

ckb-next
0.2.7

Generated by Doxygen 1.8.6

Thu Apr 27 2017 21:48:46

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	8
4.1.2.5	dpisel	8
4.1.2.6	erase	8
4.1.2.7	eraseprofile	8
4.1.2.8	freeprofile	8
4.1.2.9	fwupdate	8
4.1.2.10	get	8
4.1.2.11	hwload	8
4.1.2.12	hwsave	9
4.1.2.13	iauto	9
4.1.2.14	id	9
4.1.2.15	idle	9
4.1.2.16	inotify	9
4.1.2.17	ioff	9
4.1.2.18	ion	9
4.1.2.19	lift	9
4.1.2.20	loadprofile	9

4.1.2.21	macro	9
4.1.2.22	name	9
4.1.2.23	notify	9
4.1.2.24	pollrate	9
4.1.2.25	profileid	9
4.1.2.26	profilename	9
4.1.2.27	rebind	9
4.1.2.28	restart	9
4.1.2.29	rgb	9
4.1.2.30	setmodeindex	9
4.1.2.31	snap	9
4.1.2.32	start	9
4.1.2.33	unbind	9
4.1.2.34	updatedpi	9
4.1.2.35	updateindicators	9
4.1.2.36	updatergb	9
5	File Documentation	11
5.1	src/ckb-daemon/command.c File Reference	11
5.1.1	Macro Definition Documentation	11
5.1.1.1	TRY_WITH_RESET	12
5.1.2	Function Documentation	12
5.1.2.1	readcmd	12
5.1.3	Variable Documentation	16
5.1.3.1	cmd_strings	16
5.2	src/ckb-daemon/command.h File Reference	16
5.2.1	Data Structure Documentation	17
5.2.1.1	union devcmd	17
5.2.2	Macro Definition Documentation	18
5.2.2.1	CMD_COUNT	18
5.2.2.2	CMD_DEV_COUNT	18
5.2.3	Typedef Documentation	18
5.2.3.1	cmdhandler	19
5.2.3.2	cmdhandler_io	19
5.2.3.3	cmdhandler_mac	19
5.2.3.4	devcmd	19
5.2.4	Enumeration Type Documentation	19
5.2.4.1	cmd	19
5.2.5	Function Documentation	21
5.2.5.1	readcmd	21

5.3	src/ckb-daemon/device.c File Reference	25
5.3.1	Function Documentation	26
5.3.1.1	_start_dev	26
5.3.1.2	start_dev	27
5.3.2	Variable Documentation	28
5.3.2.1	devlistmutex	28
5.3.2.2	devmutex	28
5.3.2.3	hwload_mode	28
5.3.2.4	inputmutex	28
5.3.2.5	keyboard	28
5.4	src/ckb-daemon/device.h File Reference	28
5.4.1	Macro Definition Documentation	30
5.4.1.1	ACT_LIGHT	30
5.4.1.2	ACT_LOCK	30
5.4.1.3	ACT_M1	30
5.4.1.4	ACT_M2	30
5.4.1.5	ACT_M3	30
5.4.1.6	ACT_MR_RING	30
5.4.1.7	ACT_NEXT	30
5.4.1.8	ACT_NEXT_NOWRAP	30
5.4.1.9	DEV_MAX	31
5.4.1.10	dmutex	31
5.4.1.11	imutex	31
5.4.1.12	IN_CORSAIR	31
5.4.1.13	IN_HID	31
5.4.1.14	IS_CONNECTED	31
5.4.1.15	setactive	31
5.4.2	Function Documentation	31
5.4.2.1	cmd_active_kb	31
5.4.2.2	cmd_active_mouse	32
5.4.2.3	cmd_idle_kb	32
5.4.2.4	cmd_idle_mouse	33
5.4.2.5	cmd_pollrate	33
5.4.2.6	setactive_kb	33
5.4.2.7	setactive_mouse	35
5.4.2.8	setmodeindex_nrgb	36
5.4.2.9	start_dev	36
5.4.2.10	start_kb_nrgb	37
5.4.3	Variable Documentation	37
5.4.3.1	devmutex	37

5.4.3.2	inputmutex	37
5.4.3.3	keyboard	37
5.5	src/ckb-daemon/device_keyboard.c File Reference	38
5.5.1	Function Documentation	38
5.5.1.1	cmd_active_kb	38
5.5.1.2	cmd_idle_kb	39
5.5.1.3	setactive_kb	39
5.5.1.4	setmodeindex_nrgb	41
5.5.1.5	start_kb_nrgb	41
5.6	src/ckb-daemon/device_mouse.c File Reference	41
5.6.1	Function Documentation	42
5.6.1.1	cmd_active_mouse	42
5.6.1.2	cmd_idle_mouse	42
5.6.1.3	cmd_pollrate	43
5.6.1.4	setactive_mouse	43
5.7	src/ckb-daemon/device_vtable.c File Reference	44
5.7.1	Function Documentation	45
5.7.1.1	cmd_io_none	45
5.7.1.2	cmd_macro_none	45
5.7.1.3	cmd_none	46
5.7.1.4	int1_int_none	46
5.7.1.5	int1_void_none	46
5.7.1.6	loadprofile_none	46
5.7.2	Variable Documentation	46
5.7.2.1	vtable_keyboard	46
5.7.2.2	vtable_keyboard_nonrgb	46
5.7.2.3	vtable_mouse	46
5.8	src/ckb-daemon/devnode.c File Reference	47
5.8.1	Data Structure Documentation	48
5.8.1.1	struct_readlines_ctx	48
5.8.2	Macro Definition Documentation	48
5.8.2.1	MAX_BUFFER	48
5.8.2.2	S_GID_READ	49
5.8.3	Function Documentation	49
5.8.3.1	_mkdevpath	49
5.8.3.2	_mknotifynode	51
5.8.3.3	_rmnotifynode	52
5.8.3.4	_updateconnected	52
5.8.3.5	mkdevpath	53
5.8.3.6	mkfwnode	54

5.8.3.7	mknotifynode	55
5.8.3.8	readlines	55
5.8.3.9	readlines_ctx_free	56
5.8.3.10	readlines_ctx_init	57
5.8.3.11	rm_recursive	57
5.8.3.12	rmdevpath	58
5.8.3.13	rmnotifynode	58
5.8.3.14	updateconnected	59
5.8.4	Variable Documentation	60
5.8.4.1	devpath	60
5.8.4.2	gid	60
5.9	src/ckb-daemon/devnode.h File Reference	60
5.9.1	Macro Definition Documentation	61
5.9.1.1	S_CUSTOM	61
5.9.1.2	S_CUSTOM_R	61
5.9.1.3	S_READ	61
5.9.1.4	S_READDIR	61
5.9.1.5	S_READWRITE	61
5.9.2	Typedef Documentation	62
5.9.2.1	readlines_ctx	62
5.9.3	Function Documentation	62
5.9.3.1	mkdevpath	62
5.9.3.2	mkfwnode	63
5.9.3.3	mknotifynode	63
5.9.3.4	readlines	64
5.9.3.5	readlines_ctx_free	65
5.9.3.6	readlines_ctx_init	65
5.9.3.7	rmdevpath	66
5.9.3.8	rmnotifynode	66
5.9.3.9	updateconnected	67
5.9.4	Variable Documentation	68
5.9.4.1	devpath	68
5.9.4.2	gid	68
5.10	src/ckb-daemon/dpi.c File Reference	68
5.10.1	Function Documentation	68
5.10.1.1	cmd_dpi	68
5.10.1.2	cmd_dpisel	69
5.10.1.3	cmd_lift	69
5.10.1.4	cmd_snap	69
5.10.1.5	loaddpi	70

5.10.1.6	printdpi	71
5.10.1.7	savedpi	71
5.10.1.8	updatedpi	72
5.11	src/ckb-daemon/dpi.h File Reference	73
5.11.1	Function Documentation	73
5.11.1.1	cmd_dpi	73
5.11.1.2	cmd_dpisel	74
5.11.1.3	cmd_lift	74
5.11.1.4	cmd_snap	75
5.11.1.5	loaddpi	75
5.11.1.6	printdpi	76
5.11.1.7	savedpi	76
5.11.1.8	updatedpi	77
5.12	src/ckb-daemon/extra_mac.c File Reference	78
5.13	src/ckb-daemon/firmware.c File Reference	78
5.13.1	Macro Definition Documentation	79
5.13.1.1	FW_MAXSIZE	79
5.13.1.2	FW_NOFILE	79
5.13.1.3	FW_OK	79
5.13.1.4	FW_USBFAIL	79
5.13.1.5	FW_WRONGDEV	79
5.13.2	Function Documentation	79
5.13.2.1	cmd_fwupdate	79
5.13.2.2	fwupdate	80
5.13.2.3	getfwversion	82
5.14	src/ckb-daemon/firmware.h File Reference	83
5.14.1	Function Documentation	83
5.14.1.1	cmd_fwupdate	84
5.14.1.2	getfwversion	84
5.15	src/ckb-daemon/includes.h File Reference	85
5.15.1	Macro Definition Documentation	87
5.15.1.1	__FILE_NOPATH__	87
5.15.1.2	ckb_err	87
5.15.1.3	ckb_err_fn	87
5.15.1.4	ckb_err_nofile	87
5.15.1.5	ckb_fatal	87
5.15.1.6	ckb_fatal_fn	87
5.15.1.7	ckb_fatal_nofile	87
5.15.1.8	ckb_info	87
5.15.1.9	ckb_info_fn	88

5.15.1.10	ckb_info_nofile	88
5.15.1.11	ckb_s_err	88
5.15.1.12	ckb_s_out	88
5.15.1.13	ckb_warn	88
5.15.1.14	ckb_warn_fn	88
5.15.1.15	ckb_warn_nofile	88
5.15.1.16	INDEX_OF	88
5.15.1.17	timespec_eq	88
5.15.1.18	timespec_ge	89
5.15.1.19	timespec_gt	89
5.15.1.20	timespec_le	89
5.15.1.21	timespec_lt	89
5.15.2	Typedef Documentation	89
5.15.2.1	uchar	89
5.15.2.2	ushort	89
5.15.3	Function Documentation	89
5.15.3.1	timespec_add	89
5.16	src/ckb-daemon/input.c File Reference	89
5.16.1	Macro Definition Documentation	90
5.16.1.1	IS_WHEEL	90
5.16.2	Function Documentation	90
5.16.2.1	_cmd_macro	90
5.16.2.2	cmd_bind	92
5.16.2.3	cmd_macro	92
5.16.2.4	cmd_rebind	93
5.16.2.5	cmd_unbind	93
5.16.2.6	freebind	93
5.16.2.7	initbind	94
5.16.2.8	inputupdate	94
5.16.2.9	inputupdate_keys	95
5.16.2.10	macromask	97
5.16.2.11	updateindicators_kb	98
5.17	src/ckb-daemon/input.h File Reference	98
5.17.1	Macro Definition Documentation	99
5.17.1.1	IS_MOD	99
5.17.2	Function Documentation	100
5.17.2.1	cmd_bind	100
5.17.2.2	cmd_macro	100
5.17.2.3	cmd_rebind	100
5.17.2.4	cmd_unbind	101

5.17.2.5	freebind	101
5.17.2.6	initbind	101
5.17.2.7	inputupdate	102
5.17.2.8	os_inputclose	103
5.17.2.9	os_inputopen	103
5.17.2.10	os_keypress	104
5.17.2.11	os_mousemove	105
5.17.2.12	os_setupindicators	106
5.17.2.13	updateindicators_kb	107
5.18	src/ckb-daemon/input_linux.c File Reference	107
5.18.1	Function Documentation	108
5.18.1.1	_ledthread	108
5.18.1.2	isync	109
5.18.1.3	os_inputclose	109
5.18.1.4	os_inputopen	110
5.18.1.5	os_keypress	111
5.18.1.6	os_mousemove	112
5.18.1.7	os_setupindicators	112
5.18.1.8	uinputopen	113
5.19	src/ckb-daemon/input_mac.c File Reference	114
5.20	src/ckb-daemon/input_mac_mouse.c File Reference	114
5.21	src/ckb-daemon/keymap.c File Reference	114
5.21.1	Macro Definition Documentation	115
5.21.1.1	BUTTON_HID_COUNT	115
5.21.2	Function Documentation	115
5.21.2.1	corsair_kbcopy	115
5.21.2.2	corsair_mousecopy	116
5.21.2.3	hid_kb_translate	116
5.21.2.4	hid_mouse_translate	118
5.21.3	Variable Documentation	119
5.21.3.1	keymap	119
5.22	src/ckb-daemon/keymap.h File Reference	119
5.22.1	Data Structure Documentation	120
5.22.1.1	struct key	120
5.22.2	Macro Definition Documentation	121
5.22.2.1	BTN_WHEELDOWN	121
5.22.2.2	BTN_WHEELUP	121
5.22.2.3	KEY_BACKSLASH_ISO	121
5.22.2.4	KEY_CORSAIR	121
5.22.2.5	KEY_NONE	122

5.22.2.6	KEY_UNBOUND	122
5.22.2.7	LED_DPI	122
5.22.2.8	LED_MOUSE	122
5.22.2.9	MOUSE_BUTTON_FIRST	122
5.22.2.10	MOUSE_EXTRA_FIRST	122
5.22.2.11	N_BUTTONS_EXTENDED	122
5.22.2.12	N_BUTTONS_HW	122
5.22.2.13	N_KEY_ZONES	122
5.22.2.14	N_KEYBYTES_EXTENDED	122
5.22.2.15	N_KEYBYTES_HW	123
5.22.2.16	N_KEYBYTES_INPUT	123
5.22.2.17	N_KEYS_EXTENDED	123
5.22.2.18	N_KEYS_EXTRA	123
5.22.2.19	N_KEYS_HW	123
5.22.2.20	N_KEYS_INPUT	123
5.22.2.21	N_MOUSE_ZONES	123
5.22.2.22	N_MOUSE_ZONES_EXTENDED	123
5.22.2.23	SCAN_KBD	123
5.22.2.24	SCAN_MOUSE	123
5.22.2.25	SCAN_SILENT	124
5.22.3	Function Documentation	124
5.22.3.1	corsair_kbcopy	124
5.22.3.2	corsair_mousecopy	124
5.22.3.3	hid_kb_translate	125
5.22.3.4	hid_mouse_translate	126
5.22.4	Variable Documentation	127
5.22.4.1	keymap	127
5.23	src/ckb-daemon/keymap_mac.h File Reference	127
5.24	src/ckb-daemon/led.c File Reference	128
5.24.1	Function Documentation	129
5.24.1.1	cmd_iauto	129
5.24.1.2	cmd_inotify	129
5.24.1.3	cmd_ioff	130
5.24.1.4	cmd_ion	130
5.24.1.5	cmd_rgb	130
5.24.1.6	has_key	131
5.24.1.7	iselect	131
5.24.1.8	printrgb	132
5.25	src/ckb-daemon/led.h File Reference	134
5.25.1	Function Documentation	134

