206 of file usb_linux.c.

References ckb_err, usbdevice::handle, and usbdevice::ileds.

Referenced by updateindicators_kb().

```
206                                     {
207     struct usbdevfs_ctrltransfer transfer = { 0x21, 0x09, 0x0200, 0x00, 1, 500, &kb->
ileds };
208     int res = ioctl(kb->handle - 1, USBDEVFS_CONTROL, &transfer);
209     if(res <= 0)
210         ckb_err("%s\n", res ? strerror(errno) : "No data written");
211 }
```

Here is the caller graph for this function:



5.39.3.6 os_setupusb()

```
int os_setupusb (
    usbdevice * kb )
```

os_setupusb OS-specific setup for a specific usb device.

Perform the operating system-specific opening of the interface in [os_setupusb\(\)](#). As a result, some parameters should be set in kb (name, serial, fwversion, epcount = number of usb endpoints), and all endpoints should be claimed with [usbclaim\(\)](#). Claiming is the only point where [os_setupusb\(\)](#) can produce an error (-1).

- Copy device description and serial
- Copy firmware version (needed to determine USB protocol)
- Do some output about connecting interfaces
- Claim the USB interfaces

Todo in these modules a pullrequest is outstanding

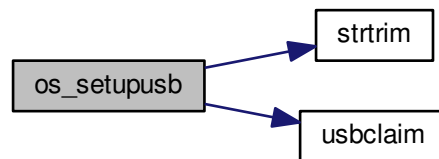
Definition at line 524 of file usb_linux.c.

References [ckb_err](#), [ckb_info](#), [ckb_warn](#), [devpath](#), [usbdevice::epcount](#), [FEAT_RGB](#), [usbdevice::fwversion](#), [HAS_FEATURES](#), [INDEX_OF](#), [KB_NAME_LEN](#), [keyboard](#), [usbdevice::name](#), [usbdevice::serial](#), [SERIAL_LEN](#), [strtrim\(\)](#), [usbdevice::udev](#), and [usbclaim\(\)](#).

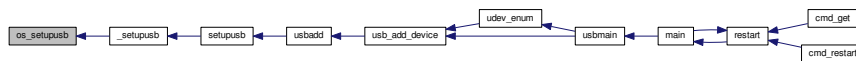
Referenced by [_setupusb\(\)](#).

```
524         {
527     struct udev_device* dev = kb->udev;
528     const char* name = udev_device_get_sysattr_value(dev, "product");
529     if(name)
530         strncpy(kb->name, name, KB_NAME_LEN);
531     strtrim(kb->name);
532     const char* serial = udev_device_get_sysattr_value(dev, "serial");
533     if(serial)
534         strncpy(kb->serial, serial, SERIAL_LEN);
535     strtrim(kb->serial);
538     const char* firmware = udev_device_get_sysattr_value(dev, "bcdDevice");
539     if(firmware)
540         sscanf(firmware, "%hx", &kb->fwversion);
541     else
542         kb->fwversion = 0;
543     int index = INDEX_OF(kb, keyboard);
546     ckb_info("Connecting %s at %s%d\n", kb->name, devpath, index);
547
553     const char* ep_str = udev_device_get_sysattr_value(dev, "bNumInterfaces");
554     kb->epcount = 0;
555     if(ep_str)
556         sscanf(ep_str, "%d", &kb->epcount);
557     if(kb->epcount == 0){
558         // This shouldn't happen, but if it does, assume EP count based on what the device is supposed to
have
559         kb->epcount = (HAS_FEATURES(kb, FEAT_RGB) ? 4 : 3);
560         ckb_warn("Unable to read endpoint count from udev, assuming %d...\n", kb->
epcount);
561     }
562     if(usbclaim(kb)){
563         ckb_err("Failed to claim interfaces: %s\n", strerror(errno));
564         return -1;
565     }
566     return 0;
567 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



5.39.3.7 os_usbrecv()

```

int os_usbrecv (
    usbdevice * kb,
    uchar * in_msg,
    const char * file,
    int line )

```

os_usbrecv does what its name says:

The comment at the beginning of the procedure causes the suspicion that the firmware versionspecific distinction is missing for receiving from usb endpoint 3 or 4. The commented code contains only the reception from EP4, but this may be wrong for a software version 2.0 or higher (see the code for os_usbseend ()).

So all the receiving is done via an ioctl() like in os_usbseend. The ioctl() is given a struct usbdevfs_ctrltransfer, in which the relevant parameters are entered:

bRequest↔ Type	bRequest	wValue	EP	size	Timeout	data
0xA1	0x01	0x0200	endpoint to be addressed from epcount - 1	MSG_SIZE	5ms	the message buffer pointer
Device to Host, Type=Class, Recipi- ent=Interface	1 = RECEIVE?	specific	Interface #	64	5000	in_msg

The `ioctl()` returns the number of bytes received. Here is the usual check again:

- If the return value is -1 AND the error is a timeout (ETIMEDOUT), `os_usbrecv()` will return -1 to indicate that it is probably a recoverable problem and a retry is recommended.
- For another negative value or other error identifier OR 0 bytes are received, 0 is returned as an identifier for a heavy error.
- In all other cases, the function returns the number of bytes received.

If this is not the entire blocksize (MSG_SIZE bytes), an error message is issued on the standard error channel [warning "Read YY bytes (expected 64)"].

Definition at line 127 of file `usb_linux.c`.

References `ckb_err_fn`, `ckb_warn_fn`, `usbdevice::epcount`, `usbdevice::handle`, and `MSG_SIZE`.

Referenced by `_usbrecv()`.

```

127                                     {
128     int res;
129     // This is what CUE does, but it doesn't seem to work on linux.
130     /*if(kb->fwversion >= 0x130){
131         struct usbdevfs_bulktransfer transfer;
132         memset(&transfer, 0, sizeof(transfer));
133         transfer.ep = 0x84;
134         transfer.len = MSG_SIZE;
135         transfer.timeout = 5000;
136         transfer.data = in_msg;
137         res = ioctl(kb->handle - 1, USBDEVFS_BULK, &transfer);
138     } else {*/
139         struct usbdevfs_ctrltransfer transfer = { 0xa1, 0x01, 0x0300, kb->
140     epcount - 1, MSG_SIZE, 5000, in_msg };
141         res = ioctl(kb->handle - 1, USBDEVFS_CONTROL, &transfer);
142     }
143     if(res <= 0){
144         ckb_err_fn("%s\n", file, line, res ? strerror(errno) : "No data read");
145         if(res == -1 && errno == ETIMEDOUT)
146             return -1;
147         else
148             return 0;
149     } else if(res != MSG_SIZE)
150         ckb_warn_fn("Read %d bytes (expected %d)\n", file, line, res,
151     MSG_SIZE);
152     return res;
153 }
```

Here is the caller graph for this function:



5.39.3.8 os_usbseend()

```
int os_usbseend (
    usbdevice * kb,
    const uchar * out_msg,
    int is_recv,
    const char * file,
    int line )
```

os_usbseend has two functions:

- if is_recv == false, it tries to send a given MSG_SIZE buffer via the usb interface given with kb.
- otherwise a request is sent via the usb device to initiate the receiving of a message from the remote device.

The functionality for sending distinguishes two cases, depending on the version number of the firmware of the connected device:

If the firmware is less or equal 1.2, the transmission is done via an ioctl(). The ioctl() is given a struct usbdevfs_↵ ctrltransfer, in which the relevant parameters are entered:

bRequest↵ Type	bRequest	wValue	EP	size	Timeout	data
0x21	0x09	0x0200	endpoint / IF to be addressed from epcount- 1	MSG_SIZE	5000 (=5ms)	the message buffer pointer
Host to Device, Type=Class, Recipi- ent=Interface	9 = Send data?	specific	last or pre-last device #	64	5000	out_msg

The ioctl command is USBDEVFS_CONTROL.

The same constellation is used if the device is requested to send its data (is_recv = true).

For a more recent firmware and is_recv = false, the ioctl command USBDEVFS_CONTROL is not used (this tells the bus to enter the control mode), but the bulk method is used: USBDEVFS_BULK. This is astonishing, because all of the endpoints are type Interrupt, not bulk.

Anyhow, for this purpose a different structure is used for the ioctl() (struct **usbdevfs_bulktransfer**) and this is also initialized differently:

The length and timeout parameters are given the same values as above. The formal parameter out_msg is also passed as a buffer pointer. For the endpoints, the firmware version is differentiated again:

For a firmware version between 1.3 and <2.0 endpoint 4 is used, otherwise (it can only be >=2.0) endpoint 3 is used.

Todo Since the handling of endpoints has already led to problems elsewhere, this implementation is extremely hardware-dependent and critical!

Eg. the new keyboard K95PLATINUMRGB has a version number significantly less than 2.0 - will it run with this implementation?

The ioctl() - no matter what type - returns the number of bytes sent. Now comes the usual check:

- If the return value is -1 AND the error is a timeout (ETIMEDOUT), `os_usbsend()` will return -1 to indicate that it is probably a recoverable problem and a retry is recommended.
- For another negative value or other error identifier OR 0 bytes sent, 0 is returned as a heavy error identifier.
- In all other cases, the function returns the number of bytes sent.

If this is not the entire blocksize (MSG_SIZE bytes), an error message is issued on the standard error channel [warning "Wrote YY bytes (expected 64)"].

If DEBUG_USB is set during compilation, the number of bytes sent and their representation are logged to the error channel.

Definition at line 66 of file usb_linux.c.

References `ckb_err_fn`, `ckb_warn_fn`, `usbdevice::epcount`, `usbdevice::fwversion`, `usbdevice::handle`, and `MSG_SIZE`.

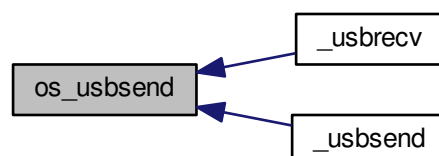
Referenced by `_usbrecv()`, and `_usbsend()`.