5.25.1.1	<code>cmd_iauto</code>	134
5.25.1.2	<code>cmd_inotify</code>	135
5.25.1.3	<code>cmd_ioff</code>	135
5.25.1.4	<code>cmd_ion</code>	136
5.25.1.5	<code>cmd_rgb</code>	136
5.25.1.6	<code>loadrgb_kb</code>	137
5.25.1.7	<code>loadrgb_mouse</code>	138
5.25.1.8	<code>printrgb</code>	139
5.25.1.9	<code>savergb_kb</code>	140
5.25.1.10	<code>savergb_mouse</code>	142
5.25.1.11	<code>updatergb_kb</code>	142
5.25.1.12	<code>updatergb_mouse</code>	144
5.26	<code>src/ckb-daemon/led_keyboard.c</code> File Reference	145
5.26.1	Macro Definition Documentation	168
5.26.1.1	<code>BR1</code>	168
5.26.1.2	<code>BR2</code>	168
5.26.1.3	<code>BR4</code>	168
5.26.1.4	<code>O0</code>	168
5.26.1.5	<code>O1</code>	168
5.26.1.6	<code>O2</code>	168
5.26.1.7	<code>O3</code>	168
5.26.1.8	<code>O4</code>	168
5.26.1.9	<code>O5</code>	168
5.26.1.10	<code>O6</code>	168
5.26.1.11	<code>O7</code>	168
5.26.1.12	<code>O8</code>	168
5.26.2	Function Documentation	169
5.26.2.1	<code>loadrgb_kb</code>	169
5.26.2.2	<code>makergb_512</code>	170
5.26.2.3	<code>makergb_full</code>	171
5.26.2.4	<code>ordered8to3</code>	172
5.26.2.5	<code>quantize8to3</code>	172
5.26.2.6	<code>rgbcmp</code>	172
5.26.2.7	<code>savergb_kb</code>	173
5.26.2.8	<code>updatergb_kb</code>	174
5.26.3	Variable Documentation	175
5.26.3.1	<code>bit_reverse_table</code>	176
5.27	<code>src/ckb-daemon/led_mouse.c</code> File Reference	177
5.27.1	Function Documentation	177
5.27.1.1	<code>isblack</code>	177

5.27.1.2	loadrgb_mouse	178
5.27.1.3	rgbcmp	178
5.27.1.4	savergb_mouse	179
5.27.1.5	updatergb_mouse	179
5.28	src/ckb-daemon/main.c File Reference	180
5.28.1	Function Documentation	181
5.28.1.1	localecase	181
5.28.1.2	main	182
5.28.1.3	quit	184
5.28.1.4	quitWithLock	185
5.28.1.5	restart	186
5.28.1.6	sighandler	187
5.28.1.7	sighandler2	188
5.28.1.8	timespec_add	188
5.28.2	Variable Documentation	189
5.28.2.1	features_mask	189
5.28.2.2	hwload_mode	189
5.28.2.3	main_ac	189
5.28.2.4	main_av	189
5.28.2.5	reset_stop	189
5.29	src/ckb-daemon/notify.c File Reference	189
5.29.1	Macro Definition Documentation	190
5.29.1.1	HW_STANDARD	190
5.29.1.2	HWMODE_OR_RETURN	190
5.29.2	Function Documentation	191
5.29.2.1	_cmd_get	191
5.29.2.2	cmd_get	193
5.29.2.3	cmd_notify	194
5.29.2.4	cmd_restart	194
5.29.2.5	nprintf	195
5.29.2.6	nprintind	196
5.29.2.7	nprintkey	196
5.29.2.8	restart	197
5.30	src/ckb-daemon/notify.h File Reference	198
5.30.1	Function Documentation	199
5.30.1.1	cmd_get	199
5.30.1.2	cmd_notify	200
5.30.1.3	cmd_restart	200
5.30.1.4	nprintf	200
5.30.1.5	nprintind	201

5.30.1.6	nprintkey	202
5.31	src/ckb-daemon/os.h File Reference	202
5.31.1	Macro Definition Documentation	203
5.31.1.1	_DEFAULT_SOURCE	203
5.31.1.2	_GNU_SOURCE	203
5.31.1.3	eid_guard_start	203
5.31.1.4	eid_guard_stop	203
5.31.1.5	UINPUT_VERSION	204
5.32	src/ckb-daemon/profile.c File Reference	204
5.32.1	Function Documentation	205
5.32.1.1	_freeprofile	205
5.32.1.2	allocprofile	205
5.32.1.3	cmd_erase	206
5.32.1.4	cmd_eraseprofile	207
5.32.1.5	cmd_id	207
5.32.1.6	cmd_name	208
5.32.1.7	cmd_profileid	208
5.32.1.8	cmd_profilename	209
5.32.1.9	freemode	209
5.32.1.10	freeprofile	210
5.32.1.11	gethwmodename	210
5.32.1.12	gethwprofilename	211
5.32.1.13	getid	212
5.32.1.14	getmodename	212
5.32.1.15	getprofilename	213
5.32.1.16	hwtonative	214
5.32.1.17	initmode	214
5.32.1.18	loadprofile	215
5.32.1.19	nativetohw	215
5.32.1.20	printname	216
5.32.1.21	setid	217
5.32.1.22	u16dec	217
5.32.1.23	u16enc	218
5.32.1.24	urldecode2	219
5.32.1.25	urlencode2	219
5.32.2	Variable Documentation	220
5.32.2.1	utf16to8	220
5.32.2.2	utf8to16	220
5.33	src/ckb-daemon/profile.h File Reference	220
5.33.1	Macro Definition Documentation	221

5.33.1.1	hwloadprofile	221
5.33.2	Function Documentation	221
5.33.2.1	allocprofile	221
5.33.2.2	cmd_erase	222
5.33.2.3	cmd_eraseprofile	223
5.33.2.4	cmd_hwload_kb	223
5.33.2.5	cmd_hwload_mouse	224
5.33.2.6	cmd_hwsave_kb	225
5.33.2.7	cmd_hwsave_mouse	226
5.33.2.8	cmd_id	227
5.33.2.9	cmd_name	227
5.33.2.10	cmd_profileid	228
5.33.2.11	cmd_profilename	228
5.33.2.12	freeprofile	229
5.33.2.13	gethwmodename	229
5.33.2.14	gethwprofilename	230
5.33.2.15	getid	231
5.33.2.16	getmodename	231
5.33.2.17	getprofilename	232
5.33.2.18	hwtonative	233
5.33.2.19	loadprofile	233
5.33.2.20	nativetohw	234
5.33.2.21	setid	234
5.34	src/ckb-daemon/profile_keyboard.c File Reference	235
5.34.1	Function Documentation	235
5.34.1.1	cmd_hwload_kb	235
5.34.1.2	cmd_hwsave_kb	236
5.34.1.3	hwloadmode	237
5.35	src/ckb-daemon/profile_mouse.c File Reference	238
5.35.1	Function Documentation	238
5.35.1.1	cmd_hwload_mouse	239
5.35.1.2	cmd_hwsave_mouse	240
5.36	src/ckb-daemon/structures.h File Reference	240
5.36.1	Data Structure Documentation	242
5.36.1.1	struct usbid	242
5.36.1.2	struct macroaction	243
5.36.1.3	struct keymacro	244
5.36.1.4	struct binding	245
5.36.1.5	struct dpiset	246
5.36.1.6	struct lighting	247

5.36.1.7	struct usbmode	248
5.36.1.8	struct usbprofile	249
5.36.1.9	struct hwprofile	250
5.36.1.10	struct usbinput	251
5.36.1.11	struct usbdevice	252
5.36.2	Macro Definition Documentation	254
5.36.2.1	CLEAR_KEYBIT	254
5.36.2.2	DPI_COUNT	254
5.36.2.3	FEAT_ADJRATE	254
5.36.2.4	FEAT_ANSI	254
5.36.2.5	FEAT_BIND	254
5.36.2.6	FEAT_COMMON	254
5.36.2.7	FEAT_FWUPDATE	254
5.36.2.8	FEAT_FWVERSION	255
5.36.2.9	FEAT_HWLOAD	255
5.36.2.10	FEAT_ISO	255
5.36.2.11	FEAT_LMASK	255
5.36.2.12	FEAT_MONOCHROME	255
5.36.2.13	FEAT_MOUSEACCEL	255
5.36.2.14	FEAT_NOTIFY	255
5.36.2.15	FEAT_POLLRATE	255
5.36.2.16	FEAT_RGB	255
5.36.2.17	FEAT_STD_NRGB	256
5.36.2.18	FEAT_STD_RGB	256
5.36.2.19	HAS_ANY_FEATURE	256
5.36.2.20	HAS_FEATURES	256
5.36.2.21	HWMODE_K70	256
5.36.2.22	HWMODE_K95	256
5.36.2.23	HWMODE_MAX	256
5.36.2.24	I_CAPS	256
5.36.2.25	I_NUM	256
5.36.2.26	I_SCROLL	256
5.36.2.27	IFACE_MAX	257
5.36.2.28	KB_NAME_LEN	257
5.36.2.29	LIFT_MAX	257
5.36.2.30	LIFT_MIN	257
5.36.2.31	MACRO_MAX	257
5.36.2.32	MD_NAME_LEN	257
5.36.2.33	MODE_COUNT	257
5.36.2.34	MSG_SIZE	257

5.36.2.35	NEEDS_FW_UPDATE	257
5.36.2.36	OUTFIFO_MAX	258
5.36.2.37	PR_NAME_LEN	258
5.36.2.38	SCROLL_ACCELERATED	258
5.36.2.39	SCROLL_MAX	258
5.36.2.40	SCROLL_MIN	258
5.36.2.41	SERIAL_LEN	258
5.36.2.42	SET_KEYBIT	258
5.36.3	Variable Documentation	258
5.36.3.1	vtable_keyboard	258
5.36.3.2	vtable_keyboard_nonrgb	258
5.36.3.3	vtable_mouse	259
5.37	src/ckb-daemon/usb.c File Reference	259
5.37.1	Function Documentation	260
5.37.1.1	_resetusb	260
5.37.1.2	_setupusb	260
5.37.1.3	_usbrecv	265
5.37.1.4	_usbsend	266
5.37.1.5	closeusb	267
5.37.1.6	devmain	269
5.37.1.7	get_vtable	271
5.37.1.8	product_str	272
5.37.1.9	revertusb	273
5.37.1.10	setupusb	274
5.37.1.11	usb_tryreset	275
5.37.1.12	vendor_str	276
5.37.2	Variable Documentation	276
5.37.2.1	features_mask	276
5.37.2.2	hwload_mode	276
5.37.2.3	reset_stop	277
5.37.2.4	usbmutex	277
5.38	src/ckb-daemon/usb.h File Reference	277
5.38.1	Detailed Description	281
5.38.2	Macro Definition Documentation	281
5.38.2.1	DELAY_LONG	281
5.38.2.2	DELAY_MEDIUM	281
5.38.2.3	DELAY_SHORT	281
5.38.2.4	IS_FULLRANGE	281
5.38.2.5	IS_K65	282
5.38.2.6	IS_K70	282

5.38.2.7 IS_K95	282
5.38.2.8 IS_M65	282
5.38.2.9 IS_MONOCHROME	282
5.38.2.10 IS_MONOCHROME_DEV	282
5.38.2.11 IS_MOUSE	282
5.38.2.12 IS_MOUSE_DEV	282
5.38.2.13 IS_RGB	282
5.38.2.14 IS_RGB_DEV	283
5.38.2.15 IS_SABRE	283
5.38.2.16 IS_SCIMITAR	283
5.38.2.17 IS_STRAFE	283
5.38.2.18 NK95_HWOFF	283
5.38.2.19 NK95_HWON	283
5.38.2.20 NK95_M1	283
5.38.2.21 NK95_M2	283
5.38.2.22 NK95_M3	284
5.38.2.23 nk95cmd	284
5.38.2.24 P_HARPOON	284
5.38.2.25 P_HARPOON_STR	284
5.38.2.26 P_K65	284
5.38.2.27 P_K65_LUX	284
5.38.2.28 P_K65_LUX_STR	284
5.38.2.29 P_K65_NRGB	284
5.38.2.30 P_K65_NRGB_STR	284
5.38.2.31 P_K65_RFIRE	284
5.38.2.32 P_K65_RFIRE_STR	285
5.38.2.33 P_K65_STR	285
5.38.2.34 P_K70	285
5.38.2.35 P_K70_LUX	285
5.38.2.36 P_K70_LUX_NRGB	285
5.38.2.37 P_K70_LUX_NRGB_STR	285
5.38.2.38 P_K70_LUX_STR	285
5.38.2.39 P_K70_NRGB	285
5.38.2.40 P_K70_NRGB_STR	285
5.38.2.41 P_K70_RFIRE	285
5.38.2.42 P_K70_RFIRE_NRGB	286
5.38.2.43 P_K70_RFIRE_NRGB_STR	286
5.38.2.44 P_K70_RFIRE_STR	286
5.38.2.45 P_K70_STR	286
5.38.2.46 P_K95	286