```

66                                     {
67     int res;
68     if(kb->fwversion >= 0x120 && !is_recv){
69         struct usbdevfs_bulktransfer transfer;
70         memset(&transfer, 0, sizeof(transfer));
71         transfer.ep = (kb->fwversion >= 0x130 && kb->fwversion < 0x200) ? 4 : 3;
72         transfer.len = MSG_SIZE;
73         transfer.timeout = 5000;
74         transfer.data = (void*)out_msg;
75         res = ioctl(kb->handle - 1, USBDEVFS_BULK, &transfer);
76     } else {
77         struct usbdevfs_ctltransfer transfer = { 0x21, 0x09, 0x0200, kb->
78     epcount - 1, MSG_SIZE, 5000, (void*)out_msg };
79         res = ioctl(kb->handle - 1, USBDEVFS_CONTROL, &transfer);
80     }
81     if(res <= 0){
82         ckb_err_fn("%s\n", file, line, res ? strerror(errno) : "No data written");
83         if(res == -1 && errno == ETIMEDOUT)
84             return -1;
85         else
86             return 0;
87     } else if(res != MSG_SIZE)
88         ckb_warn_fn("Wrote %d bytes (expected %d)\n", file, line, res,
89     MSG_SIZE);
90 #ifdef DEBUG_USB
91     char converted[MSG_SIZE*3 + 1];
92     for(int i=0;i<MSG_SIZE;i++)
93         sprintf(&converted[i*3], "%02x ", out_msg[i]);
94     ckb_warn_fn("Sent %s\n", file, line, converted);
95 #endif
96     return res;
97 }

```

Here is the caller graph for this function:



5.39.3.9 `strtrim()`

```
void strtrim (
    char * string )
```

`strtrim` trims a string by removing leading and trailing spaces.

Parameters

<i>string</i>	
---------------	--

Definition at line 499 of file `usb_linux.c`.

Referenced by `os_setupusb()`.

```
499     {
500         // Find last non-space
501         char* last = string;
502         for(char* c = string; *c != 0; c++){
503             if(!isspace(*c))
504                 last = c;
505         }
506         last[1] = 0;
507         // Find first non-space
508         char* first = string;
509         for(; *first != 0; first++){
510             if(!isspace(*first))
511                 break;
512         }
513         if(first != string)
514             memmove(string, first, last - first);
515     }
```

Here is the caller graph for this function:



5.39.3.10 `udev_enum()`

```
static void udev_enum ( ) [static]
```

Reduce the hits of the enumeration by limiting to usb as technology and corsair as idVendor. Then filter with `udev_↵_enumerate_scan_devices ()` all hits.

The following call to `udev_enumerate_get_list_entry()` fetches the entire hitlist as `udev_list_entry *`.

Use `udev_list_entry_foreach()` to iterate through the hit set.

If both the device name exists (`udev_list_entry_get_name`) and the subsequent creation of a new `udev_device` (`udev_device_new_from_syspath`) is ok, the new device is added to the list with [usb_add_device\(\)](#).

If the latter does not work, the new device is released again (`udev_device_unref ()`).

After the last iteration, the enumerator is released with `udev_enumerate_unref ()`.

Definition at line 724 of file `usb_linux.c`.

References `udev`, `usb_add_device()`, and `V_CORSAIR_STR`.

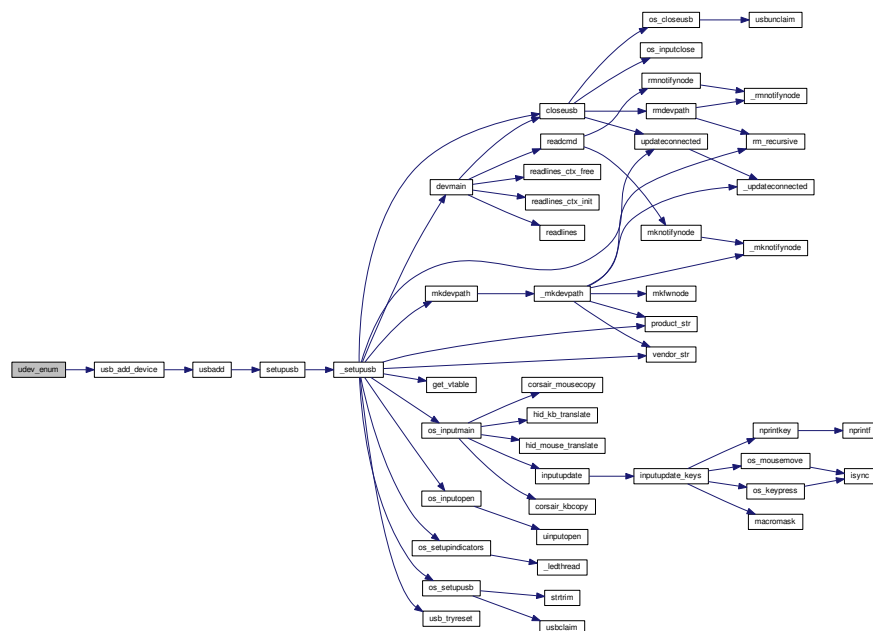
Referenced by `usbmain()`.

```

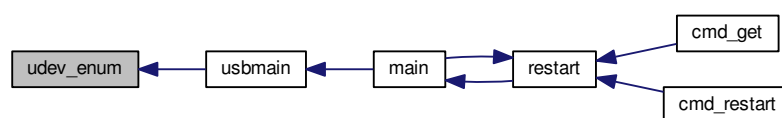
724         {
725             struct udev_enumerate* enumerator = udev_enumerate_new(udev);
726             udev_enumerate_add_match_subsystem(enumerator, "usb");
727             udev_enumerate_add_match_sysattr(enumerator, "idVendor", V_CORSAIR_STR);
728             udev_enumerate_scan_devices(enumerator);
729             struct udev_list_entry* devices, *dev_list_entry;
730             devices = udev_enumerate_get_list_entry(enumerator);
731
732             udev_list_entry_foreach(dev_list_entry, devices){
733                 const char* path = udev_list_entry_get_name(dev_list_entry);
734                 if(!path)
735                     continue;
736                 struct udev_device* dev = udev_device_new_from_syspath(udev, path);
737                 if(!dev)
738                     continue;
739                 // If the device matches a recognized device ID, open it
740                 if(usb_add_device(dev))
741                     // Release device if not
742                     udev_device_unref(dev);
743             }
744             udev_enumerate_unref(enumerator);
745         }

```

Here is the call graph for this function:



Here is the caller graph for this function:



5.39.3.11 usb_add_device()

```
static int usb_add_device (
    struct udev_device * dev ) [static]
```

Parameters

<code>dev</code>	the functions <code>usb_*_device</code> get a struct <code>udev*</code> with the necessary hardware-related information.
------------------	--

Returns

the retval of `usbadd()`: 0 if device was recognized/added or 1 if either vendor is not corsair or product is not mentioned in `model[]`.

First get the `idVendor` via `udev_device_get_sysattr_value()`. If this is equal to the ID-string of corsair ("1b1c"), get the `idProduct` on the same way.

If we can find the model name in the model array, call `usbadd()` with the model number.

Todo So why the hell not a transformation between the string and the short presentation? Lets check if the string representation is used elsewhere.

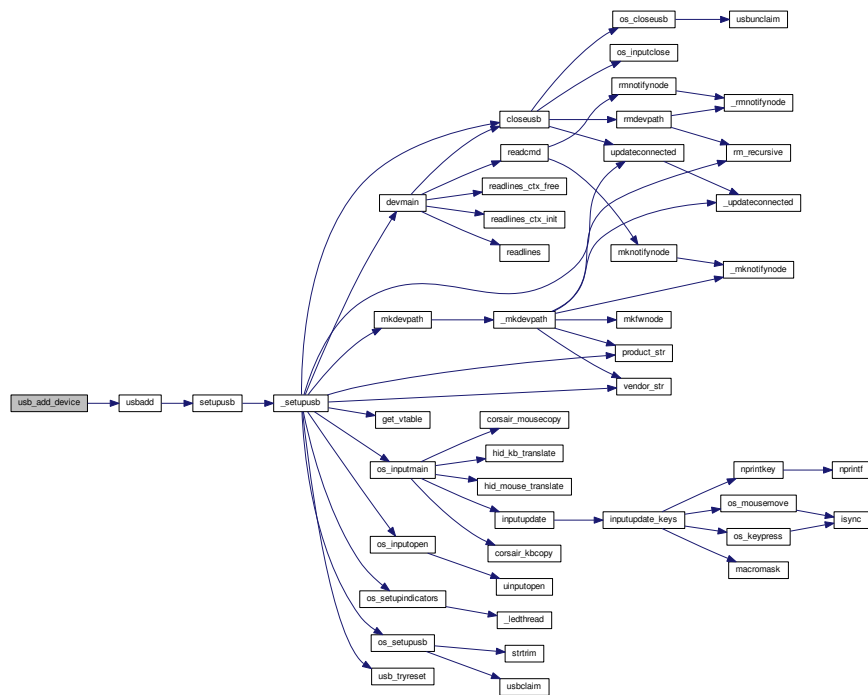
Definition at line 672 of file `usb_linux.c`.

References `N_MODELS`, `usbadd()`, `V_CORSAIR`, and `V_CORSAIR_STR`.

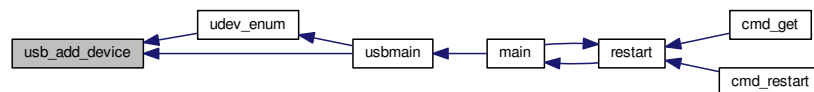
Referenced by `udev_enum()`, and `usbmain()`.

```
672                                     {
673     const char* vendor = udev_device_get_sysattr_value(dev, "idVendor");
674     if(vendor && !strcmp(vendor, V_CORSAIR_STR)){
675         const char* product = udev_device_get_sysattr_value(dev, "idProduct");
676         if(product){
677             for(_model* model = models; model < models +
N_MODELS; model++){
678                 if(!strcmp(product, model->name)){
679                     return usbadd(dev, V_CORSAIR, model->number);
680                 }
681             }
682         }
683     }
684     return 1;
685 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



5.39.3.12 usb_rm_device()

```
static void usb_rm_device (
    struct udev_device * dev ) [static]
```

Parameters

<code>dev</code>	the functions <code>usb_*_device</code> get a struct <code>udev*</code> with the necessary hardware-related information.
------------------	--

First try to find the system path of the device given in parameter dev. The index where the name is found is the same index we need to address the global keyboard array. That array holds all usbdevices. Searching for the correct name in kbsyspath-array and closing the usb via `closeusb()` are protected by lock..unlock of the corresponding devmutex arraymember.

Definition at line 697 of file usb_linux.c.

References `closeusb()`, `DEV_MAX`, `devmutex`, `kbsyspath`, and `keyboard`.

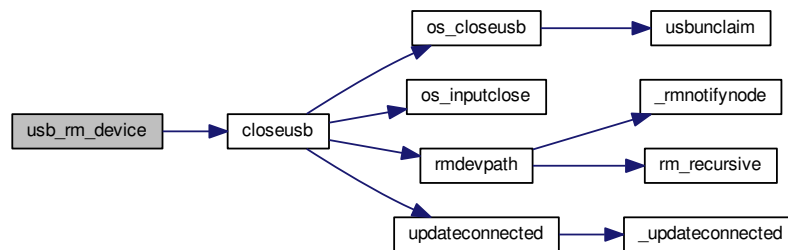
Referenced by `usbmain()`.

```

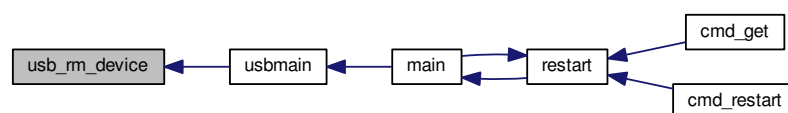
697                                     {
698     // Device removed. Look for it in our list of keyboards
699     const char* syspath = udev_device_get_syspath(dev);
700     if(!syspath || syspath[0] == 0)
701         return;
702     for(int i = 1; i < DEV_MAX; i++){
703         pthread_mutex_lock(devmutex + i);
704         if(!strcmp(syspath, kbsyspath[i]))
705             closeusb(keyboard + i);
706         pthread_mutex_unlock(devmutex + i);
707     }
708 }

```

Here is the call graph for this function:



Here is the caller graph for this function:



5.39.3.13 usbadd()

```

int usbadd (
    struct udev_device * dev,
    short vendor,
    short product )

```

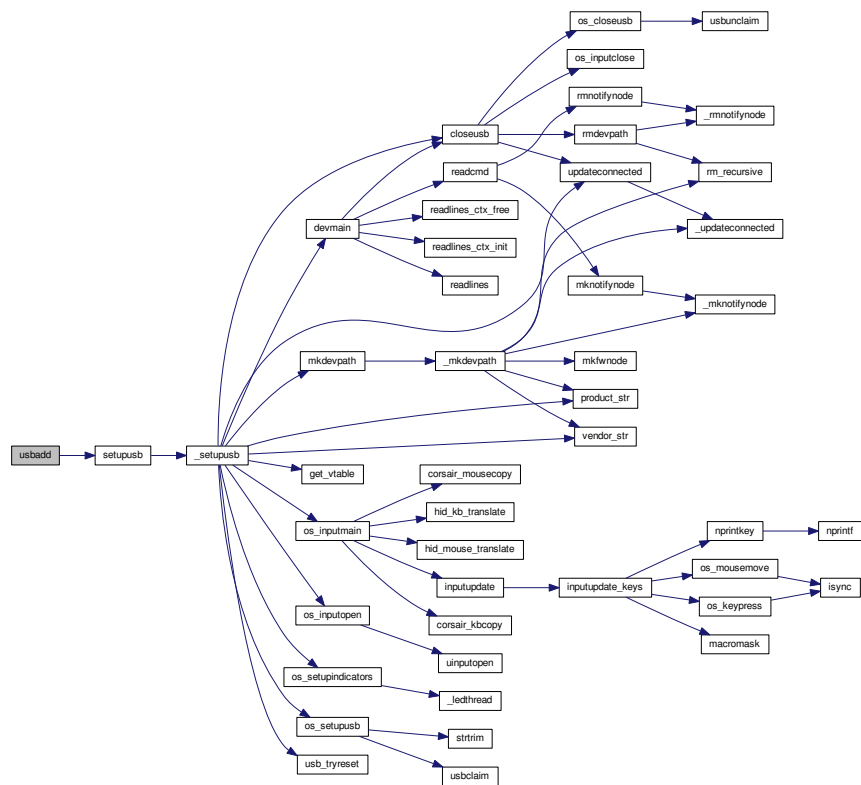
Definition at line 569 of file usb_linux.c.

References `ckb_err`, `DEV_MAX`, `dmutex`, `usbdevice::handle`, `IS_CONNECTED`, `kbsyspath`, `keyboard`, `usbdevice::product`, `setupusb()`, `usbdevice::udev`, and `usbdevice::vendor`.

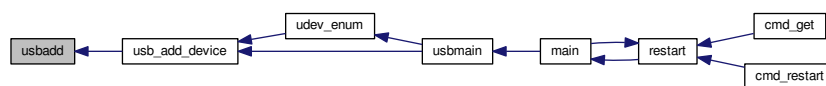
Referenced by `usb_add_device()`.

```
569                                     {
570     const char* path = udev_device_get_devnode(dev);
571     const char* syspath = udev_device_get_syspath(dev);
572     if(!path || !syspath || path[0] == 0 || syspath[0] == 0){
573         ckb_err("Failed to get device path\n");
574         return -1;
575     }
576     // Find a free USB slot
577     for(int index = 1; index < DEV_MAX; index++){
578         usbdevice* kb = keyboard + index;
579         if(pthread_mutex_trylock(dmutex(kb))){
580             // If the mutex is locked then the device is obviously in use, so keep going
581             if(!strcmp(syspath, kbsyspath[index])){
582                 // Make sure this existing keyboard doesn't have the same syspath (this shouldn't happen)
583                 return 0;
584             }
585             continue;
586         }
587         if(!IS_CONNECTED(kb)){
588             // Open the sysfs device
589             kb->handle = open(path, O_RDWR) + 1;
590             if(kb->handle <= 0){
591                 ckb_err("Failed to open USB device: %s\n", strerror(errno));
592                 kb->handle = 0;
593                 pthread_mutex_unlock(dmutex(kb));
594                 return -1;
595             } else {
596                 // Set up device
597                 kb->udev = dev;
598                 kb->vendor = vendor;
599                 kb->product = product;
600                 strncpy(kbsyspath[index], syspath, FILENAME_MAX);
601                 // Mutex remains locked
602                 setupusb(kb);
603                 return 0;
604             }
605         }
606         pthread_mutex_unlock(dmutex(kb));
607     }
608     ckb_err("No free devices\n");
609     return -1;
610 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



5.39.3.14 usbclaim()

```
static int usbclaim (
    usbdevice * kb ) [static]
```

usbclaim does claiming all EPs for the usb device gicen by kb.

Parameters

<i>kb</i>	THE usbdevice*
-----------	----------------

Returns

0 on success, -1 otherwise.

Claim all endpoints for a given device (remember the decrementing of the file descriptor) via `ioctl(USBDEVFS_DISCONNECT)` and `ioctl(USBDEVFS_CLAIMINTERFACE)`.

Error handling is done for the `ioctl(USBDEVFS_CLAIMINTERFACE)` only. If this fails, now an error message is thrown and -1 is returned. Function is called in `usb_linux.c` only, so it is declared as static now.

Definition at line 447 of file `usb_linux.c`.

References `ckb_err`, `usbdevice::epcount`, and `usbdevice::handle`.

Referenced by `os_resetusb()`, and `os_setupusb()`.

```

447     {
448     int count = kb->epcount;
449     for (int i = 0; i < count; i++) {
450         int rv;
451         struct usbdevfs_ioctl ctl = { i, USBDEVFS_DISCONNECT, 0 };
452         if ((rv = ioctl(kb->handle - 1, USBDEVFS_IOCTL, &ctl)) {
453             ckb_err("usbclaim 1: got retcode %d and errno = %d, %s. Ignoring.\n", rv, errno,
strerror(errno));
454         }
455
456         if ((rv = ioctl(kb->handle - 1, USBDEVFS_CLAIMINTERFACE, &i)) {
457             ckb_err("usbclaim 2: got retcode %d and errno = %d, %s. Aborting.\n", rv, errno,
strerror(errno));
458             return -1;
459         }
460     }
461     return 0;
462 }
```

Here is the caller graph for this function:



5.39.3.15 usbkill()

```
void usbkill ( )
```

Stop the USB system.

Definition at line 815 of file `usb_linux.c`.

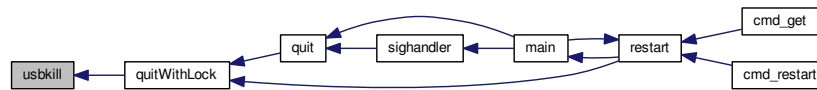
References `udev`.

Referenced by `quitWithLock()`.

```

815     {
816         udev_unref(udev);
817         udev = 0;
818     }
```

Here is the caller graph for this function:



5.39.3.16 usbmain()

```
int usbmain ( )
```

Start the USB main loop. Returns program exit code when finished.

usbmain is called by [main\(\)](#) after setting up all other stuff.

Returns

0 normally or -1 if fatal error occurs (up to now only if no new devices are available)

First check whether the uinput module is loaded by the kernel.

Todo Why isn't missing of uinput a fatal error?

Create the udev object with `udev_new()` (is a function from `libudev.h`) terminate -1 if error

Enumerate all currently connected devices

Todo lae. here the work has to go on...

Definition at line 752 of file `usb_linux.c`.

References `ckb_fatal`, `ckb_warn`, `udev`, `udev_enum()`, `usb_add_device()`, and `usb_rm_device()`.

Referenced by `main()`.

```

752     {
753     // Load the uinput module (if it's not loaded already)
754     if(system("modprobe uinput") != 0)
755         ckb_warn("Failed to load uinput module\n");
756
757     if(!(udev = udev_new())) {
758         ckb_fatal("Failed to initialize udev in usbmain(), usb_linux.c\n");
759         return -1;
760     }
761     udev_enum();
762
763     // Done scanning. Enter a loop to poll for device updates
764     struct udev_monitor* monitor = udev_monitor_new_from_netlink(udev, "udev");
765     udev_monitor_filter_add_match_subsystem_devtype(monitor, "usb", 0);
766     udev_monitor_enable_receiving(monitor);
767     // Get an fd for the monitor
768     int fd = udev_monitor_get_fd(monitor);