5.38.2.47 P_K95_NRGB	286
5.38.2.48 P_K95_NRGB_STR	286
5.38.2.49 P_K95_PLATINUM	286
5.38.2.50 P_K95_PLATINUM_STR	286
5.38.2.51 P_K95_STR	286
5.38.2.52 P_M65	286
5.38.2.53 P_M65_PRO	287
5.38.2.54 P_M65_PRO_STR	287
5.38.2.55 P_M65_STR	287
5.38.2.56 P_SABRE_L	287
5.38.2.57 P_SABRE_L_STR	287
5.38.2.58 P_SABRE_N	287
5.38.2.59 P_SABRE_N_STR	287
5.38.2.60 P_SABRE_O	287
5.38.2.61 P_SABRE_O2	287
5.38.2.62 P_SABRE_O2_STR	287
5.38.2.63 P_SABRE_O_STR	287
5.38.2.64 P_SCIMITAR	288
5.38.2.65 P_SCIMITAR_PRO	288
5.38.2.66 P_SCIMITAR_PRO_STR	288
5.38.2.67 P_SCIMITAR_STR	288
5.38.2.68 P_STRAFE	288
5.38.2.69 P_STRAFE_NRGB	288
5.38.2.70 P_STRAFE_NRGB_STR	288
5.38.2.71 P_STRAFE_STR	288
5.38.2.72 resetusb	288
5.38.2.73 USB_DELAY_DEFAULT	288
5.38.2.74 usbrecv	288
5.38.2.75 usbsend	289
5.38.2.76 V_CORSAIR	289
5.38.2.77 V_CORSAIR_STR	289
5.38.3 Function Documentation	289
5.38.3.1 _nk95cmd	289
5.38.3.2 _resetusb	291
5.38.3.3 _usbrecv	291
5.38.3.4 _usbsend	293
5.38.3.5 closeusb	295
5.38.3.6 os_closeusb	296
5.38.3.7 os_inputmain	297
5.38.3.8 os_resetusb	301

5.38.3.9	os_sendindicators	302
5.38.3.10	os_setupusb	303
5.38.3.11	os_usbrecv	304
5.38.3.12	os_usbsend	306
5.38.3.13	product_str	308
5.38.3.14	revertusb	310
5.38.3.15	setupusb	311
5.38.3.16	usb_tryreset	313
5.38.3.17	usbkill	314
5.38.3.18	usbmain	314
5.38.3.19	vendor_str	316
5.39	src/ckb-daemon/usb_linux.c File Reference	316
5.39.1	Data Structure Documentation	318
5.39.1.1	struct _model	318
5.39.2	Macro Definition Documentation	318
5.39.2.1	N_MODELS	318
5.39.2.2	TEST_RESET	318
5.39.3	Function Documentation	319
5.39.3.1	_nk95cmd	319
5.39.3.2	os_closeusb	320
5.39.3.3	os_inputmain	321
5.39.3.4	os_resetusb	324
5.39.3.5	os_sendindicators	325
5.39.3.6	os_setupusb	326
5.39.3.7	os_usbrecv	327
5.39.3.8	os_usbsend	328
5.39.3.9	strtrim	330
5.39.3.10	udev_enum	331
5.39.3.11	usb_add_device	332
5.39.3.12	usb_rm_device	333
5.39.3.13	usbadd	334
5.39.3.14	usbclaim	336
5.39.3.15	usbkill	336
5.39.3.16	usbmain	337
5.39.3.17	usbunclaim	338
5.39.4	Variable Documentation	339
5.39.4.1	kbsyspath	339
5.39.4.2	models	339
5.39.4.3	udev	340
5.39.4.4	udevthread	340

CONTENTS	xxi
5.39.4.5 usbthread	340
5.40 src/ckb-daemon/usb_mac.c File Reference	340
Index	342

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:

src/ckb-daemon/command.c	11
src/ckb-daemon/command.h	16
src/ckb-daemon/device.c	25
src/ckb-daemon/device.h	28
src/ckb-daemon/device_keyboard.c	38
src/ckb-daemon/device_mouse.c	41
src/ckb-daemon/device_vtable.c	44
src/ckb-daemon/devnode.c	47
src/ckb-daemon/devnode.h	60
src/ckb-daemon/dpi.c	68
src/ckb-daemon/dpi.h	73
src/ckb-daemon/extra_mac.c	78
src/ckb-daemon/firmware.c	78
src/ckb-daemon/firmware.h	83
src/ckb-daemon/includes.h	85
src/ckb-daemon/input.c	89
src/ckb-daemon/input.h	98
src/ckb-daemon/input_linux.c	107
src/ckb-daemon/input_mac.c	114
src/ckb-daemon/input_mac_mouse.c	114
src/ckb-daemon/keymap.c	114
src/ckb-daemon/keymap.h	119
src/ckb-daemon/keymap_mac.h	127
src/ckb-daemon/led.c	128
src/ckb-daemon/led.h	134
src/ckb-daemon/led_keyboard.c	145
src/ckb-daemon/led_mouse.c	177
src/ckb-daemon/main.c	180
src/ckb-daemon/notify.c	189
src/ckb-daemon/notify.h	198
src/ckb-daemon/os.h	202
src/ckb-daemon/profile.c	204
src/ckb-daemon/profile.h	220
src/ckb-daemon/profile_keyboard.c	235
src/ckb-daemon/profile_mouse.c	238
src/ckb-daemon/structures.h	240
src/ckb-daemon/usb.c	259

src/ckb-daemon/ usb.h	
Definitions for using USB interface	277
src/ckb-daemon/ usb_linux.c	316
src/ckb-daemon/ usb_mac.c	340

Chapter 4

Data Structure Documentation

4.1 devcmd.__unnamed__ Struct Reference

Collaboration diagram for devcmd.__unnamed__:

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

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

4.1.2.2

4.1.2.3

4.1.2.4

4.1.2.5

4.1.2.6

4.1.2.7

4.1.2.8

4.1.2.9

4.1.2.10

4.1.2.11

4.1.2.12

4.1.2.13

4.1.2.14

4.1.2.15

4.1.2.16

4.1.2.17

4.1.2.18

4.1.2.19

4.1.2.20

4.1.2.21

4.1.2.22

4.1.2.23

4.1.2.24

4.1.2.25

4.1.2.26

4.1.2.27

4.1.2.28

4.1.2.29

4.1.2.30

4.1.2.31

4.1.2.32

4.1.2.33

4.1.2.34

4.1.2.35

4.1.2.36

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:
```



- ## Functions

- ## Variables

- ### 5.1.1 Macro Definition Documentation

5.1.1.1 #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 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 #endif
177 case MODE: {
178     // Select a mode number (1 - 6)
179     int newmode;
180     if(sscanf(word, "%u", &newmode) == 1 && newmode > 0 && newmode <=
181     MODE_COUNT)
182         mode = profile->mode + newmode - 1;
183     continue;
184 }
185 case FPS: {
186     // USB command delay (2 - 10ms)
187     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, notifynumber, 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, notifynumber, 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, notifynumber, 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

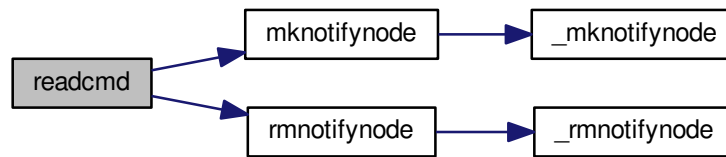
```

```

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             }
281             continue;
282         }
283         break;
284     }
285     case MACRO:
286         if(!strcmp(word, "clear")){
287             // Macro has a special clear command
288             vt->macro(kb, mode, notifynumber, 0, 0);
289             continue;
290         }
291         break;
292     default:;
293     }
294     // For anything else, split the parameter at the colon
295     int left = -1;
296     sscanf(word, "%*[^:]\n", &left);
297     if(left <= 0)
298         continue;
299     const char* right = word + left;
300     if(right[0] == ':')
301         right++;
302     // Macros and DPI have a separate left-side handler
303     if(command == MACRO || command == DPI){
304         word[left] = 0;
305         vt->do_macro[command](kb, mode, notifynumber, word, right);
306         continue;
307     }
308     // Scan the left side for key names and run the requested command
309     int position = 0, field = 0;
310     char keyname[11];
311     while(position < left && sscanf(word + position, "%10[^:]\n", keyname, &field) == 1){
312         int keycode;
313         if(!strcmp(keyname, "all")){
314             // Set all keys
315             for(int i = 0; i < N_KEYS_EXTENDED; i++){
316                 vt->do_cmd[command](kb, mode, notifynumber, i, right);
317             }
318         } else if((sscanf(keyname, "%d", &keycode) && keycode >= 0 && keycode <
N_KEYS_EXTENDED) || (sscanf(keyname, "%x", &keycode) && keycode >= 0 && keycode <
N_KEYS_EXTENDED)){
319             // Set a key numerically
320             vt->do_cmd[command](kb, mode, notifynumber, keycode, right);
321         } else {
322             // Find this key in the keymap
323             for(unsigned i = 0; i < N_KEYS_EXTENDED; i++){
324                 if(keymap[i].name && !strcmp(keyname, keymap[i].name)){
325                     vt->do_cmd[command](kb, mode, notifynumber, i, right);
326                     break;
327                 }
328             }
329         }
330         if(word[position += field] == ',')
331             position++;
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 `const char* const cmd_strings[(CMD_LAST-CMD_FIRST+2)-1]` [static]

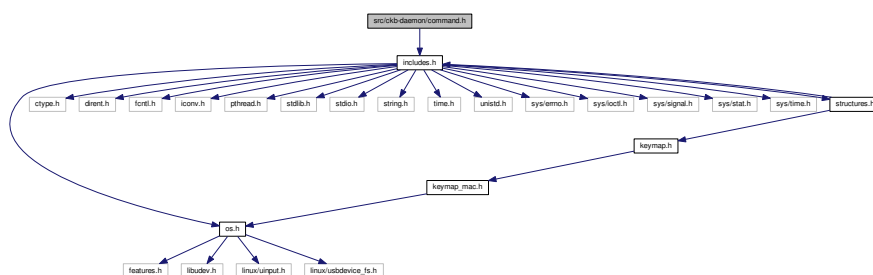
Definition at line 9 of file `command.c`.

Referenced by `readcmd()`.

5.2 `src/ckb-daemon/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

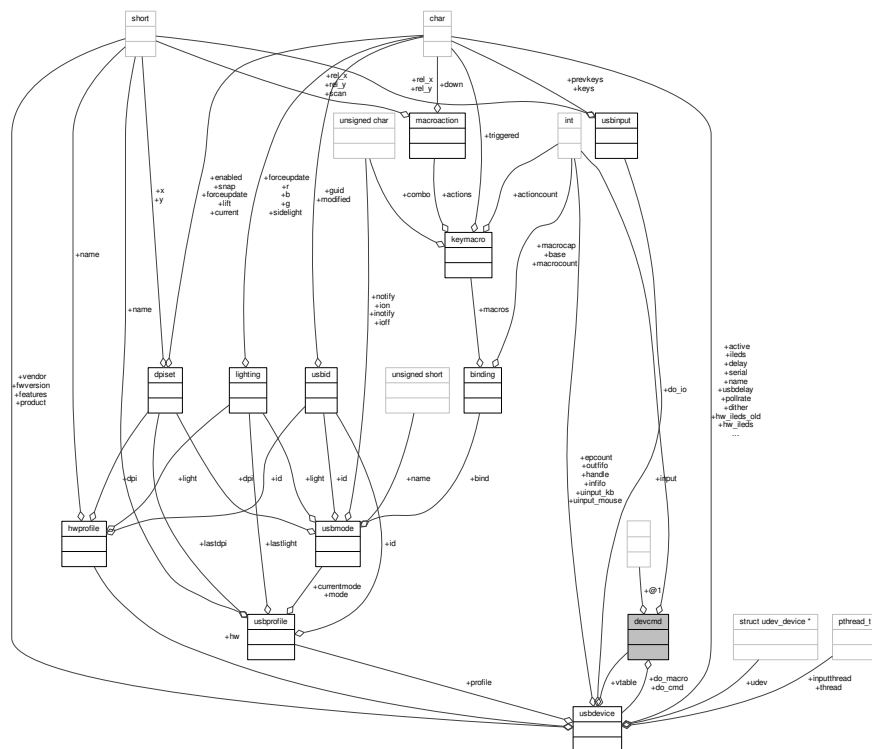
- `int readcmd(usbdevice *kb, const char *line)`

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_ LA - ST-CMD_VT_FI - RST+1)]	
cmdhandler_ - mac	do_macro[(CM - D_LAST-CMD_ - VT_FIRST+1)]	

5.2.2 Macro Definition Documentation

5.2.2.1 #define CMD_COUNT (CMD_LAST - CMD_FIRST + 2)

Definition at line 65 of file command.h.

Referenced by readcmd().

5.2.2.2 #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 `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 `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 `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 `typedef union devcmd devcmd`

5.2.4 Enumeration Type Documentation

5.2.4.1 `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

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 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, DPISSEL, 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
    have the feature enabled.
116     if(command == NONE
117         || (!HAS_FEATURES(kb, FEAT_BIND) && (command ==
BIND || command == UNBIND || command == REBIND || command ==
MACRO || command == DELAY))
118         || (!HAS_FEATURES(kb, FEAT_NOTIFY) && command ==
NOTIFY)){
119         next_loop:
120         continue;
121     }
122     // Reject anything not related to fwupdate if device has a bricked FW
123     if(NEEDS_FW_UPDATE(kb) && command != FWUPDATE && command !=
NOTIFYON && command != NOTIFYOFF)
124         continue;
125
126     // Specially handled commands - these are available even when keyboard is IDLE
127     switch(command){
128     case NOTIFYON: {
129         // Notification node on
130         int notify;
131         if(sscanf(word, "%u", &notify) == 1)
132             mknotifynode(kb, notify);
133         continue;
134     } case NOTIFYOFF: {
135         // Notification node off
136         int notify;
137         if(sscanf(word, "%u", &notify) == 1 && notify != 0) // notify0 can't be removed
138             rmnotifynode(kb, notify);
139         continue;
140     } case GET:
141         // Output data to notification node
142         vt->get(kb, mode, notifynumber, 0, word);
143         continue;
144     case LAYOUT:
145         // OSX: switch ANSI/ISO keyboard layout
146         if(!strcmp(word, "ansi"))
147             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")); // independant 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, 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, notifiynumber, 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
DPISL: case LIFT: case SNAP:
270         // All of the above just parse the whole word
271         vt->do_cmd[command](kb, mode, notifiynumber, 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, notifiynumber, 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, notifiynumber, 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, notifiynumber, 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, notifiynumber, i, right);
316     } else if((sscanf(keyname, "%d", &keycode) && keycode >= 0 && keycode <
N_KEYS_EXTENDED)
317 || (sscanf(keyname, "%x%x", &keycode) && keycode >= 0 && keycode <
N_KEYS_EXTENDED)){
318         // Set a key numerically
319         vt->do_cmd[command](kb, mode, notifiynumber, 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, notifiynumber, 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 }

```

```
graph LR; readcmd[readcmd] --> mknotifnode[mknotifnode]; readcmd --> rmnotifnode[rmnotifnode]; mknotifnode --> _mknotifnode[_mknotifnode]; rmnotifnode --> _rmnotifnode[_rmnotifnode];
```



- ## Variables

- Generated on Thu Apr 27 2017 21:48:46 for ckb-next by Doxygen

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 int _start_dev (usbdevice * kb, int makeactive)

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

Parameters

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

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) {
25         if (!hwload_mode || (HAS_FEATURES(kb, FEAT_HWLOAD) &&
26             getfwversion(kb))) {
27             if (hwload_mode == 2)
28                 // hwload=always. Report setup failure.
29                 return -1;
30             else if (hwload_mode) {
31                 // hwload=once. Log failure, prevent trying again, and continue.
32                 ckb_warn("Unable to load firmware version/poll rate\n");
33                 kb->features &= ~FEAT_HWLOAD;
34             }
35             kb->pollrate = 0;
36             kb->features &= ~(FEAT_POLLRATE | FEAT_ADJRATE);
37             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 int start_dev (usbdevice * kb, int makeactive)

Definition at line 77 of file device.c.