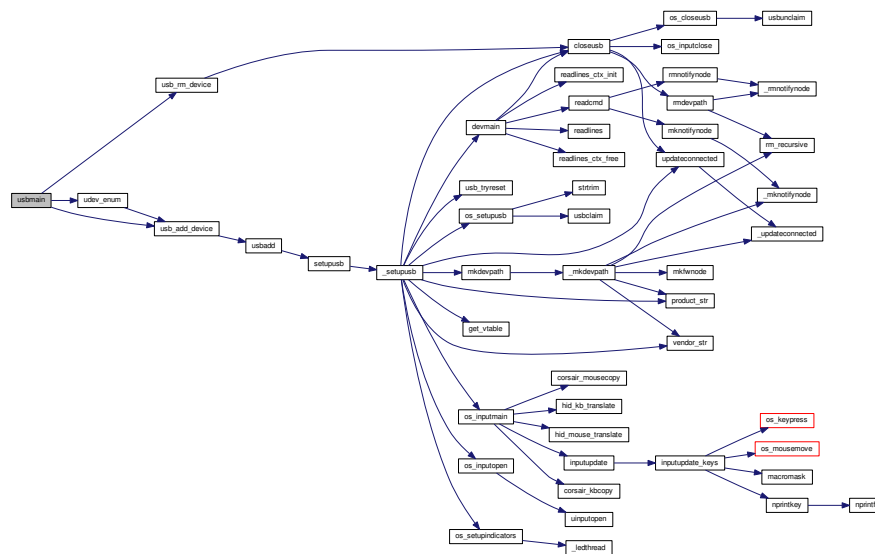
```

```

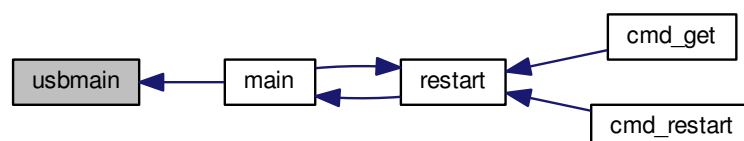
781 fd_set fds;
782 while(udev){
783     FD_ZERO(&fds);
784     FD_SET(fd, &fds);
785     // Block until an event is read
786     if(select(fd + 1, &fds, 0, 0, 0) > 0 && FD_ISSET(fd, &fds)){
787         struct udev_device* dev = udev_monitor_receive_device(monitor);
788         if(!dev)
789             continue;
790         const char* action = udev_device_get_action(dev);
791         if(!action){
792             udev_device_unref(dev);
793             continue;
794         }
795         // Add/remove device
796         if(!strcmp(action, "add")){
797             int res = usb_add_device(dev);
798             if(res == 0)
799                 continue;
800             // If the device matched but the handle wasn't opened correctly, re-enumerate (this
sometimes solves the problem)
801             if(res == -1)
802                 udev_enum();
803         } else if(!strcmp(action, "remove"))
804             usb_rm_device(dev);
805         udev_device_unref(dev);
806     }
807 }
808 udev_monitor_unref(monitor);
809 return 0;
810 }

```

Here is the call graph for this function:



Here is the caller graph for this function:



5.39.4 Variable Documentation

5.39.4.1 kbsyspath

```
char kbsyspath[9][FILENAME_MAX] [static]
```

Definition at line 11 of file usb_linux.c.

Referenced by `os_closeusb()`, `usb_rm_device()`, and `usbadd()`.

5.39.4.2 models

```
_model models[] [static]
```

Initial value:

```
= {
    { "1b17" , 0x1b17 },
    { "1b07" , 0x1b07 },
    { "1b37" , 0x1b37 },
    { "1b39" , 0x1b39 },
    { "1b13" , 0x1b13 },
    { "1b09" , 0x1b09 },
    { "1b33" , 0x1b33 },
    { "1b36" , 0x1b36 },
    { "1b38" , 0x1b38 },
    { "1b3a" , 0x1b3a },
    { "1b11" , 0x1b11 },
    { "1b08" , 0x1b08 },
    { "1b2d" , 0x1b2d },
    { "1b20" , 0x1b20 },
    { "1b15" , 0x1b15 },

    { "1b12" , 0x1b12 },
    { "1b2e" , 0x1b2e },
    { "1b14" , 0x1b14 },
    { "1b19" , 0x1b19 },
    { "1b2f" , 0x1b2f },
    { "1b1e" , 0x1b1e },
    { "1b3e" , 0x1b3e },
    { "1b32" , 0x1b32 },
    { "1b3c" , 0x1b3c }
}
```

Attention

when adding new hardware this file hat to be changed too.

In this structure array `models[]` for each device the name (the device id as string in hex without leading 0x) and its usb device id as short must be entered in this array.

Definition at line 631 of file usb_linux.c.

5.39.4.3 udev

```
struct udev* udev [static]
```

Definition at line 614 of file usb_linux.c.

Referenced by udev_enum(), usbkill(), and usbmain().

5.39.4.4 udevthread

```
pthread_t udevthread
```

Definition at line 617 of file usb_linux.c.

5.39.4.5 usbthread

```
pthread_t usbthread
```

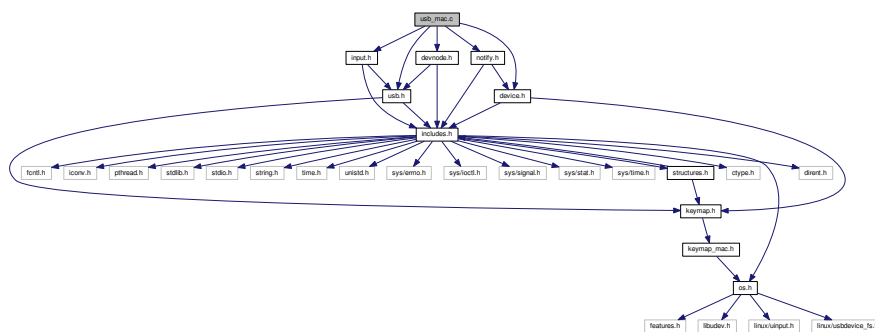
Todo These two thread variables seem to be unused: usbthread, udevthread

Definition at line 617 of file usb_linux.c.

5.40 usb_mac.c File Reference

```
#include "device.h"
#include "devnode.h"
#include "input.h"
#include "notify.h"
#include "usb.h"
```

Include dependency graph for usb_mac.c:



Index

- `_DEFAULT_SOURCE`
 - `os.h`, [271](#)
 - `_GNU_SOURCE`
 - `os.h`, [271](#)
 - `__FILE_NOPATH__`
 - `includes.h`, [105](#)
 - `_cmd_get`
 - `notify.c`, [256](#)
 - `_cmd_macro`
 - `input.c`, [111](#)
 - `_freeprofile`
 - `profile.c`, [273](#)
 - `_ledthread`
 - `input_linux.c`, [133](#)
 - `_mkdevpath`
 - `devnode.c`, [59](#)
 - `_mknotifynode`
 - `devnode.c`, [61](#)
 - `_model`, [410](#)
 - `_nk95cmd`
 - `usb.h`, [379](#)
 - `usb_linux.c`, [411](#)
 - `_readlines_ctx`, [58](#)
 - `_resetusb`
 - `usb.c`, [339](#)
 - `usb.h`, [380](#)
 - `_rmnotifynode`
 - `devnode.c`, [62](#)
 - `_setupusb`
 - `usb.c`, [340](#)
 - `_start_dev`
 - `device.c`, [29](#)
 - `_updateconnected`
 - `devnode.c`, [62](#)
 - `_usbrecv`
 - `usb.c`, [344](#)
 - `usb.h`, [381](#)
 - `_usbsend`
 - `usb.c`, [346](#)
 - `usb.h`, [383](#)
- `ACT_LIGHT`
 - `device.h`, [33](#)
- `ACT_LOCK`
 - `device.h`, [34](#)
- `ACT_M1`
 - `device.h`, [34](#)
- `ACT_M2`
 - `device.h`, [34](#)
- `ACT_M3`
 - `device.h`, [34](#)
- `ACT_MR_RING`
 - `device.h`, [34](#)
- `ACT_NEXT_NOWRAP`
 - `device.h`, [35](#)
- `ACT_NEXT`
 - `device.h`, [35](#)
- `active`
 - `devcmd.__unnamed__`, [8](#)
- `allocprofile`
 - `devcmd.__unnamed__`, [8](#)
 - `profile.c`, [274](#)
 - `profile.h`, [293](#)
- `BR1`
 - `led_keyboard.c`, [201](#)
- `BR2`
 - `led_keyboard.c`, [201](#)
- `BR4`
 - `led_keyboard.c`, [202](#)
- `BTN_WHEELDOWN`
 - `keymap.h`, [149](#)
- `BTN_WHEELUP`
 - `keymap.h`, [149](#)
- `BUTTON_HID_COUNT`
 - `keymap.c`, [142](#)
- `bind`
 - `devcmd.__unnamed__`, [8](#)
- `binding`, [320](#)
- `bit_reverse_table`
 - `led_keyboard.c`, [212](#)
- `CLEAR_KEYBIT`
 - `structures.h`, [329](#)
- `CMD_COUNT`
 - `command.h`, [21](#)
- `CMD_DEV_COUNT`
 - `command.h`, [21](#)
- `ckb_err`
 - `includes.h`, [105](#)
- `ckb_err_fn`
 - `includes.h`, [105](#)
- `ckb_err_nofile`
 - `includes.h`, [105](#)
- `ckb_fatal`
 - `includes.h`, [106](#)
- `ckb_fatal_fn`
 - `includes.h`, [106](#)
- `ckb_fatal_nofile`
 - `includes.h`, [106](#)

- ckb_info
 - includes.h, 106
- ckb_info_fn
 - includes.h, 107
- ckb_info_nofile
 - includes.h, 107
- ckb_s_err
 - includes.h, 107
- ckb_s_out
 - includes.h, 107
- ckb_warn
 - includes.h, 107
- ckb_warn_fn
 - includes.h, 108
- ckb_warn_nofile
 - includes.h, 108
- closeusb
 - usb.c, 347
 - usb.h, 385
- cmd
 - command.h, 22
- cmd_active_kb
 - device.h, 37
 - device_keyboard.c, 45
- cmd_active_mouse
 - device.h, 37
 - device_mouse.c, 50
- cmd_bind
 - input.c, 113
 - input.h, 123
- cmd_dpi
 - dpi.c, 82
 - dpi.h, 89
- cmd_dpisel
 - dpi.c, 83
 - dpi.h, 90
- cmd_erase
 - profile.c, 275
 - profile.h, 294
- cmd_eraseprofile
 - profile.c, 276
 - profile.h, 295
- cmd_fwupdate
 - firmware.c, 97
 - firmware.h, 101
- cmd_get
 - notify.c, 259
 - notify.h, 266
- cmd_hwload_kb
 - profile.h, 296
 - profile_keyboard.c, 311
- cmd_hwload_mouse
 - profile.h, 297
 - profile_mouse.c, 314
- cmd_hwsave_kb
 - profile.h, 298
 - profile_keyboard.c, 312
- cmd_hwsave_mouse
 - profile.h, 299
 - profile_mouse.c, 315
- cmd_iauto
 - led.c, 159
 - led.h, 166
- cmd_id
 - profile.c, 276
 - profile.h, 300
- cmd_idle_kb
 - device.h, 38
 - device_keyboard.c, 46
- cmd_idle_mouse
 - device.h, 38
 - device_mouse.c, 50
- cmd_inotify
 - led.c, 160
 - led.h, 167
- cmd_io_none
 - device_vtable.c, 54
- cmd_ioff
 - led.c, 160
 - led.h, 167
- cmd_ion
 - led.c, 161
 - led.h, 168
- cmd_lift
 - dpi.c, 83
 - dpi.h, 90
- cmd_macro
 - input.c, 114
 - input.h, 123
- cmd_macro_none
 - device_vtable.c, 54
- cmd_name
 - profile.c, 277
 - profile.h, 301
- cmd_none
 - device_vtable.c, 54
- cmd_notify
 - notify.c, 260
 - notify.h, 266
- cmd_pollrate
 - device.h, 39
 - device_mouse.c, 51
- cmd_profileid
 - profile.c, 277
 - profile.h, 301
- cmd_profilename
 - profile.c, 278
 - profile.h, 302
- cmd_rebind
 - input.c, 114
 - input.h, 124
- cmd_restart
 - notify.c, 260
 - notify.h, 267
- cmd_rgb
 - led.c, 162

- led.h, 169
- cmd_snap
 - dpi.c, 84
 - dpi.h, 91
- cmd_strings
 - command.c, 18
- cmd_unbind
 - input.c, 115
 - input.h, 124
- cmdhandler
 - command.h, 21
- cmdhandler_io
 - command.h, 21
- cmdhandler_mac
 - command.h, 21
- command.c, 13
 - cmd_strings, 18
 - readcmd, 14
 - TRY_WITH_RESET, 14
- command.h, 19
 - CMD_COUNT, 21
 - CMD_DEV_COUNT, 21
 - cmd, 22
 - cmdhandler, 21
 - cmdhandler_io, 21
 - cmdhandler_mac, 21
 - devcmd, 22
 - readcmd, 24
- corsair_kbcopy
 - keymap.c, 142
 - keymap.h, 153
- corsair_mousecopy
 - keymap.c, 142
 - keymap.h, 154
- DELAY_LONG
 - usb.h, 363
- DELAY_MEDIUM
 - usb.h, 363
- DELAY_SHORT
 - usb.h, 363
- DEV_MAX
 - device.h, 35
- DPI_COUNT
 - structures.h, 329
- devcmd, 20
 - command.h, 22
- devcmd.__unnamed__, 7
 - active, 8
 - allocprofile, 8
 - bind, 8
 - dpi, 8
 - dpisel, 9
 - erase, 9
 - eraseprofile, 9
 - freeprofile, 9
 - fwupdate, 9
 - get, 9
 - hwload, 9
 - hwsave, 9
 - iauto, 9
 - id, 9
 - idle, 10
 - inotify, 10
 - ioff, 10
 - ion, 10
 - lift, 10
 - loadprofile, 10
 - macro, 10
 - name, 10
 - notify, 10
 - pollrate, 10
 - profileid, 11
 - profilename, 11
 - rebind, 11
 - restart, 11
 - rgb, 11
 - setmodeindex, 11
 - snap, 11
 - start, 11
 - unbind, 11
 - updatedpi, 11
 - updateindicators, 11
 - updatergb, 11
- device.c, 28
 - _start_dev, 29
 - devlistmutex, 31
 - devmutex, 31
 - hwload_mode, 31
 - inputmutex, 32
 - keyboard, 32
 - start_dev, 31
- device.h, 32
 - ACT_LIGHT, 33
 - ACT_LOCK, 34
 - ACT_M1, 34
 - ACT_M2, 34
 - ACT_M3, 34
 - ACT_MR_RING, 34
 - ACT_NEXT_NOWRAP, 35
 - ACT_NEXT, 35
 - cmd_active_kb, 37
 - cmd_active_mouse, 37
 - cmd_idle_kb, 38
 - cmd_idle_mouse, 38
 - cmd_pollrate, 39
 - DEV_MAX, 35
 - devmutex, 44
 - dmutex, 35
 - IN_CORSAIR, 36
 - IN_HID, 36
 - IS_CONNECTED, 36
 - imutex, 35
 - inputmutex, 44
 - keyboard, 44
 - setactive, 36
 - setactive_kb, 39

- setactive_mouse, 41
 - setmodeindex_nrgb, 43
 - start_dev, 43
 - start_kb_nrgb, 44
- device_keyboard.c, 45
 - cmd_active_kb, 45
 - cmd_idle_kb, 46
 - setactive_kb, 46
 - setmodeindex_nrgb, 48
 - start_kb_nrgb, 49
- device_mouse.c, 49
 - cmd_active_mouse, 50
 - cmd_idle_mouse, 50
 - cmd_pollrate, 51
 - setactive_mouse, 51
- device_vtable.c, 53
 - cmd_io_none, 54
 - cmd_macro_none, 54
 - cmd_none, 54
 - int1_int_none, 55
 - int1_void_none, 55
 - loadprofile_none, 55
 - vtable_keyboard, 56
 - vtable_keyboard_nonrgb, 56
 - vtable_mouse, 56
- devlistmutex
 - device.c, 31
- devmain
 - usb.c, 349
- devmutex
 - device.c, 31
 - device.h, 44
- devnode.c, 56
 - _mkdevpath, 59
 - _mknotifynode, 61
 - _rmnotifynode, 62
 - _updateconnected, 62
 - devpath, 71
 - gid, 71
 - MAX_BUFFER, 58
 - mkdevpath, 63
 - mkfwnode, 64
 - mknotifynode, 65
 - readlines, 66
 - readlines_ctx_free, 67
 - readlines_ctx_init, 67
 - rm_recursive, 68
 - rmdevpath, 68
 - rmnotifynode, 69
 - S_GID_READ, 58
 - updateconnected, 70
- devnode.h, 72
 - devpath, 81
 - gid, 81
 - mkdevpath, 74
 - mkfwnode, 75
 - mknotifynode, 76
 - readlines, 77
 - readlines_ctx, 74
 - readlines_ctx_free, 78
 - readlines_ctx_init, 78
 - rmdevpath, 78
 - rmnotifynode, 79
 - S_CUSTOM_R, 73
 - S_CUSTOM, 73
 - S_READDIR, 73
 - S_READWRITE, 74
 - S_READ, 73
 - updateconnected, 80
- devpath
 - devnode.c, 71
 - devnode.h, 81
- dmutex
 - device.h, 35
- dpi
 - devcmd.__unnamed__, 8
- dpi.c, 82
 - cmd_dpi, 82
 - cmd_dpisel, 83
 - cmd_lift, 83
 - cmd_snap, 84
 - loaddpi, 84
 - printdpi, 86
 - savedpi, 86
 - updatedpi, 87
- dpi.h, 88
 - cmd_dpi, 89
 - cmd_dpisel, 90
 - cmd_lift, 90