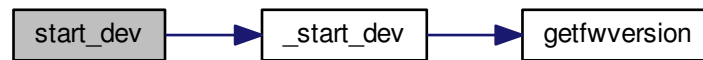
References `_start_dev()`, `USB_DELAY_DEFAULT`, and `usbdevice::usbdelay`.

```

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 `pthread_mutex_t devlistmutex = PTHREAD_MUTEX_INITIALIZER`

Definition at line 11 of file `device.c`.

5.3.2.2 `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 `int hwload_mode = 1`

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

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

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

Definition at line 13 of file `device.c`.

5.3.2.5 `usbdevice keyboard[9]`

Definition at line 10 of file `device.c`.

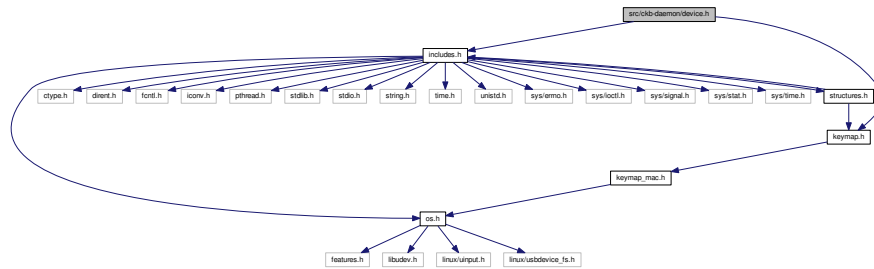
Referenced by `_mkdevpath()`, `_mknotifynode()`, `_rmnotifynode()`, `_setupusb()`, `_updateconnected()`, `closeusb()`, `main()`, `mkfwnode()`, `os_closeusb()`, `os_inputmain()`, `os_inputopen()`, `os_setupusb()`, `quitWithLock()`, `rmdevpath()`, `usb_rm_device()`, and `usbadd()`.

5.4 `src/ckb-daemon/device.h` File Reference

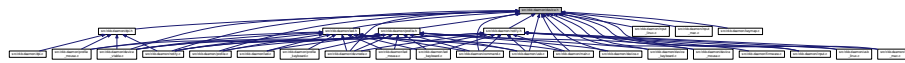
```

#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 `#define ACT_LIGHT 1`

Definition at line 60 of file device.h.

Referenced by `setactive_kb()`.

5.4.1.2 `#define ACT_LOCK 8`

Definition at line 63 of file device.h.

Referenced by `setactive_kb()`.

5.4.1.3 `#define ACT_M1 10`

Definition at line 65 of file device.h.

Referenced by `setactive_kb()`.

5.4.1.4 `#define ACT_M2 11`

Definition at line 66 of file device.h.

Referenced by `setactive_kb()`.

5.4.1.5 `#define ACT_M3 12`

Definition at line 67 of file device.h.

Referenced by `setactive_kb()`.

5.4.1.6 `#define ACT_MR_RING 9`

Definition at line 64 of file device.h.

Referenced by `setactive_kb()`.

5.4.1.7 `#define ACT_NEXT 3`

Definition at line 61 of file device.h.

5.4.1.8 `#define ACT_NEXT_NOWRAP 5`

Definition at line 62 of file device.h.

5.4.1.9 #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 #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 #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 #define IN_CORSAIR 0x40

Definition at line 57 of file device.h.

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

5.4.1.13 #define IN_HID 0x80

Definition at line 56 of file device.h.

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

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

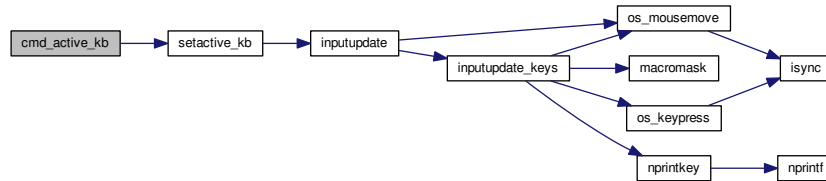
Definition at line 112 of file device_keyboard.c.

References `setactive_kb()`.

```

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

Here is the call graph for this function:



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

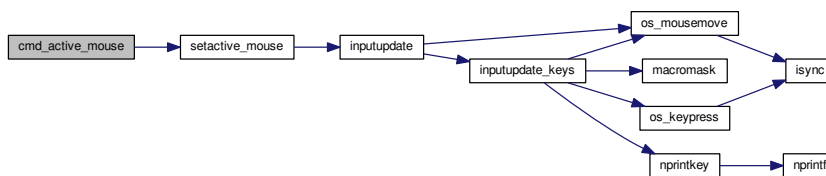
Definition at line 44 of file device_mouse.c.

References setactive_mouse().

```

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

Here is the call graph for this function:



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

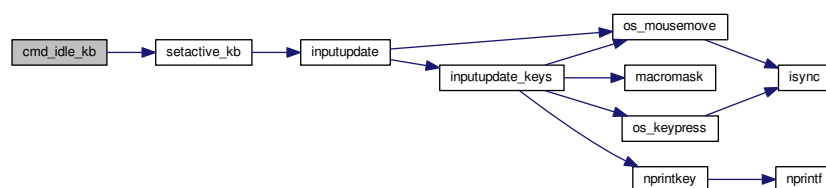
Definition at line 116 of file device_keyboard.c.

References setactive_kb().

```

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

Here is the call graph for this function:



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

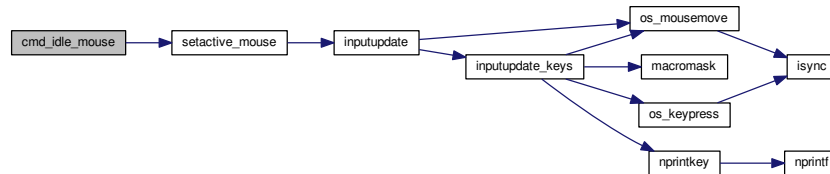
Definition at line 48 of file device_mouse.c.

References `setactive_mouse()`.

```

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

Here is the call graph for this function:



5.4.2.5 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 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`, `usbdevice::keys`, `usbprofile::lastlight`, `MSG_SIZE`, `N_KEYS_HW`, `NEEDS_FW_UPDATE`, `usbdevice::profile`, `usbdevice::send`, and `usbdevice::vtable`.

Referenced by `cmd_active_kb()`, and `cmd_idle_kb()`.

```

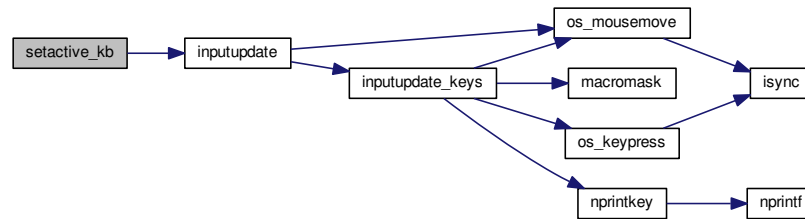
18                                     {
19     if (NEEDS_FW_UPDATE(kb))
20         return 0;
21
22     pthread_mutex_lock(&imutex(kb));
23     kb->active = !active;
24     kb->profile->lastlight.forceupdate = 1;
25     // Clear input
26     memset(&kb->input.keys, 0, sizeof(kb->input.keys));
27     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     };
34 }
```

```

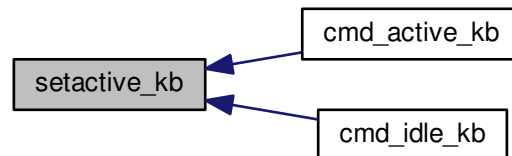
33     { 0x07, 0x05, 2, 0, 0x03, 0x00 } // Commits key input selection
34 };
35 if(active){
36     // Put the M-keys (K95) as well as the Brightness/Lock keys into software-controlled mode.
37     msg[0][2] = 2;
38     if(!usbSend(kb, msg[0], 1))
39         return -1;
40     DELAY_MEDIUM(kb);
41     // 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
    representing the mode.
43     for(int key = 0; key < N_KEYS_HW; ){
44         int pair;
45         for(pair = 0; pair < 30 && key < N_KEYS_HW; pair++, key++){
46             // Select both standard and Corsair input. The standard input will be ignored except in
    BIOS mode.
47             uchar action = IN_HID | IN_CORSAIR;
48             // Additionally, make MR activate the MR ring (this is disabled for now, may be back later)
49             //if(keymap[key].name && !strcmp(keymap[key].name, "mr"))
50             //    action |= ACT_MR_RING;
51             msg[1][4 + pair * 2] = key;
52             msg[1][5 + pair * 2] = action;
53         }
54         // Byte 2 = pair count (usually 30, less on final message)
55         msg[1][2] = pair;
56         if(!usbSend(kb, msg[1], 1))
57             return -1;
58     }
59     // Commit new input settings
60     if(!usbSend(kb, msg[2], 1))
61         return -1;
62     DELAY_MEDIUM(kb);
63 } else {
64     // Set the M-keys back into hardware mode, restore hardware RGB profile. It has to be sent twice
    for some reason.
65     msg[0][2] = 1;
66     if(!usbSend(kb, msg[0], 1))
67         return -1;
68     DELAY_MEDIUM(kb);
69     if(!usbSend(kb, msg[0], 1))
70         return -1;
71     DELAY_MEDIUM(kb);
72 #ifdef OS_LINUX
73     // On OSX the default key mappings are fine. On Linux, the G keys will freeze the keyboard. Set the
    keyboard entirely to HID input.
74     for(int key = 0; key < N_KEYS_HW; ){
75         int pair;
76         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 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()`.

```

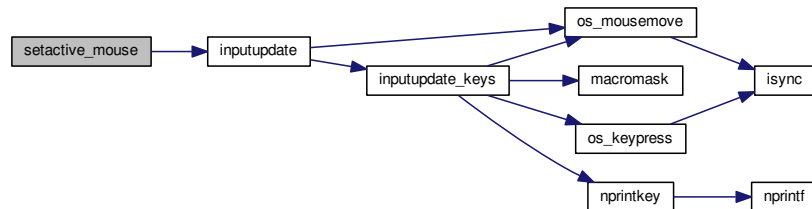
9                                     {
10    if (NEEDS_FW_UPDATE(kb))
11        return 0;
12    const int keycount = 20;
13    uchar msg[2][MSG_SIZE] = {
14        { 0x07, 0x04, 0 },
15        { 0x07, 0x40, keycount, 0 },
16    };
17    if (active)
18        // Put the mouse into SW mode
19        msg[0][2] = 2;
20    else
21        // Restore HW mode
22        msg[0][2] = 1;
23    pthread_mutex_lock(&imutex(kb));
24    kb->active = !active;
25    kb->profile->lastlight.forceupdate = 1;
26    // Clear input
27    memset(&kb->input.keys, 0, sizeof(kb->input.keys));
28    inputupdate(kb);
29    pthread_mutex_unlock(&imutex(kb));
30    if (!usbsend(kb, msg[0], 1))
31        return -1;
32    if (active) {
33        // Set up key input
34        if (!usbsend(kb, msg[1], 1))
  
```

```

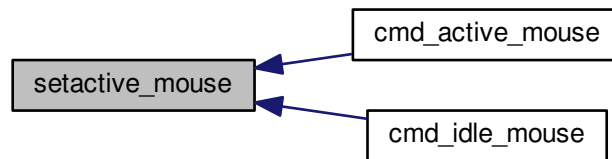
35         return -1;
36     for(int i = 0; i < keycount; i++){
37         msg[1][i * 2 + 4] = i + 1;
38         msg[1][i * 2 + 5] = (i < 6 ? IN_HID : IN_CORSAIR);
39     }
40 }
41 return 0;
42 }

```

Here is the call graph for this function:



Here is the caller graph for this function:



5.4.2.8 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 int start_dev (usbdevice * kb, int makeactive)

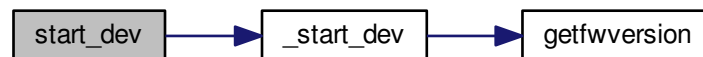
Definition at line 77 of file device.c.

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 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 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 pthread_mutex_t inputmutex[9]

Definition at line 13 of file `device.c`.

5.4.3.3 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()`.

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

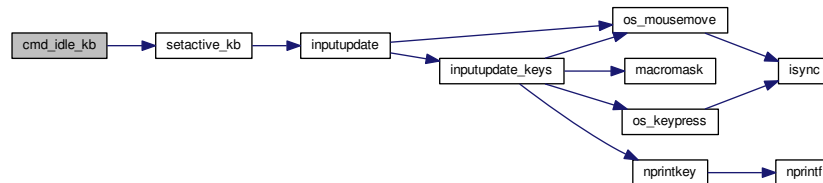
Definition at line 116 of file device_keyboard.c.

References setactive_kb().

```

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

Here is the call graph for this function:



5.5.1.3 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, usbdevice::send, and usbdevice::vtable.