 - cmd_snap, 91
 - loaddpi, 91
 - printdpi, 92
 - savedpi, 93
 - updatedpi, 94
- dpisel
 - devcmd.__unnamed__, 9
- dpiset, 321
- erase
 - devcmd.__unnamed__, 9
- eraseprofile
 - devcmd.__unnamed__, 9
- euid_guard_start
 - os.h, 271
- euid_guard_stop
 - os.h, 271
- extra_mac.c, 95
- FEAT_ADJRATE
 - structures.h, 329
- FEAT_ANSI
 - structures.h, 329
- FEAT_BIND
 - structures.h, 330
- FEAT_COMMON
 - structures.h, 330
- FEAT_FWUPDATE

- structures.h, [330](#)
- FEAT_FWVERSION
 - structures.h, [330](#)
- FEAT_HWLOAD
 - structures.h, [330](#)
- FEAT_ISO
 - structures.h, [331](#)
- FEAT_LMASK
 - structures.h, [331](#)
- FEAT_MONOCHROME
 - structures.h, [331](#)
- FEAT_MOUSEACCEL
 - structures.h, [331](#)
- FEAT_NOTIFY
 - structures.h, [331](#)
- FEAT_POLLRATE
 - structures.h, [332](#)
- FEAT_RGB
 - structures.h, [332](#)
- FEAT_STD_NRGB
 - structures.h, [332](#)
- FEAT_STD_RGB
 - structures.h, [332](#)
- FW_MAXSIZE
 - firmware.c, [96](#)
- FW_NOFILE
 - firmware.c, [96](#)
- FW_OK
 - firmware.c, [96](#)
- FW_USBFAIL
 - firmware.c, [96](#)
- FW_WRONGDEV
 - firmware.c, [96](#)
- features_mask
 - main.c, [254](#)
 - usb.c, [358](#)
- firmware.c, [95](#)
 - cmd_fwupdate, [97](#)
 - FW_MAXSIZE, [96](#)
 - FW_NOFILE, [96](#)
 - FW_OK, [96](#)
 - FW_USBFAIL, [96](#)
 - FW_WRONGDEV, [96](#)
 - fwupdate, [97](#)
 - getfwversion, [99](#)
- firmware.h, [101](#)
 - cmd_fwupdate, [101](#)
 - getfwversion, [102](#)
- freebind
 - input.c, [115](#)
 - input.h, [125](#)
- freemode
 - profile.c, [279](#)
- freeprofile
 - devcmd.__unnamed__, [9](#)
 - profile.c, [280](#)
 - profile.h, [303](#)
- fwupdate
 - devcmd.__unnamed__, [9](#)
 - firmware.c, [97](#)
- get
 - devcmd.__unnamed__, [9](#)
- get_vtable
 - usb.c, [352](#)
- getfwversion
 - firmware.c, [99](#)
 - firmware.h, [102](#)
- gethwmodename
 - profile.c, [280](#)
 - profile.h, [303](#)
- gethwprofilename
 - profile.c, [281](#)
 - profile.h, [304](#)
- getid
 - profile.c, [282](#)
 - profile.h, [305](#)
- getmodename
 - profile.c, [282](#)
 - profile.h, [306](#)
- getprofilename
 - profile.c, [283](#)
 - profile.h, [306](#)
- gid
 - devnode.c, [71](#)
 - devnode.h, [81](#)
- HAS_ANY_FEATURE
 - structures.h, [332](#)
- HAS_FEATURES
 - structures.h, [333](#)
- HW_STANDARD
 - notify.c, [256](#)
- HWMODE_K70
 - structures.h, [333](#)
- HWMODE_K95
 - structures.h, [333](#)
- HWMODE_MAX
 - structures.h, [333](#)
- HWMODE_OR_RETURN
 - notify.c, [256](#)
- has_key
 - led.c, [162](#)
- hid_kb_translate
 - keymap.c, [143](#)
 - keymap.h, [154](#)
- hid_mouse_translate
 - keymap.c, [145](#)
 - keymap.h, [157](#)
- hwload
 - devcmd.__unnamed__, [9](#)
- hwload_mode
 - device.c, [31](#)
 - main.c, [254](#)
 - usb.c, [358](#)
- hwloadmode
 - profile_keyboard.c, [313](#)

- hwloadprofile
 - profile.h, [293](#)
- hwprofile, [325](#)
- hwsave
 - devcmd.__unnamed__, [9](#)
- hwtonative
 - profile.c, [284](#)
 - profile.h, [307](#)
- I_CAPS
 - structures.h, [333](#)
- I_NUM
 - structures.h, [334](#)
- I_SCROLL
 - structures.h, [334](#)
- IFACE_MAX
 - structures.h, [334](#)
- IN_CORSAIR
 - device.h, [36](#)
- IN_HID
 - device.h, [36](#)
- INDEX_OF
 - includes.h, [108](#)
- IS_CONNECTED
 - device.h, [36](#)
- IS_FULLRANGE
 - usb.h, [363](#)
- IS_K65
 - usb.h, [364](#)
- IS_K70
 - usb.h, [364](#)
- IS_K95
 - usb.h, [364](#)
- IS_M65
 - usb.h, [364](#)
- IS_MONOCHROME_DEV
 - usb.h, [365](#)
- IS_MONOCHROME
 - usb.h, [365](#)
- IS_MOUSE_DEV
 - usb.h, [365](#)
- IS_MOUSE
 - usb.h, [365](#)
- IS_MOD
 - input.h, [123](#)
- IS_RGB_DEV
 - usb.h, [366](#)
- IS_RGB
 - usb.h, [366](#)
- IS_SABRE
 - usb.h, [366](#)
- IS_SCIMITAR
 - usb.h, [366](#)
- IS_STRAFE
 - usb.h, [367](#)
- IS_WHEEL
 - input.c, [111](#)
- iauto
 - devcmd.__unnamed__, [9](#)
- id
 - devcmd.__unnamed__, [9](#)
- idle
 - devcmd.__unnamed__, [10](#)
- imutex
 - device.h, [35](#)
- includes.h, [103](#)
 - __FILE_NOPATH__, [105](#)
 - ckb_err, [105](#)
 - ckb_err_fn, [105](#)
 - ckb_err_nofile, [105](#)
 - ckb_fatal, [106](#)
 - ckb_fatal_fn, [106](#)
 - ckb_fatal_nofile, [106](#)
 - ckb_info, [106](#)
 - ckb_info_fn, [107](#)
 - ckb_info_nofile, [107](#)
 - ckb_s_err, [107](#)
 - ckb_s_out, [107](#)
 - ckb_warn, [107](#)
 - ckb_warn_fn, [108](#)
 - ckb_warn_nofile, [108](#)
 - INDEX_OF, [108](#)
 - timespec_add, [110](#)
 - timespec_eq, [108](#)
 - timespec_ge, [109](#)
 - timespec_gt, [109](#)
 - timespec_le, [109](#)
 - timespec_lt, [109](#)
 - uchar, [110](#)
 - ushort, [110](#)
- initbind
 - input.c, [116](#)
 - input.h, [125](#)
- initmode
 - profile.c, [285](#)
- inotify
 - devcmd.__unnamed__, [10](#)
- input.c, [110](#)
 - _cmd_macro, [111](#)
 - cmd_bind, [113](#)
 - cmd_macro, [114](#)
 - cmd_rebind, [114](#)
 - cmd_unbind, [115](#)
 - freebind, [115](#)
 - IS_WHEEL, [111](#)
 - initbind, [116](#)
 - inputupdate, [117](#)
 - inputupdate_keys, [118](#)
 - macromask, [120](#)
 - updateindicators_kb, [121](#)
- input.h, [122](#)
 - cmd_bind, [123](#)
 - cmd_macro, [123](#)
 - cmd_rebind, [124](#)
 - cmd_unbind, [124](#)
 - freebind, [125](#)
 - IS_MOD, [123](#)

- initbind, 125
- inputupdate, 126
- os_inputclose, 127
- os_inputopen, 128
- os_keypress, 129
- os_mousemove, 130
- os_setupindicators, 131
- updateindicators_kb, 132
- input_linux.c, 133
 - _ledthread, 133
 - isync, 134
 - os_inputclose, 134
 - os_inputopen, 135
 - os_keypress, 136
 - os_mousemove, 137
 - os_setupindicators, 138
 - uinputopen, 139
- input_mac.c, 140
- input_mac_mouse.c, 141
- inputmutex
 - device.c, 32
 - device.h, 44
- inputupdate
 - input.c, 117
 - input.h, 126
- inputupdate_keys
 - input.c, 118
- int1_int_none
 - device_vtable.c, 55
- int1_void_none
 - device_vtable.c, 55
- ioff
 - devcmd.__unnamed__, 10
- ion
 - devcmd.__unnamed__, 10
- isblack
 - led_mouse.c, 240
- iselect
 - led.c, 163
- isync
 - input_linux.c, 134
- KB_NAME_LEN
 - structures.h, 334
- KEY_BACKSLASH_ISO
 - keymap.h, 149
- KEY_CORSAIR
 - keymap.h, 149
- KEY_NONE
 - keymap.h, 149
- KEY_UNBOUND
 - keymap.h, 150
- kbsyspath
 - usb_linux.c, 435
- key, 148
- keyboard
 - device.c, 32
 - device.h, 44
- keymacro, 319
- keymap
 - keymap.c, 146
 - keymap.h, 158
- keymap.c, 141
 - BUTTON_HID_COUNT, 142
 - corsair_kbcopy, 142
 - corsair_mousecopy, 142
 - hid_kb_translate, 143
 - hid_mouse_translate, 145
 - keymap, 146
- keymap.h, 147
 - BTN_WHEELDOWN, 149
 - BTN_WHEELUP, 149
 - corsair_kbcopy, 153
 - corsair_mousecopy, 154
 - hid_kb_translate, 154
 - hid_mouse_translate, 157
 - KEY_BACKSLASH_ISO, 149
 - KEY_CORSAIR, 149
 - KEY_NONE, 149
 - KEY_UNBOUND, 150
 - keymap, 158
 - LED_DPI, 150
 - LED_MOUSE, 150
 - MOUSE_BUTTON_FIRST, 150
 - MOUSE_EXTRA_FIRST, 150
 - N_BUTTONS_EXTENDED, 151
 - N_BUTTONS_HW, 151
 - N_KEY_ZONES, 151
 - N_KEYBYTES_EXTENDED, 151
 - N_KEYBYTES_HW, 151
 - N_KEYBYTES_INPUT, 151
 - N_KEYS_EXTENDED, 152
 - N_KEYS_EXTRA, 152
 - N_KEYS_HW, 152
 - N_KEYS_INPUT, 152
 - N_MOUSE_ZONES_EXTENDED, 153
 - N_MOUSE_ZONES, 152
 - SCAN_KBD, 153
 - SCAN_MOUSE, 153
 - SCAN_SILENT, 153
- keymap_mac.h, 158
- LED_DPI
 - keymap.h, 150
- LED_MOUSE
 - keymap.h, 150
- LIFT_MAX
 - structures.h, 334
- LIFT_MIN
 - structures.h, 335
- led.c, 159
 - cmd_iauto, 159
 - cmd_inotify, 160
 - cmd_ioff, 160
 - cmd_ion, 161
 - cmd_rgb, 162
 - has_key, 162
 - iselect, 163

- printrgb, 164
- led.h, 165
 - cmd_iauto, 166
 - cmd_inotify, 167
 - cmd_ioff, 167
 - cmd_ion, 168
 - cmd_rgb, 169
 - loadrgb_kb, 169
 - loadrgb_mouse, 171
 - printrgb, 172
 - savergb_kb, 174
 - savergb_mouse, 175
 - updatergb_kb, 176
 - updatergb_mouse, 177
- led_keyboard.c, 178
 - BR1, 201
 - BR2, 201
 - BR4, 202