Referenced by cmd_active_kb(), and cmd_idle_kb().

```

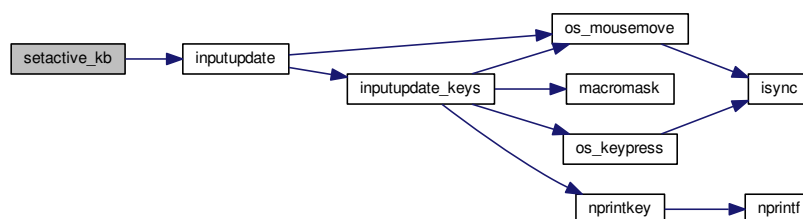
18                                     {
19     if (NEEDS_FW_UPDATE(kb))
20         return 0;
21
22     pthread_mutex_lock(imutex(kb));
23     kb->active = !active;
24     kb->profile->lastlight.forceupdate = 1;
25     // Clear input
26     memset(&kb->input.keys, 0, sizeof(kb->input.keys));
27     inputupdate(kb);
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 (!usbdevice::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         }
57     }
58 }
```

```

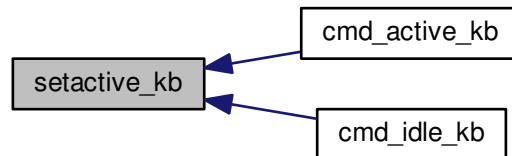
53         }
54         // Byte 2 = pair count (usually 30, less on final message)
55         msg[1][2] = pair;
56         if(!usbSend(kb, msg[1], 1))
57             return -1;
58     }
59     // Commit new input settings
60     if(!usbSend(kb, msg[2], 1))
61         return -1;
62     DELAY_MEDIUM(kb);
63 } else {
64     // Set the M-keys back into hardware mode, restore hardware RGB profile. It has to be sent twice
65     for some reason.
66     msg[0][2] = 1;
67     if(!usbSend(kb, msg[0], 1))
68         return -1;
69     DELAY_MEDIUM(kb);
70     if(!usbSend(kb, msg[0], 1))
71         return -1;
72     DELAY_MEDIUM(kb);
73 #ifdef OS_LINUX
74     // On OSX the default key mappings are fine. On Linux, the G keys will freeze the keyboard. Set the
75     keyboard entirely to HID input.
76     for(int key = 0; key < N_KEYS_HW; ){
77         int pair;
78         for(pair = 0; pair < 30 && key < N_KEYS_HW; pair++, key++){
79             uchar action = IN_HID;
80             // Enable hardware actions
81             if(keymap[key].name){
82                 if(!strcmp(keymap[key].name, "mr"))
83                     action = ACT_MR_RING;
84                 else if(!strcmp(keymap[key].name, "m1"))
85                     action = ACT_M1;
86                 else if(!strcmp(keymap[key].name, "m2"))
87                     action = ACT_M2;
88                 else if(!strcmp(keymap[key].name, "m3"))
89                     action = ACT_M3;
90                 else if(!strcmp(keymap[key].name, "light"))
91                     action = ACT_LIGHT;
92                 else if(!strcmp(keymap[key].name, "lock"))
93                     action = ACT_LOCK;
94             }
95             msg[1][4 + pair * 2] = key;
96             msg[1][5 + pair * 2] = action;
97         }
98         // Byte 2 = pair count (usually 30, less on final message)
99         msg[1][2] = pair;
100         if(!usbSend(kb, msg[1], 1))
101             return -1;
102     }
103     // Commit new input settings
104     if(!usbSend(kb, msg[2], 1))
105         return -1;
106     DELAY_MEDIUM(kb);
107 #endif
108 }
109 // Update indicator LEDs if the profile contains settings for them
110 kb->vtable->updateIndicators(kb, 0);
111 return 0;
112 }

```

Here is the call graph for this function:



Here is the caller graph for this function:



5.5.1.4 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 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.

```

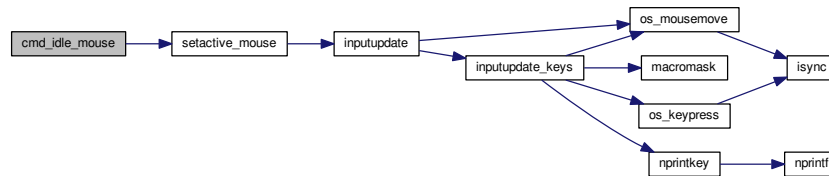
9                                     {
10     // Put the non-RGB K95 into software mode. Nothing else needs to be done hardware wise
11     nk95cmd(kb, NK95_HWOFF);
12     // Fill out RGB features for consistency, even though the keyboard doesn't have them
13     kb->active = 1;
14     kb->pollrate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

5.6 src/ckb-daemon/device_mouse.c File Reference

```

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


Here is the call graph for this function:



5.6.1.3 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 usbsend.

```

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

```

5.6.1.4 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     if(active)
18         // Put the mouse into SW mode
19         msg[0][2] = 2;
20     else
21         // Restore HW mode
22         msg[0][2] = 1;
23     pthread_mutex_lock(&imutex(kb));
24     kb->active = !active;
25     kb->profile->lastlight.forceupdate = 1;
26     // Clear input
27     memset(&kb->input.keys, 0, sizeof(kb->input.keys));
28     inputupdate(kb);
29     pthread_mutex_unlock(&imutex(kb));
30     if(!usbsend(kb, msg[0], 1))
31         return -1;
32     if(active){
33         // Set up key input
34         if(!usbsend(kb, msg[1], 1))
35             return -1;
36         for(int i = 0; i < keycount; i++){
37             msg[1][i * 2 + 4] = i + 1;
38             msg[1][i * 2 + 5] = (i < 6 ? IN_HID : IN_CORSAIR);

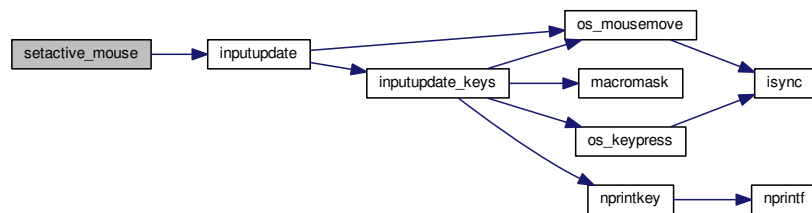
```

```

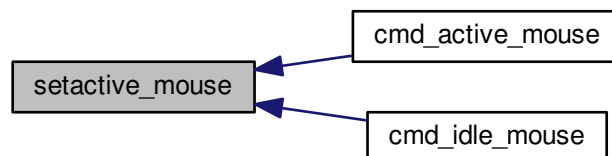
39     }
40 }
41 return 0;
42 }

```

Here is the call graph for this function:



Here is the caller graph for this function:



5.7 src/ckb-daemon/device_vtable.c File Reference

```

#include "command.h"
#include "device.h"
#include "dpi.h"
#include "firmware.h"
#include "input.h"
#include "led.h"
#include "notify.h"
#include "profile.h"

```

- static void `cmd_none` (`usbdevice` *kb, `usbmode` *dummy1, int dummy2, int dummy3, const char *dummy4)
- static int `cmd_io_none` (`usbdevice` *kb, `usbmode` *dummy1, int dummy2, int dummy3, const char *dummy4)
- static void `cmd_macro_none` (`usbdevice` *kb, `usbmode` *dummy1, int dummy2, const char *dummy3, const char *dummy4)
- static int `loadprofile_none` (`usbdevice` *kb)
- static void `int1_void_none` (`usbdevice` *kb, int dummy)
- static int `int1_int_none` (`usbdevice` *kb, int dummy)

- const devcmd vtable_keyboard
- const devcmd vtable_keyboard_nonrgb
- const devcmd vtable_mouse

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

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

Definition at line 16 of file device_vtable.c.

```

16
17 }

```

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

Definition at line 18 of file device_vtable.c.

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

5.7.2 Variable Documentation

5.7.2.1 `const devcmd vtable_keyboard`

Definition at line 28 of file device_vtable.c.

Referenced by get_vtable().

5.7.2.2 `const devcmd vtable_keyboard_nonrgb`

Definition at line 75 of file device_vtable.c.

Referenced by get_vtable().

5.7.2.3 `const devcmd vtable_mouse`

Definition at line 122 of file device_vtable.c.

Referenced by get_vtable().

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

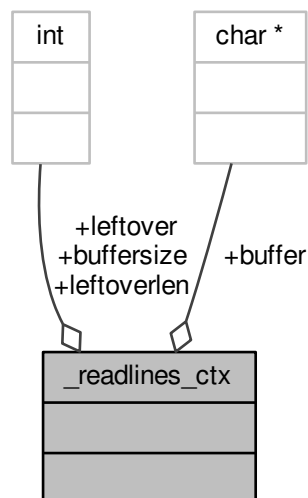
Group ID for the control nodes. -1 to give read/write access to everybody.

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	buffersize	
int	leftover	
int	leftoverlen	

5.8.2 Macro Definition Documentation

5.8.2.1 #define MAX_BUFFER (1024 * 1024 - 1)

Definition at line 334 of file devnode.c.

Referenced by readlines().

5.8.2.2 #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 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

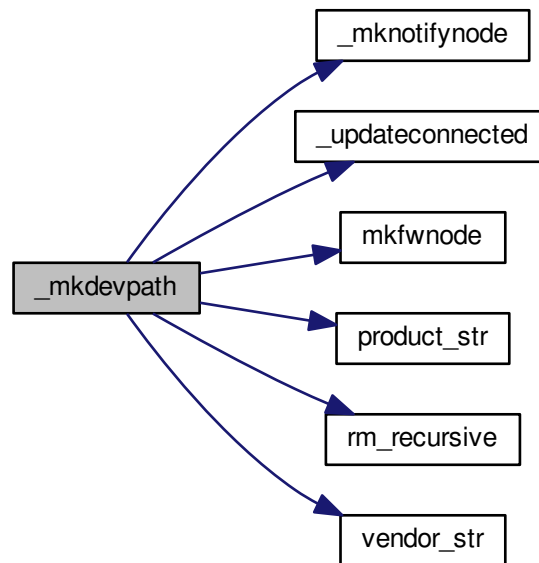
```

```

190         || (kb->inififo = 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->inififo = 0;
195         return -1;
196     }
197     if(gid >= 0)
198         fchown(kb->inififo - 1, 0, gid);
199
200     // Create notification FIFO
201     _mknotifynode(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),
product_str(kb->product));
237         if(HAS_FEATURES(kb, FEAT_MONOCHROME))
238             fputs(" monochrome", ffile);
239         if(HAS_FEATURES(kb, FEAT_RGB))
240             fputs(" rgb", ffile);
241         if(HAS_FEATURES(kb, FEAT_POLLRATE))
242             fputs(" pollrate", ffile);
243         if(HAS_FEATURES(kb, FEAT_ADJRATE))
244             fputs(" adjrate", ffile);
245         if(HAS_FEATURES(kb, FEAT_BIND))
246             fputs(" bind", ffile);
247         if(HAS_FEATURES(kb, FEAT_NOTIFY))
248             fputs(" notify", ffile);
249         if(HAS_FEATURES(kb, FEAT_FWVERSION))
250             fputs(" fwversion", ffile);
251         if(HAS_FEATURES(kb, FEAT_FWUPDATE))
252             fputs(" fwupdate", ffile);
253         fputc('\n', ffile);
254         fclose(ffile);
255         chmod(fpath, S_GID_READ);
256         if(gid >= 0)
257             chown(fpath, 0, gid);
258     } else {
259         ckb_warn("Unable to create %s: %s\n", fpath, strerror(errno));
260         remove(fpath);
261     }
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 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()`.

```

87     {
88         if(notify < 0 || notify >= OUTFIFO_MAX)
89             return -1;
90         if(kb->outfifo[notify] != 0)
91             return 0;
92         // Create the notification node
93         int index = INDEX_OF(kb, keyboard);
94         char outpath[strlen(devpath) + 10];
95         snprintf(outpath, sizeof(outpath), "%s%d/notify%d", devpath, index, notify);
96         if(mkfifo(outpath, S_GID_READ) != 0 || (kb->outfifo[notify] = open(outpath, O_RDWR |
O_NONBLOCK) + 1) == 0){
97             // Add one to the FD because 0 is a valid descriptor, but ckb uses 0 for uninitialized devices
98             ckb_warn("Unable to create %s: %s\n", outpath, strerror(errno));
99             kb->outfifo[notify] = 0;
100             remove(outpath);
101             return -1;
102         }
103         if(gid >= 0)
104             fchown(kb->outfifo[notify] - 1, 0, gid);
105         return 0;
106     }

```

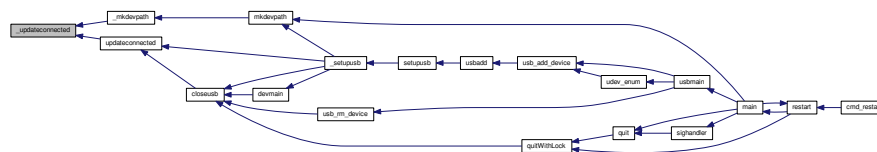


```

56 pthread_mutex_lock(&devmutex);
57 char cpath[strlen(&devpath) + 12];
58 snprintf(cpath, sizeof(cpath), "%s0/connected", &devpath);
59 FILE* cfile = fopen(cpath, "w");
60 if(!cfile){
61     ckb_warn("Unable to update %s: %s\n", cpath, strerror(errno));
62     pthread_mutex_unlock(&devmutex);
63     return;
64 }
65 int written = 0;
66 for(int i = 1; i < DEV_MAX; i++){
67     if(IS_CONNECTED(keyboard + i)){
68         written = 1;
69         fprintf(cfile, "%s%d %s %s\n", &devpath, i, &keyboard[i].serial,
keyboard[i].name);
70     }
71 }
72 if(!written)
73     fputc('\n', cfile);
74 fclose(cfile);
75 chmod(cpath, S_GID_READ);
76 if(gid >= 0)
77     chown(cpath, 0, gid);
78 pthread_mutex_unlock(&devmutex);
79 }

```

Here is the caller graph for this function:



5.8.3.5 int mkdevpath (usbdevice * kb)

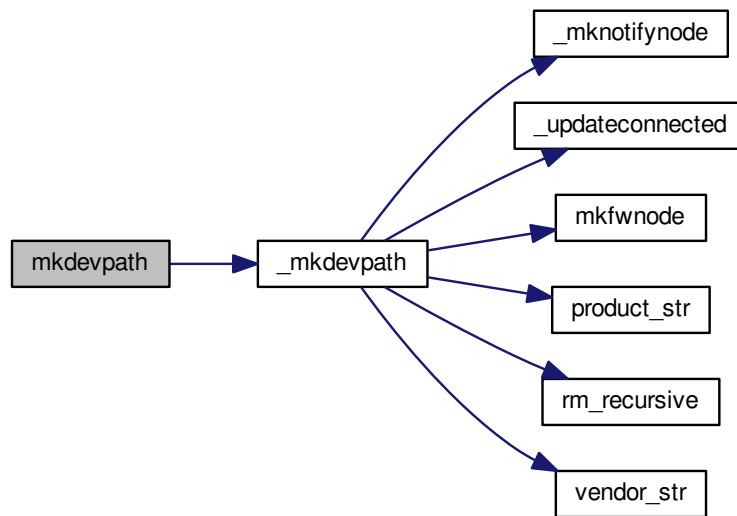
Definition at line 268 of file devnode.c.

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

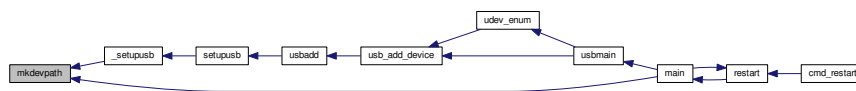
Referenced by `_setupusb()`, and `main()`.

```
268                                     {
269     uuid_guard_start;
270     int res = _mkdevpath(kb);
271     uuid_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 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()`.

```

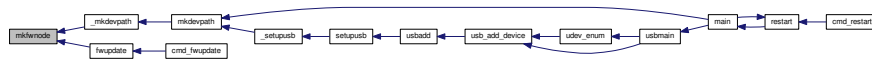
299     {
300         int index = INDEX_OF(kb, keyboard);
301         char fwpath[strlen(devpath) + 12];
302         snprintf(fwpath, sizeof(fwpath), "%s%d/fwversion", devpath, index);
303         FILE* fwfile = fopen(fwpath, "w");
304         if(fwfile){
305             fprintf(fwfile, "%04x", kb->fwversion);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

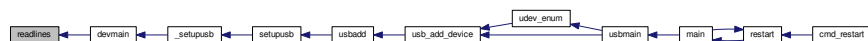
Referenced by `devmain()`.

```

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 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 void readlines_ctx_init (readlines_ctx * ctx)

Definition at line 341 of file devnode.c.

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 int rm_recursive (const char * *path*)

Definition at line 19 of file devnode.c.

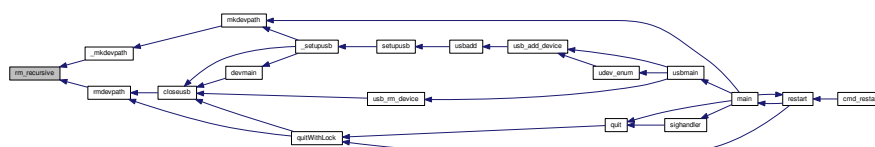
Referenced by `_mkdevpath()`, and `rmdevpath()`.

```

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         sprintf(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:



5.8.3.12 int rmdevpath (usbdevice * kb)

Definition at line 275 of file devnode.c.

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

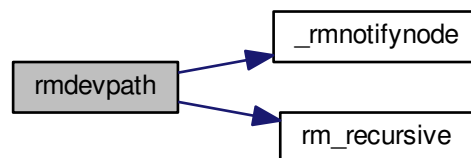
Referenced by `closeusb()`, and `quitWithLock()`.

```

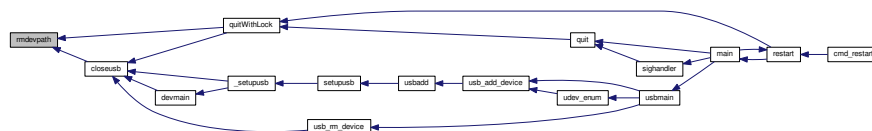
275     {
276         euid_guard_start;
277         int index = INDEX_OF(kb, keyboard);
278         if(kb->infifo != 0) {
279             #ifdef OS_LINUX
280                 write(kb->infifo - 1, "\n", 1); // hack to prevent the FIFO thread from perma-blocking
281             #endif
282             close(kb->infifo - 1);
283             kb->infifo = 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             euid_guard_stop;
292             return -1;
293         }
294         ckb_info("Removed device path %s\n", path);
295         euid_guard_stop;
296         return 0;
297     }

```

Here is the call graph for this function:



Here is the caller graph for this function:



5.8.3.13 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()`.


```

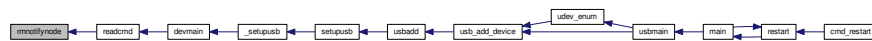
129                                     {
130     euid_guard_start;
131     int res = _rmnotifynode(kb, notify);
132     euid_guard_stop;
133     return res;
134 }

```

Here is the call graph for this function:



Here is the caller graph for this function:



5.8.3.14 void updateconnected ()

Definition at line 81 of file devnode.c.

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

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

```

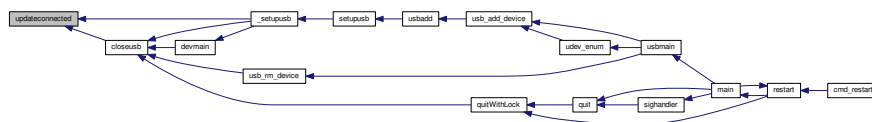
81     {
82     euid_guard_start;
83     _updateconnected();
84     euid_guard_stop;
85 }

```

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 `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()`.

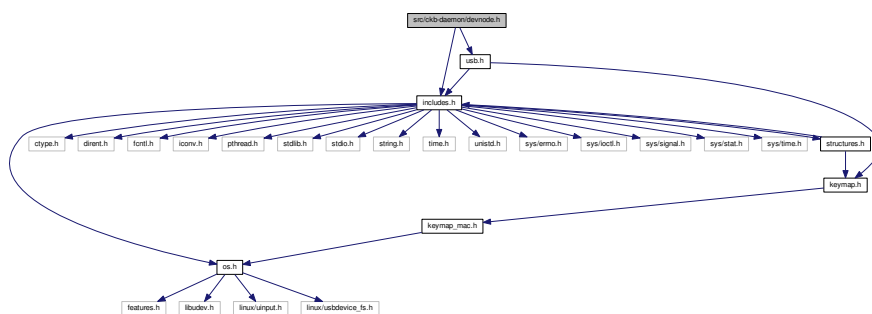
5.8.4.2 `long gid = -1`

Definition at line 16 of file `devnode.c`.

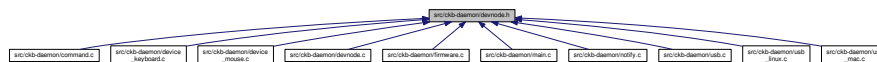
Referenced by `_mkdevpath()`, `_mknotifynode()`, `_updateconnected()`, `main()`, and `mkfwnode()`.

5.9 `src/ckb-daemon/devnode.h` File Reference

```
#include "includes.h"
#include "usb.h"
Include dependency graph for devnode.h:
```



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



Macros

- `#define S_READDIR (S_IRWXU | S_IRGRP | S_IROTH | S_IXGRP | S_IXOTH)`
- `#define S_READ (S_IRUSR | S_IRGRP | S_IROTH | S_IWUSR)`
- `#define S_READWRITE (S_IRUSR | S_IRGRP | S_IROTH | S_IWUSR | S_IWGRP | S_IWOTH)`
- `#define S_CUSTOM (S_IRUSR | S_IWUSR | S_IRGRP | S_IWGRP)`
- `#define S_CUSTOM_R (S_IRUSR | S_IWUSR | S_IRGRP)`

Typedefs

- `typedef struct _readlines_ctx * readlines_ctx`

Custom readline is needed for FIFOs. `fopen()/getline()` will die if the data is sent in too fast.

Functions

- void `updateconnected` ()
Update the list of connected devices.
- 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 `mknotifynode` (usbdevice *kb, int notify)
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 #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 #define S_CUSTOM_R (S_IRUSR | S_IWUSR | S_IRGRP)

Definition at line 18 of file devnode.h.

5.9.1.3 #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 #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 #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 typedef struct _readlines_ctx* readlines_ctx

Definition at line 39 of file devnode.h.

5.9.3 Function Documentation

5.9.3.1 int mkdevpath (usbdevice * kb)

Definition at line 268 of file devnode.c.

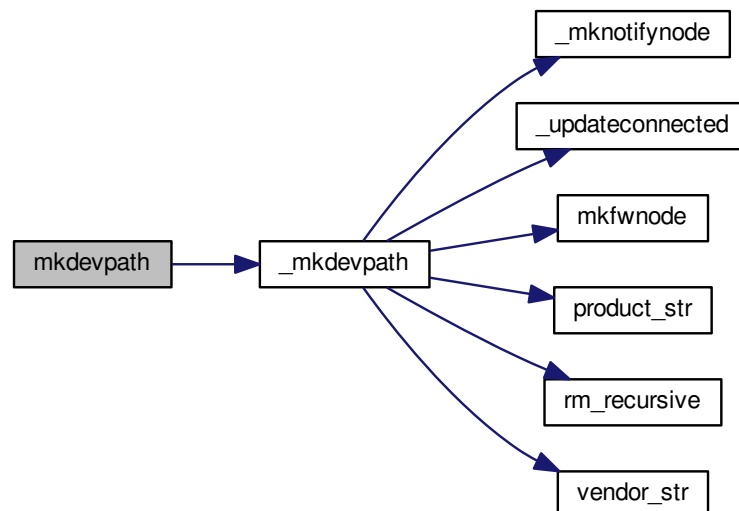
References `_mkdevpath()`, `euid_guard_start`, and `euid_guard_stop`.

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

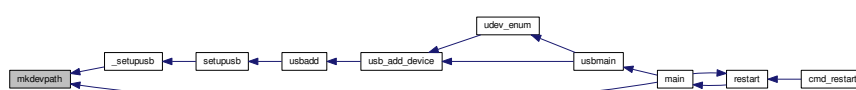
```

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 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.

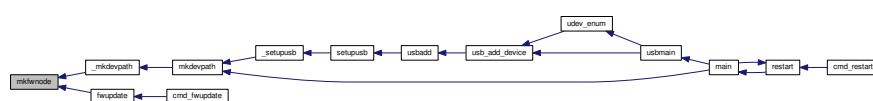
Referenced by `_mkdevpath()`, and `fwupdate()`.

```

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", ppath, 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 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     uuid_guard_start;
110     int res = _mknotifynode(kb, notify);
111     uuid_guard_stop;
112     return res;
113 }

```

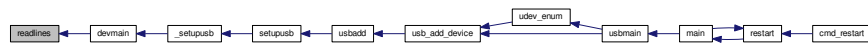


```

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 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 void readlines_ctx_init (readlines_ctx * ctx)

Definition at line 341 of file devnode.c.

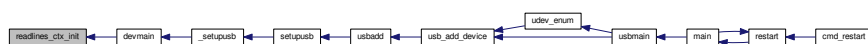
Referenced by `devmain()`.

```

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:




```

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 void updateconnected ()

Definition at line 81 of file devnode.c.

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

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

```

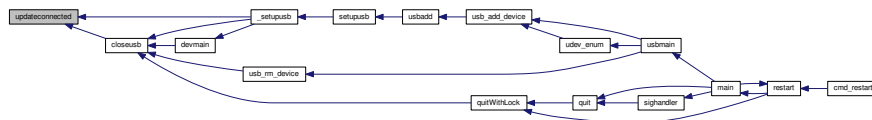
81     {
82     euid_guard_start;
83     _updateconnected();
84     euid_guard_stop;
85 }

```

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 `const char* const devpath`

Definition at line 8 of file `devnode.h`.

5.9.4.2 `long gid`

Definition at line 16 of file `devnode.c`.

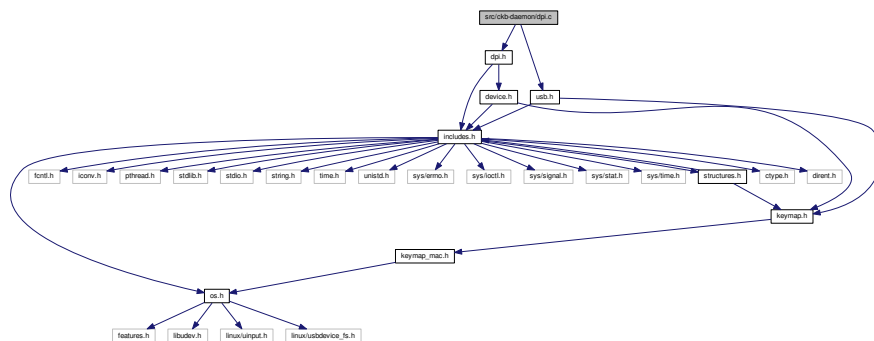
Referenced by `_mkdevpath()`, `_mknotifynode()`, `_updateconnected()`, `main()`, and `mkfwnode()`.

5.10 `src/ckb-daemon/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 `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, "%hu", &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 void cmd_dpiset (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, "%hu", &stagenum) != 1)
47         return;
48     if(stagenum > DPI_COUNT)
49         return;
50     mode->dpi.current = stagenum;
51 }

```

5.10.1.3 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, "%hu", &heightnum) != 1)
56         return;
57     if(heightnum > LIFT_MAX || heightnum < LIFT_MIN)
58         return;
59     mode->dpi.lift = heightnum;
60 }

```

5.10.1.4 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 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`.

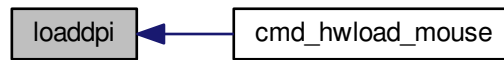
Referenced by `cmd_hwload_mouse()`.

```

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

Definition at line 69 of file dpi.c.

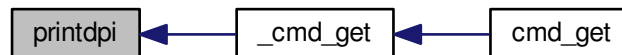
References `_readlines_ctx::buffer`, `DPI_COUNT`, `dpiset::enabled`, `dpiset::x`, and `dpiset::y`.

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->
84 y[i], &newlen);
85         length += newlen;
86     }
87     return buffer;
88 }
  
```

Here is the caller graph for this function:



5.10.1.7 `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`, `usbsend`, `dpiset::x`, and `dpiset::y`.

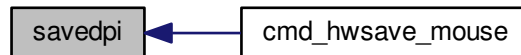
Referenced by `cmd_hwsave_mouse()`.