 - bit_reverse_table, 212
 - loadrgb_kb, 203
 - makergb_512, 205
 - makergb_full, 206
 - O0, 202
 - O1, 202
 - O2, 202
 - O3, 202
 - O4, 202
 - O5, 203
 - O6, 203
 - O7, 203
 - O8, 203
 - ordered8to3, 207
 - quantize8to3, 208
 - rgbcmp, 208
 - savergb_kb, 208
 - updatergb_kb, 210
- led_mouse.c, 240
 - isblack, 240
 - loadrgb_mouse, 241
 - rgbcmp, 242
 - savergb_mouse, 242
 - updatergb_mouse, 243
- lift
 - devcmd.__unnamed__, 10
- lighting, 322
- loaddpi
 - dpi.c, 84
 - dpi.h, 91
- loadprofile
 - devcmd.__unnamed__, 10
 - profile.c, 286
 - profile.h, 308
- loadprofile_none
 - device_vtable.c, 55
- loadrgb_kb
 - led.h, 169
 - led_keyboard.c, 203
- loadrgb_mouse
 - led.h, 171
 - led_mouse.c, 241
- localecase
 - main.c, 245
- MACRO_MAX
 - structures.h, 335
- MAX_BUFFER
 - devnode.c, 58
- MD_NAME_LEN
 - structures.h, 335
- MODE_COUNT
 - structures.h, 335
- MOUSE_BUTTON_FIRST
 - keymap.h, 150
- MOUSE_EXTRA_FIRST
 - keymap.h, 150
- MSG_SIZE
 - structures.h, 335
- macro
 - devcmd.__unnamed__, 10
- macroaction, 319
- macromask
 - input.c, 120
- main
 - main.c, 246
- main.c, 244
 - features_mask, 254
 - hwload_mode, 254
 - localecase, 245
 - main, 246
 - main_ac, 254
 - main_av, 254
 - quit, 249
 - quitWithLock, 250
 - reset_stop, 254
 - restart, 251
 - sighandler, 252
 - sighandler2, 253
 - timespec_add, 253
- main_ac
 - main.c, 254
- main_av
 - main.c, 254
- makergb_512
 - led_keyboard.c, 205
- makergb_full
 - led_keyboard.c, 206
- mkdevpath
 - devnode.c, 63
 - devnode.h, 74
- mkfwnode
 - devnode.c, 64
 - devnode.h, 75
- mknotifynode
 - devnode.c, 65
 - devnode.h, 76
- models
 - usb_linux.c, 435

- N_BUTTONS_EXTENDED
 - keymap.h, [151](#)
- N_BUTTONS_HW
 - keymap.h, [151](#)
- N_KEY_ZONES
 - keymap.h, [151](#)
- N_KEYBYTES_EXTENDED
 - keymap.h, [151](#)
- N_KEYBYTES_HW
 - keymap.h, [151](#)
- N_KEYBYTES_INPUT
 - keymap.h, [151](#)
- N_KEYS_EXTENDED
 - keymap.h, [152](#)
- N_KEYS_EXTRA
 - keymap.h, [152](#)
- N_KEYS_HW
 - keymap.h, [152](#)
- N_KEYS_INPUT
 - keymap.h, [152](#)
- N_MODELS
 - usb_linux.c, [410](#)
- N_MOUSE_ZONES_EXTENDED
 - keymap.h, [153](#)
- N_MOUSE_ZONES
 - keymap.h, [152](#)
- NEEDS_FW_UPDATE
 - structures.h, [336](#)
- NK95_HWOFF
 - usb.h, [367](#)
- NK95_HWON
 - usb.h, [367](#)
- NK95_M1
 - usb.h, [367](#)
- NK95_M2
 - usb.h, [368](#)
- NK95_M3
 - usb.h, [368](#)
- name
 - devcmd.__unnamed__, [10](#)
- nativetohw
 - profile.c, [286](#)
 - profile.h, [308](#)
- nk95cmd
 - usb.h, [368](#)
- notify
 - devcmd.__unnamed__, [10](#)
- notify.c, [255](#)
 - _cmd_get, [256](#)
 - cmd_get, [259](#)
 - cmd_notify, [260](#)
 - cmd_restart, [260](#)
 - HW_STANDARD, [256](#)
 - HWMODE_OR_RETURN, [256](#)
 - nprintf, [261](#)
 - nprintind, [262](#)
 - nprintkey, [263](#)
 - restart, [264](#)
 - notify.h, [265](#)
 - cmd_get, [266](#)
 - cmd_notify, [266](#)
 - cmd_restart, [267](#)
 - nprintf, [267](#)
 - nprintind, [268](#)
 - nprintkey, [269](#)
- nprintf
 - notify.c, [261](#)
 - notify.h, [267](#)
- nprintind
 - notify.c, [262](#)
 - notify.h, [268](#)
- nprintkey
 - notify.c, [263](#)
 - notify.h, [269](#)
- O0
 - led_keyboard.c, [202](#)
- O1
 - led_keyboard.c, [202](#)
- O2
 - led_keyboard.c, [202](#)
- O3
 - led_keyboard.c, [202](#)
- O4
 - led_keyboard.c, [202](#)
- O5
 - led_keyboard.c, [203](#)
- O6
 - led_keyboard.c, [203](#)
- O7
 - led_keyboard.c, [203](#)
- O8
 - led_keyboard.c, [203](#)
- OUTFIFO_MAX
 - structures.h, [336](#)
- ordered8to3
 - led_keyboard.c, [207](#)
- os.h, [270](#)
 - _DEFAULT_SOURCE, [271](#)
 - _GNU_SOURCE, [271](#)
 - euid_guard_start, [271](#)
 - euid_guard_stop, [271](#)
 - UINPUT_VERSION, [272](#)
- os_closeusb
 - usb.h, [387](#)
 - usb_linux.c, [412](#)
- os_inputclose
 - input.h, [127](#)
 - input_linux.c, [134](#)
- os_inputmain
 - usb.h, [388](#)
 - usb_linux.c, [413](#)
- os_inputopen
 - input.h, [128](#)
 - input_linux.c, [135](#)
- os_keypress
 - input.h, [129](#)

- input_linux.c, [136](#)
- os_mousemove
 - input.h, [130](#)
 - input_linux.c, [137](#)
- os_resetusb
 - usb.h, [391](#)
 - usb_linux.c, [416](#)
- os_sendindicators
 - usb.h, [393](#)
 - usb_linux.c, [417](#)
- os_setupindicators
 - input.h, [131](#)
 - input_linux.c, [138](#)
- os_setupusb
 - usb.h, [394](#)
 - usb_linux.c, [418](#)
- os_usbrecv
 - usb.h, [395](#)
 - usb_linux.c, [420](#)
- os_usbsend
 - usb.h, [397](#)
 - usb_linux.c, [421](#)
- P_HARPOON_STR
 - usb.h, [368](#)
- P_HARPOON
 - usb.h, [368](#)
- P_K65
 - usb.h, [369](#)
- P_K65_LUX_STR
 - usb.h, [369](#)
- P_K65_LUX
 - usb.h, [369](#)
- P_K65_NRGB_STR
 - usb.h, [369](#)
- P_K65_NRGB
 - usb.h, [369](#)
- P_K65_RFIRE_STR
 - usb.h, [370](#)
- P_K65_RFIRE
 - usb.h, [370](#)
- P_K65_STR
 - usb.h, [370](#)
- P_K70
 - usb.h, [370](#)
- P_K70_LUX_NRGB_STR
 - usb.h, [371](#)
- P_K70_LUX_NRGB
 - usb.h, [370](#)
- P_K70_LUX_STR
 - usb.h, [371](#)
- P_K70_LUX
 - usb.h, [370](#)
- P_K70_NRGB_STR
 - usb.h, [371](#)
- P_K70_NRGB
 - usb.h, [371](#)
- P_K70_RFIRE_NRGB_STR
 - usb.h, [372](#)
- P_K70_RFIRE_NRGB
 - usb.h, [371](#)
- P_K70_RFIRE_STR
 - usb.h, [372](#)
- P_K70_RFIRE
 - usb.h, [371](#)
- P_K70_STR
 - usb.h, [372](#)
- P_K95
 - usb.h, [372](#)
- P_K95_NRGB_STR
 - usb.h, [372](#)
- P_K95_NRGB
 - usb.h, [372](#)
- P_K95_PLATINUM_STR
 - usb.h, [373](#)
- P_K95_PLATINUM
 - usb.h, [373](#)
- P_K95_STR
 - usb.h, [373](#)
- P_M65
 - usb.h, [373](#)
- P_M65_PRO_STR
 - usb.h, [373](#)
- P_M65_PRO
 - usb.h, [373](#)
- P_M65_STR
 - usb.h, [374](#)
- P_SABRE_L_STR
 - usb.h, [374](#)
- P_SABRE_N_STR
 - usb.h, [374](#)
- P_SABRE_O2
 - usb.h, [375](#)
- P_SABRE_O2_STR
 - usb.h, [375](#)
- P_SABRE_O_STR
 - usb.h, [375](#)
- P_SABRE_L
 - usb.h, [374](#)
- P_SABRE_N
 - usb.h, [374](#)
- P_SABRE_O
 - usb.h, [374](#)
- P_SCIMITAR_PRO_STR
 - usb.h, [376](#)
- P_SCIMITAR_PRO
 - usb.h, [375](#)
- P_SCIMITAR_STR
 - usb.h, [376](#)
- P_SCIMITAR
 - usb.h, [375](#)
- P_STRAFE_NRGB_STR
 - usb.h, [376](#)
- P_STRAFE_NRGB
 - usb.h, [376](#)
- P_STRAFE_STR
 - usb.h, [377](#)

P_STRAFE
 usb.h, 376

PR_NAME_LEN
 structures.h, 336

pollrate
 devcmd.__unnamed__, 10

printdpi
 dpi.c, 86
 dpi.h, 92

printname
 profile.c, 287

printrgb
 led.c, 164
 led.h, 172

product_str
 usb.c, 352
 usb.h, 399

profile.c, 272
 _freeprofile, 273
 allocprofile, 274
 cmd_erase, 275
 cmd_eraseprofile, 276
 cmd_id, 276
 cmd_name, 277
 cmd_profileid, 277
 cmd_profilename, 278
 freemode, 279
 freeprofile, 280
 gethwmodename, 280
 gethwprofilename, 281
 getid, 282
 getmodename, 282
 getprofilename, 283
 hwtonative, 284
 initmode, 285
 loadprofile, 286
 nativetohw, 286
 printname, 287
 setid, 288
 u16dec, 288
 u16enc, 289
 urldecode2, 290
 urlencode2, 291
 utf16to8, 292
 utf8to16, 292

profile.h, 292
 allocprofile, 293
 cmd_erase, 294
 cmd_eraseprofile, 295
 cmd_hwload_kb, 296
 cmd_hwload_mouse, 297
 cmd_hwsave_kb, 298
 cmd_hwsave_mouse, 299
 cmd_id, 300
 cmd_name, 301
 cmd_profileid, 301
 cmd_profilename, 302
 freeprofile, 303
 gethwmodename, 303
 gethwprofilename, 304
 getid, 305
 getmodename, 306
 getprofilename, 306
 hwloadprofile, 293
 hwtonative, 307
 loadprofile, 308
 nativetohw, 308
 setid, 309

profile_keyboard.c, 310
 cmd_hwload_kb, 311
 cmd_hwsave_kb, 312
 hwloadmode, 313

profile_mouse.c, 314
 cmd_hwload_mouse, 314
 cmd_hwsave_mouse, 315

profileid
 devcmd.__unnamed__, 11

profilename
 devcmd.__unnamed__, 11

quantize8to3
 led_keyboard.c, 208

quit
 main.c, 249

quitWithLock
 main.c, 250

readcmd
 command.c, 14
 command.h, 24

readlines
 devnode.c, 66
 devnode.h, 77

readlines_ctx
 devnode.h, 74

readlines_ctx_free
 devnode.c, 67
 devnode.h, 78