```

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(!usbSend(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:



5.10.1.8 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`, `usbSend`, `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, 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 }

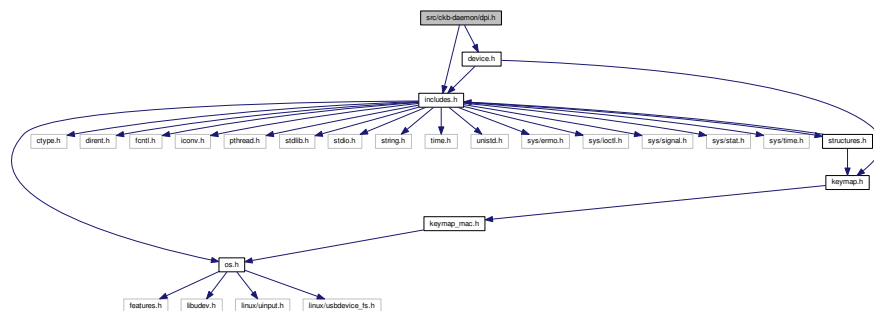
```

5.11 src/ckb-daemon/dpi.h File Reference

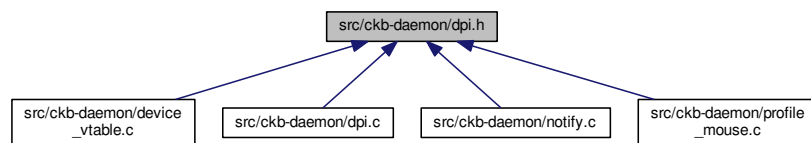
```
#include "includes.h"
```

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

Definition at line 152 of file dpi.c.

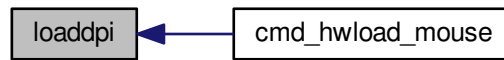
References `lighting::b`, `ckb_err`, `dpiset::current`, `DPI_COUNT`, `dpiset::enabled`, `lighting::g`, `LED_MOUSE`, `dpiset::lift`, `LIFT_MAX`, `LIFT_MIN`, `MSG_SIZE`, `N_MOUSE_ZONES`, `lighting::r`, `dpiset::snap`, `usbrecv`, `dpiset::x`, and `dpiset::y`.

Referenced by `cmd_hload_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] = 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:



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

Definition at line 69 of file `dpi.c`.

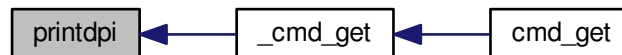
References `_readlines_ctx::buffer`, `DPI_COUNT`, `dpiset::enabled`, `dpiset::x`, and `dpiset::y`.

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 `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`, `usbsend`, `dpiset::x`, and `dpiset::y`.

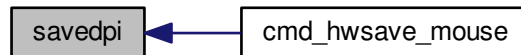
Referenced by `cmd_hwsave_mouse()`.

```

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(!usbsend(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 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`, `usbsend`, `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, 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 },

```

```

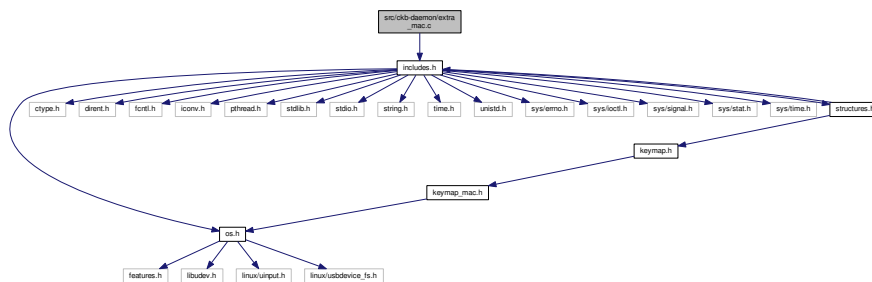
115     { 0x07, 0x13, 0x04, 0, newdpi->snap, 0x05 }
116 };
117 if(!usb_send(kb, data_pkt[0], 4))
118     return -2;
119 // Finished
120 memcpy(lastdpi, newdpi, sizeof(dpiset));
121 return 0;
122 }

```

5.12 src/ckb-daemon/extra_mac.c File Reference

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

Include dependency graph for extra_mac.c:



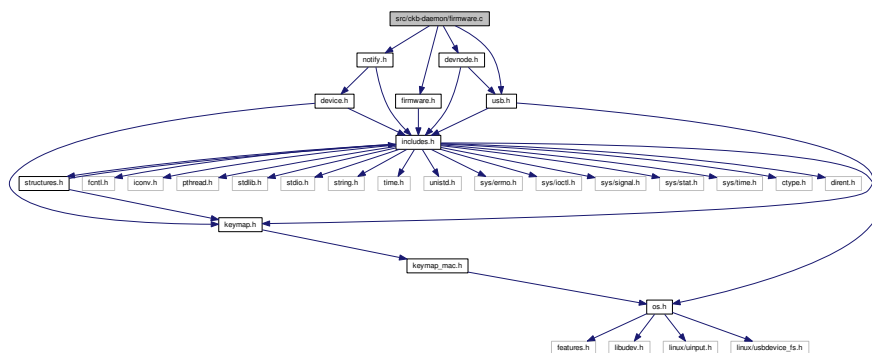
5.13 src/ckb-daemon/firmware.c File Reference

```

#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 #define FW_MAXSIZE (255 * 256)

Definition at line 51 of file firmware.c.

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

5.13.1.2 #define FW_NOFILE -1

Definition at line 7 of file firmware.c.

Referenced by [cmd_fwupdate\(\)](#), and [fwupdate\(\)](#).

5.13.1.3 #define FW_OK 0

Definition at line 6 of file firmware.c.

Referenced by [cmd_fwupdate\(\)](#), and [fwupdate\(\)](#).

5.13.1.4 #define FW_USBFAIL -3

Definition at line 9 of file firmware.c.

Referenced by [cmd_fwupdate\(\)](#), and [fwupdate\(\)](#).

5.13.1.5 #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 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\(\)](#).

```

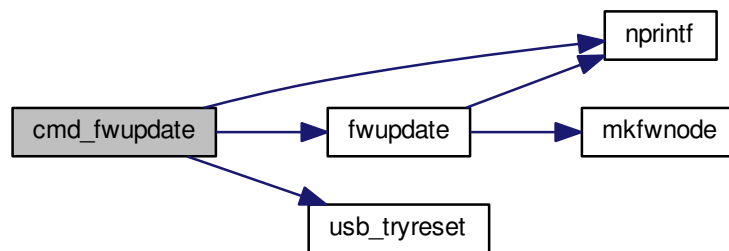
154
155     if(!HAS_FEATURES(kb, FEAT_FWUPDATE))
156         return 0;
157     // Update the firmware
158     int ret = fwupdate(kb, path, nnumber);
159     while(ret == FW_USBFAIL){
160         // Try to reset the device if it fails
161         if(usb_tryreset(kb))
162             break;
163         ret = fwupdate(kb, path, nnumber);
164     }
165     switch(ret){
166     case FW_OK:
167         nprintf(kb, nnumber, 0, "fwupdate %s ok\n", path);
168     }
```

```

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:



5.13.2.2 int fwupdate (usbdevice * kb, const char * path, int nnumber)

Definition at line 55 of file firmware.c.

References ckb_err, ckb_info, FW_MAXSIZE, FW_NOFILE, FW_OK, FW_USBFAIL, FW_WRONGDEV, usbdevice::fwversion, mkfwnode(), MSG_SIZE, nprintf(), usbdevice::product, usbdevice::usbdelay, usbdevice::usbdevice::vendor.

Referenced by cmd_fwupdate().

```

55     {
56         // Read the firmware from the given path
57         char* fwdata = calloc(1, FW_MAXSIZE + 256);
58         int fd = open(path, O_RDONLY);
59         if(fd == -1){
60             ckb_err("Failed to open firmware file %s: %s\n", path, strerror(errno));
61             return FW_NOFILE;
62         }
63         ssize_t length = read(fd, fwdata, FW_MAXSIZE + 1);
64         if(length <= 0x108 || length > FW_MAXSIZE){
65             ckb_err("Failed to read firmware file %s: %s\n", path, length <= 0 ? strerror(errno) : "
Wrong size");
66             close(fd);
67             return FW_NOFILE;
68         }
69         close(fd);
70
71         short vendor, product, version;
72         // Copy the vendor ID, product ID, and version from the firmware file
73         memcpy(&vendor, fwdata + 0x102, 2);
74         memcpy(&product, fwdata + 0x104, 2);
75         memcpy(&version, fwdata + 0x106, 2);
76         // Check against the actual device
77         if(vendor != kb->vendor || product != kb->product){
78             ckb_err("Firmware file %s doesn't match device (V: %04x P: %04x)\n", path, vendor, product);
79             return FW_WRONGDEV;
80         }
81         ckb_info("Loading firmware version %04x from %s\n", version, path);
82         nprintf(kb, nnumber, 0, "fwupdate %s 0/%d\n", path, (int)length);
83         // Force the device to 10ms delay (we need to deliver packets very slowly to make sure it doesn't get
overwhelmed)

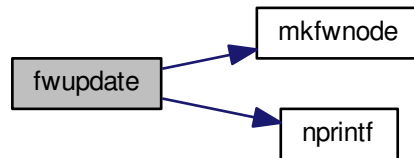
```

```

84     kb->usbdelay = 10;
85     // Send the firmware messages (256 bytes at a time)
86     uchar data_pkt[7][MSG_SIZE] = {
87         { 0x07, 0x0c, 0xf0, 0x01, 0 },
88         { 0x07, 0x0d, 0xf0, 0 },
89         { 0x7f, 0x01, 0x3c, 0 },
90         { 0x7f, 0x02, 0x3c, 0 },
91         { 0x7f, 0x03, 0x3c, 0 },
92         { 0x7f, 0x04, 0x3c, 0 },
93         { 0x7f, 0x05, 0x10, 0 }
94     };
95     int output = 0, last = 0;
96     int index = 0;
97     while(output < length){
98         int npackets = 1;
99         // Packet 1: data position
100        data_pkt[1][6] = index++;
101        while(output < length){
102            npackets++;
103            if(npackets != 6){
104                // Packets 2-5: 60 bytes of data
105                memcpy(data_pkt[npackets] + 4, fwdata + output, 60);
106                last = output;
107                output += 60;
108            } else {
109                // Packet 6: 16 bytes
110                memcpy(data_pkt[npackets] + 4, fwdata + output, 16);
111                last = output;
112                output += 16;
113                break;
114            }
115        }
116        if(index == 1){
117            if(!usb_send(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(!usb_send(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(!usb_send(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(!usb_send(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 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){

```

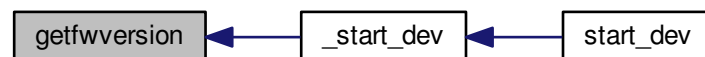


```

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 }

```

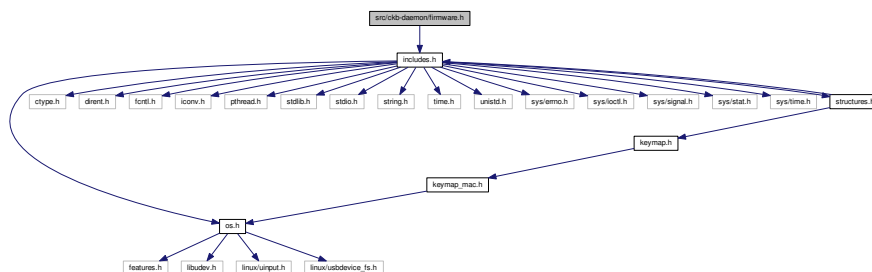
Here is the caller graph for this function:



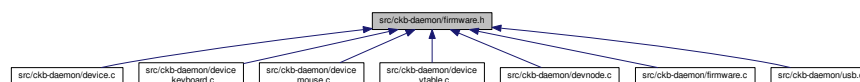
5.14 src/ckb-daemon/firmware.h File Reference

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

Definition at line 154 of file firmware.c.

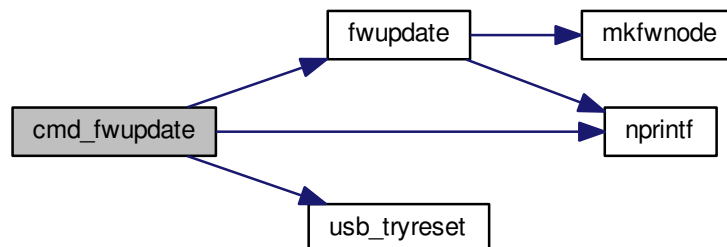
References `FEAT_FWUPDATE`, `FW_NOFILE`, `FW_OK`, `FW_USBFAIL`, `FW_WRONGDEV`, `fwupdate()`, `HAS_FEATURES`, `nprintf()`, and `usb_tryreset()`.

```

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 `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;

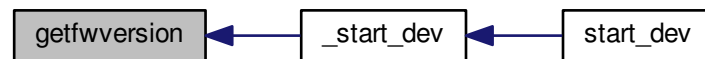
```

```

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:



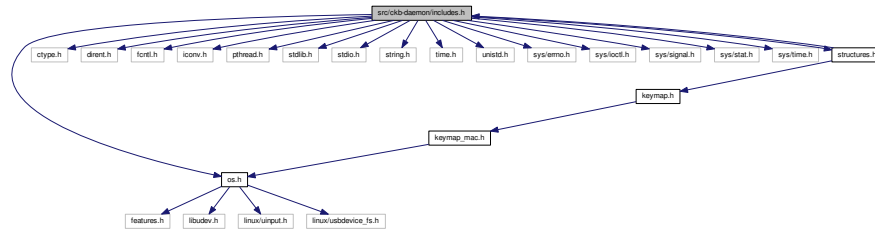
5.15 src/ckb-daemon/includes.h File Reference

```

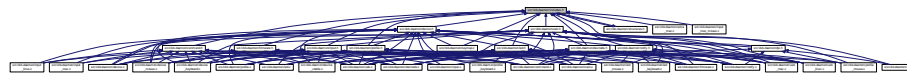
#include "os.h"
#include <ctype.h>
#include <dirent.h>
#include <fcntl.h>
#include <iconv.h>
#include <pthread.h>
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <time.h>
#include <unistd.h>
#include <sys/errno.h>
#include <sys/ioctl.h>
#include <sys/signal.h>
#include <sys/stat.h>
#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 `#define __FILE_NOPATH__ (strchr(__FILE__, '/') ? strchr(__FILE__, '/') + 1 : __FILE__)`

Definition at line 40 of file includes.h.

5.15.1.2 `#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()`, and `usbadd()`.

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

Definition at line 46 of file includes.h.

Referenced by `usbmain()`.

5.15.1.6 `#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 `#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 `#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 `#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 `#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 `#define ckb_s_err stdout`

Definition at line 36 of file includes.h.

5.15.1.12 `#define ckb_s_out stdout`

Definition at line 35 of file includes.h.

5.15.1.13 `#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 `_mkdevpath()`, `_mknotifynode()`, `_start_dev()`, `_updateconnected()`, `getfwversion()`, `hid_kb_translate()`, `isync()`, `mkfwnode()`, `os_inputclose()`, `os_keypress()`, `os_mousemove()`, `os_setupusb()`, `readlines()`, `rmdevpath()`, `uinputopen()`, and `usbmain()`.

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

Definition at line 63 of file includes.h.

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

Definition at line 62 of file includes.h.

5.15.2 Typedef Documentation

5.15.2.1 `typedef unsigned char uchar`

Definition at line 24 of file includes.h.

5.15.2.2 `typedef unsigned short ushort`

Definition at line 25 of file includes.h.

5.15.3 Function Documentation

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

Definition at line 19 of file main.c.