readlines_ctx_init
 devnode.c, 67
 devnode.h, 78

rebind
 devcmd.__unnamed__, 11

reset_stop
 main.c, 254
 usb.c, 358

resetusb
 usb.h, 377

restart
 devcmd.__unnamed__, 11
 main.c, 251
 notify.c, 264

revertusb
 usb.c, 353
 usb.h, 400

rgb
 devcmd.__unnamed__, 11

- rgbcmp
 - led_keyboard.c, 208
 - led_mouse.c, 242
- rm_recursive
 - devnode.c, 68
- rmdevpath
 - devnode.c, 68
 - devnode.h, 78
- rmnotifynode
 - devnode.c, 69
 - devnode.h, 79
- S_CUSTOM_R
 - devnode.h, 73
- S_CUSTOM
 - devnode.h, 73
- S_GID_READ
 - devnode.c, 58
- S_READDIR
 - devnode.h, 73
- S_READWRITE
 - devnode.h, 74
- S_READ
 - devnode.h, 73
- SCAN_KBD
 - keymap.h, 153
- SCAN_MOUSE
 - keymap.h, 153
- SCAN_SILENT
 - keymap.h, 153
- SCROLL_ACCELERATED
 - structures.h, 336
- SCROLL_MAX
 - structures.h, 336
- SCROLL_MIN
 - structures.h, 337
- SERIAL_LEN
 - structures.h, 337
- SET_KEYBIT
 - structures.h, 337
- savedpi
 - dpi.c, 86
 - dpi.h, 93
- savergb_kb
 - led.h, 174
 - led_keyboard.c, 208
- savergb_mouse
 - led.h, 175
 - led_mouse.c, 242
- setactive
 - device.h, 36
- setactive_kb
 - device.h, 39
 - device_keyboard.c, 46
- setactive_mouse
 - device.h, 41
 - device_mouse.c, 51
- setid
 - profile.c, 288
- profile.h, 309
- setmodeindex
 - devcmd.__unnamed__, 11
- setmodeindex_nrgb
 - device.h, 43
 - device_keyboard.c, 48
- setupusb
 - usb.c, 354
 - usb.h, 401
- sighandler
 - main.c, 252
- sighandler2
 - main.c, 253
- snap
 - devcmd.__unnamed__, 11
- start
 - devcmd.__unnamed__, 11
- start_dev
 - device.c, 31
 - device.h, 43
- start_kb_nrgb
 - device.h, 44
 - device_keyboard.c, 49
- strtrim
 - usb_linux.c, 423
- structures.h, 316
 - CLEAR_KEYBIT, 329
 - DPI_COUNT, 329
 - FEAT_ADJRATE, 329
 - FEAT_ANSI, 329
 - FEAT_BIND, 330
 - FEAT_COMMON, 330
 - FEAT_FWUPDATE, 330
 - FEAT_FWVERSION, 330
 - FEAT_HWLOAD, 330
 - FEAT_ISO, 331
 - FEAT_LMASK, 331
 - FEAT_MONOCHROME, 331
 - FEAT_MOUSEACCEL, 331
 - FEAT_NOTIFY, 331
 - FEAT_POLLRATE, 332
 - FEAT_RGB, 332
 - FEAT_STD_NRGB, 332
 - FEAT_STD_RGB, 332
 - HAS_ANY_FEATURE, 332
 - HAS_FEATURES, 333
 - HWMODE_K70, 333
 - HWMODE_K95, 333
 - HWMODE_MAX, 333
 - I_CAPS, 333
 - I_NUM, 334
 - I_SCROLL, 334
 - IFACE_MAX, 334
 - KB_NAME_LEN, 334
 - LIFT_MAX, 334
 - LIFT_MIN, 335
 - MACRO_MAX, 335
 - MD_NAME_LEN, 335

- MODE_COUNT, 335
- MSG_SIZE, 335
- NEEDS_FW_UPDATE, 336
- OUTFIFO_MAX, 336
- PR_NAME_LEN, 336
- SCROLL_ACCELERATED, 336
- SCROLL_MAX, 336
- SCROLL_MIN, 337
- SERIAL_LEN, 337
- SET_KEYBIT, 337
- vtable_keyboard, 337
- vtable_keyboard_nonrgb, 337
- vtable_mouse, 338
- TEST_RESET
 - usb_linux.c, 410
- TRY_WITH_RESET
 - command.c, 14
- timespec_add
 - includes.h, 110
 - main.c, 253
- timespec_eq
 - includes.h, 108
- timespec_ge
 - includes.h, 109
- timespec_gt
 - includes.h, 109
- timespec_le
 - includes.h, 109
- timespec_lt
 - includes.h, 109
- u16dec
 - profile.c, 288
- u16enc
 - profile.c, 289
- UINPUT_VERSION
 - os.h, 272
- USB_DELAY_DEFAULT
 - usb.h, 377
- uchar
 - includes.h, 110
- udev
 - usb_linux.c, 435
- udev_enum
 - usb_linux.c, 424
- udevthread
 - usb_linux.c, 436
- uinputopen
 - input_linux.c, 139
- unbind
 - devcmd.__unnamed__, 11
- updateconnected
 - devnode.c, 70
 - devnode.h, 80
- updatedpi
 - devcmd.__unnamed__, 11
 - dpi.c, 87
 - dpi.h, 94
- updateindicators
 - devcmd.__unnamed__, 11
- updateindicators_kb
 - input.c, 121
 - input.h, 132
- updatergb
 - devcmd.__unnamed__, 11
- updatergb_kb
 - led.h, 176
 - led_keyboard.c, 210
- updatergb_mouse
 - led.h, 177
 - led_mouse.c, 243
- urldecode2
 - profile.c, 290
- urlencode2
 - profile.c, 291
- usb.c, 338
 - _resetusb, 339
 - _setupusb, 340
 - _usbrecv, 344
 - _usbseend, 346
 - closeusb, 347
 - devmain, 349
 - features_mask, 358
 - get_vtable, 352
 - hwload_mode, 358
 - product_str, 352
 - reset_stop, 358
 - revertusb, 353
 - setupusb, 354
 - usb_tryreset, 356
 - usbmutex, 358
 - vendor_str, 357
- usb.h, 359
 - _nk95cmd, 379
 - _resetusb, 380
 - _usbrecv, 381
 - _usbseend, 383
 - closeusb, 385
 - DELAY_LONG, 363
 - DELAY_MEDIUM, 363
 - DELAY_SHORT, 363
 - IS_FULLRANGE, 363
 - IS_K65, 364
 - IS_K70, 364
 - IS_K95, 364
 - IS_M65, 364
 - IS_MONOCHROME_DEV, 365
 - IS_MONOCHROME, 365
 - IS_MOUSE_DEV, 365
 - IS_MOUSE, 365
 - IS_RGB_DEV, 366
 - IS_RGB, 366
 - IS_SABRE, 366
 - IS_SCIMITAR, 366
 - IS_STRAFE, 367
 - NK95_HWOFF, 367

NK95_HWON, [367](#)
 NK95_M1, [367](#)
 NK95_M2, [368](#)
 NK95_M3, [368](#)
 nk95cmd, [368](#)
 os_closeusb, [387](#)
 os_inputmain, [388](#)
 os_resetusb, [391](#)
 os_sendindicators, [393](#)
 os_setupusb, [394](#)
 os_usbrecv, [395](#)
 os_usbsend, [397](#)
 P_HARPOON_STR, [368](#)
 P_HARPOON, [368](#)
 P_K65, [369](#)
 P_K65_LUX_STR, [369](#)
 P_K65_LUX, [369](#)
 P_K65_NRGB_STR, [369](#)
 P_K65_NRGB, [369](#)
 P_K65_RFIRE_STR, [370](#)
 P_K65_RFIRE, [370](#)
 P_K65_STR, [370](#)
 P_K70, [370](#)
 P_K70_LUX_NRGB_STR, [371](#)
 P_K70_LUX_NRGB, [370](#)
 P_K70_LUX_STR, [371](#)
 P_K70_LUX, [370](#)
 P_K70_NRGB_STR, [371](#)
 P_K70_NRGB, [371](#)
 P_K70_RFIRE_NRGB_STR, [372](#)
 P_K70_RFIRE_NRGB, [371](#)
 P_K70_RFIRE_STR, [372](#)
 P_K70_RFIRE, [371](#)
 P_K70_STR, [372](#)
 P_K95, [372](#)
 P_K95_NRGB_STR, [372](#)
 P_K95_NRGB, [372](#)
 P_K95_PLATINUM_STR, [373](#)
 P_K95_PLATINUM, [373](#)
 P_K95_STR, [373](#)
 P_M65, [373](#)
 P_M65_PRO_STR, [373](#)
 P_M65_PRO, [373](#)
 P_M65_STR, [374](#)
 P_SABRE_L_STR, [374](#)
 P_SABRE_N_STR, [374](#)
 P_SABRE_O2, [375](#)
 P_SABRE_O2_STR, [375](#)
 P_SABRE_O_STR, [375](#)
 P_SABRE_L, [374](#)
 P_SABRE_N, [374](#)
 P_SABRE_O, [374](#)
 P_SCIMITAR_PRO_STR, [376](#)
 P_SCIMITAR_PRO, [375](#)
 P_SCIMITAR_STR, [376](#)
 P_SCIMITAR, [375](#)
 P_STRAFE_NRGB_STR, [376](#)
 P_STRAFE_NRGB, [376](#)
 P_STRAFE_STR, [377](#)
 P_STRAFE, [376](#)
 product_str, [399](#)
 resetusb, [377](#)
 revertusb, [400](#)
 setupusb, [401](#)
 USB_DELAY_DEFAULT, [377](#)
 usb_tryreset, [403](#)
 usbkill, [404](#)
 usbmain, [405](#)
 usbrecv, [377](#)
 usbsend, [378](#)
 V_CORSAIR_STR, [378](#)
 V_CORSAIR, [378](#)
 vendor_str, [407](#)
 usb_add_device
 usb_linux.c, [425](#)
 usb_linux.c, [408](#)
 _nk95cmd, [411](#)
 kbsyspath, [435](#)
 models, [435](#)
 N_MODELS, [410](#)
 os_closeusb, [412](#)
 os_inputmain, [413](#)
 os_resetusb, [416](#)
 os_sendindicators, [417](#)
 os_setupusb, [418](#)
 os_usbrecv, [420](#)
 os_usbsend, [421](#)
 strtrim, [423](#)
 TEST_RESET, [410](#)
 udev, [435](#)
 udev_enum, [424](#)
 udevthread, [436](#)
 usb_add_device, [425](#)
 usb_rm_device, [427](#)
 usbadd, [428](#)
 usbclaim, [430](#)
 usbkill, [431](#)
 usbmain, [432](#)
 usbthread, [436](#)
 usbunclaim, [434](#)
 usb_mac.c, [436](#)
 usb_rm_device
 usb_linux.c, [427](#)
 usb_tryreset
 usb.c, [356](#)
 usb.h, [403](#)
 usbadd
 usb_linux.c, [428](#)
 usbclaim
 usb_linux.c, [430](#)
 usbdevice, [327](#)
 usbid, [318](#)
 usbinput, [326](#)
 usbkill
 usb.h, [404](#)
 usb_linux.c, [431](#)

- usbmain
 - usb.h, [405](#)
 - usb_linux.c, [432](#)
- usbmode, [323](#)
- usbmutex
 - usb.c, [358](#)
- usbprofile, [324](#)
- usbrecv
 - usb.h, [377](#)
- usbsend
 - usb.h, [378](#)
- usbthread
 - usb_linux.c, [436](#)
- usbunclaim
 - usb_linux.c, [434](#)
- ushort
 - includes.h, [110](#)
- utf16to8
 - profile.c, [292](#)
- utf8to16
 - profile.c, [292](#)
- V_CORSAIR_STR
 - usb.h, [378](#)
- V_CORSAIR
 - usb.h, [378](#)
- vendor_str
 - usb.c, [357](#)
 - usb.h, [407](#)
- vtable_keyboard
 - device_vtable.c, [56](#)
 - structures.h, [337](#)
- vtable_keyboard_nonrgb
 - device_vtable.c, [56](#)
 - structures.h, [337](#)
- vtable_mouse
 - device_vtable.c, [56](#)
 - structures.h, [338](#)