```

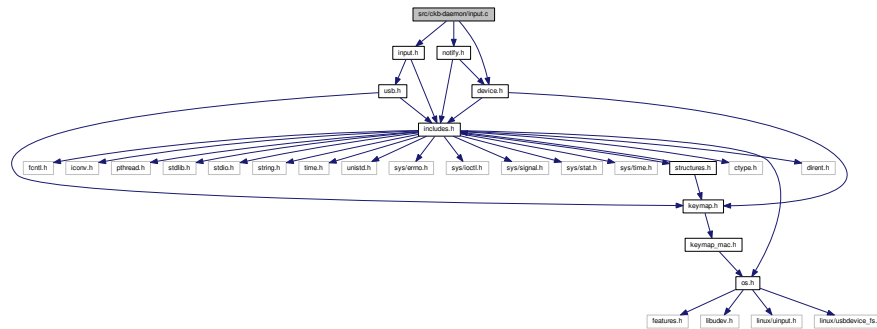
19                                     {
20     nanoseconds += timespec->tv_nsec;
21     timespec->tv_sec += nanoseconds / 1000000000;
22     timespec->tv_nsec = nanoseconds % 1000000000;
23 }
```

5.16 src/ckb-daemon/input.c File Reference

```

#include "device.h"
#include "input.h"
#include "notify.h"
```

Include dependency graph for input.c:



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 `#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 `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
228     binding* bind = &mode->bind;
229     if(!keys && !assignment){
230         // Null strings = "macro clear" -> erase the whole thing
231         for(int i = 0; i < bind->macrocount; i++)
232             free(bind->macros[i].actions);
233         bind->macrocount = 0;
234         return;
235     }
236     if(bind->macrocount >= MACRO_MAX)
237         return;
238     // Create a key macro
239     keymacro macro;
240     memset(&macro, 0, sizeof(macro));
241     // Scan the left side for key names, separated by +
242     int empty = 1;
243     int left = strlen(keys), right = strlen(assignment);
244     int position = 0, field = 0;
245     char keyname[12];
246     while(position < left && sscanf(keys + position, "%10[^+]\n", keyname, &field) == 1){
247         int keycode;
248         if((sscanf(keyname, "%d", &keycode) && keycode >= 0 && keycode <
N_KEYS_INPUT)
249             || (sscanf(keyname, "%x%x", &keycode) && keycode >= 0 && keycode <
N_KEYS_INPUT)){
250             // Set a key numerically
251             SET_KEYBIT(macro.combo, keycode);
252             empty = 0;
253         } else {
254             // Find this key in the keymap
255             for(unsigned i = 0; i < N_KEYS_INPUT; i++){
256                 if(keymap[i].name && !strcmp(keyname, keymap[i].name)){
257                     macro.combo[i / 8] |= 1 << (i % 8);
258                     empty = 0;
259                     break;
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)
286                     || (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         // 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 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%x", &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 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:



5.16.2.4 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 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 void freebind (binding * bind)

Definition at line 181 of file input.c.

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

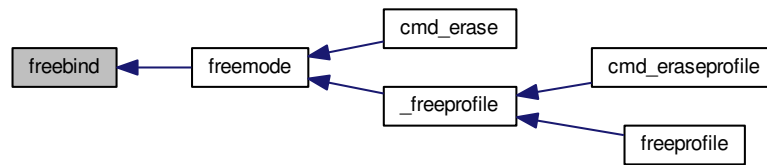
Referenced by `freemode()`.

```

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 void initbind (binding * bind)

Definition at line 173 of file input.c.

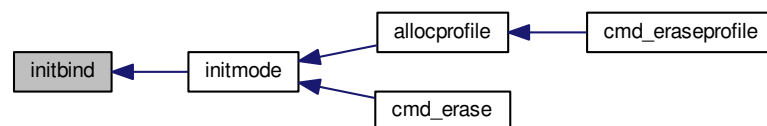
References `binding::base`, `keymap`, `binding::macrocap`, `binding::macrocount`, `binding::macros`, `N_KEYS_INPUT`, and `key::scan`.

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 void inputupdate (usbdevice * kb)

Definition at line 122 of file input.c.

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

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

```

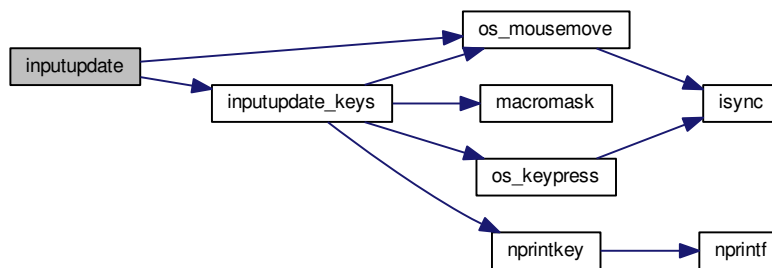
122     {
123     #ifdef OS_LINUX
124         if ((!kb->uinput_kb || !kb->uinput_mouse)
125         #else
126             if (!kb->event
127         #endif
128             || !kb->profile)
129             return;
  
```

```

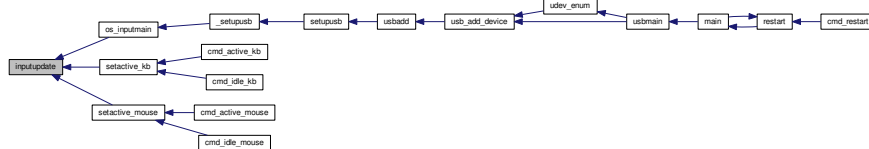
130 // Process key/button input
131 inputupdate_keys(kb);
132 // Process mouse movement
133 usbinput* input = &kb->input;
134 if(input->rel_x != 0 || input->rel_y != 0){
135     os_mousemove(kb, input->rel_x, input->rel_y);
136     input->rel_x = input->rel_y = 0;
137 }
138 // Finish up
139 memcpy(input->prevkeys, 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 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->
rel_y);
36             }
37             else {
38                 os_keypress(kb, action->scan, action->
down);
39                 if (kb->delay) {
40                     if (a > 200) usleep (100);
41                     else if (a > 20) usleep(30);
42                 }
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                         }
82                     } else {
83                         // Modifier up: Add to the end of everything
84                         events[modcount + keycount + rmodcount++] = -(scancode + 1);
85                     }
86                 } else {
87                     // Regular keypress: add to the end of regular keys
88                     for(int i = rmodcount; i > 0; i--){
89                         events[modcount + keycount + i] = events[modcount + keycount + i - 1];
90                         events[modcount + keycount++] = new ? (scancode + 1) : -(scancode + 1);
91                         // The volume wheel and the mouse wheel don't generate keyups, so create them
92                         // automatically
93                     }
94                 }
95             }
96             #define IS_WHEEL(scan, kb) (((scan) == KEY_VOLUMEUP || (scan) == KEY_VOLUMEDOWN || (scan) == BTN_WHEELUP || (scan) == BTN_WHEELDOWN) && !IS_K65(kb))
97             if(new && IS_WHEEL(map->scan, kb)){
98                 for(int i = rmodcount; i > 0; i--){
99                     events[modcount + keycount + i] = events[modcount + keycount + i - 1];
100                     events[modcount + keycount++] = -(scancode + 1);
101                     input->keys[byte] &= ~mask;
102                 }
103             }
104             // Print notifications if desired
105             if(kb->active){
106                 for(int notify = 0; notify < OUTFIFO_MAX; notify++){
107                     if(mode->notify[notify][byte] & mask){
108                         nprintkey(kb, notify, keyindex, new);
109                         // Wheels doesn't generate keyups
110                         if(new && IS_WHEEL(map->scan, kb))
111                             nprintkey(kb, notify, keyindex, 0);
112                     }
113                 }
114             }
115         }
116     }
117 }

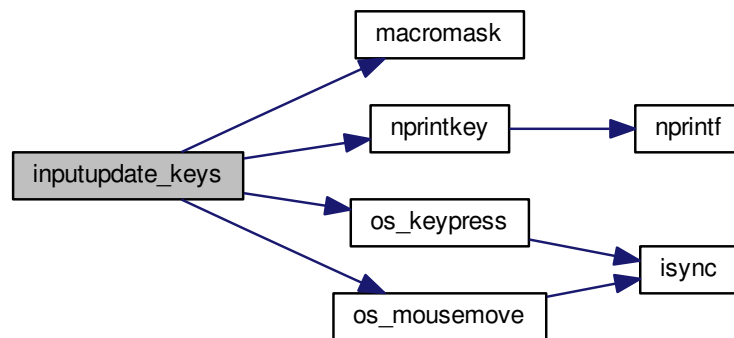
```

```

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

Definition at line 5 of file input.c.

References N_KEYBYTES_INPUT.

Referenced by inputupdate_keys().

```

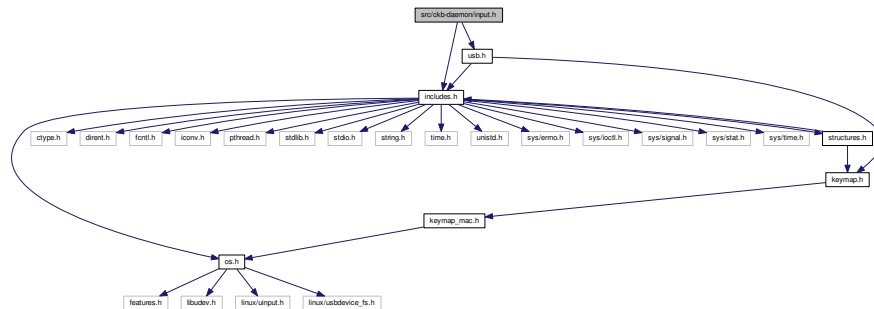
5     {
6     // Scan a macro against key input. Return 0 if any of them don't match
7     for(int i = 0; i < N_KEYBYTES_INPUT; i++){
8         // if((key1[i] & key2[i]) != key2[i])
9         if(key1[i] != key2[i]) // Changed to detect G-keys + modifiers
10            return 0;
11    }
12    return 1;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

```



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

Include dependency graph for input.h:



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



Macros

- #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)

Functions

- int **os_inputopen** (usbdevice *kb)
 os_inputopen
- void **os_inputclose** (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 *ignored)
- void **cmd_rebind** (usbdevice *kb, usbmode *mode, int dummy, int keyindex, const char *ignored)
- void **cmd_macro** (usbdevice *kb, usbmode *mode, const int notifynumber, const char *keys, const char *assignment)
- void **os_keypress** (usbdevice *kb, int scancode, int down)
- void **os_mousemove** (usbdevice *kb, int x, int y)
- int **os_setupindicators** (usbdevice *kb)

5.17.1 Macro Definition Documentation

- 5.17.1.1 #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 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 }

```

5.17.2.2 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:



5.17.2.3 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(&mutex(kb));
222     mode->bind.base[keyindex] = keymap[keyindex].scan;
223     pthread_mutex_unlock(&mutex(kb));
224 }

```

5.17.2.4 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, mutex, KEY_UNBOUND, and N_KEYS_INPUT.

```

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

```

5.17.2.5 void freebind (binding * bind)

Definition at line 181 of file input.c.

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

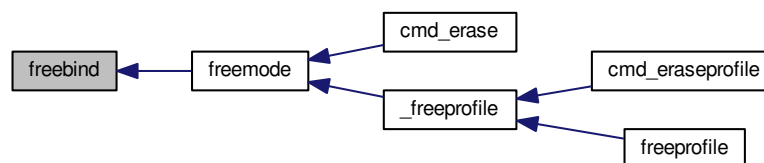
Referenced by freemode().

```

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 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++)

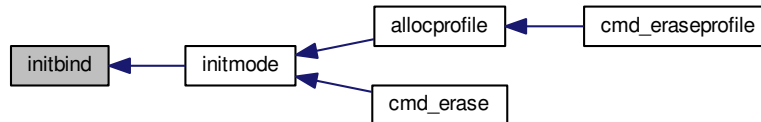
```

```

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 void inputupdate (usbdevice * kb)

Definition at line 122 of file input.c.

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

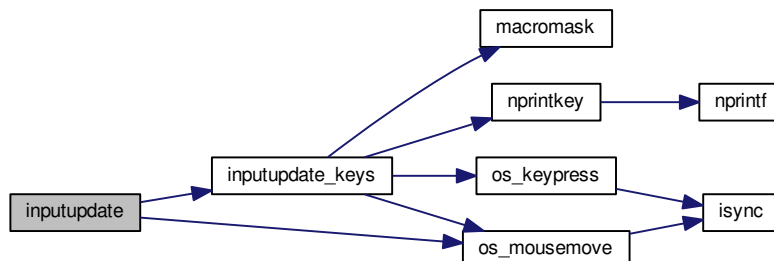
Referenced by `os_inputmain()`, `setactive_kb()`, and `setactive_mouse()`.

```

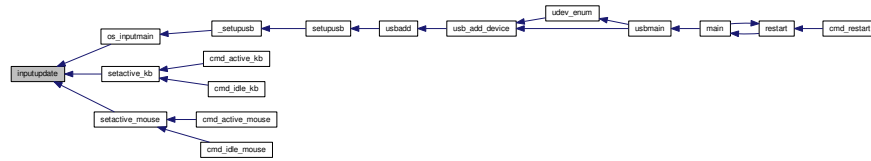
122     {
123     #ifdef OS_LINUX
124         if ((!kb->uinput_kb || !kb->uinput_mouse)
125     #else
126         if (!kb->event
127     #endif
128         || !kb->profile)
129         return;
130         // Process key/button input
131         inputupdate_keys(kb);
132         // Process mouse movement
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 void os_inputclose (usbdevice * kb)

Definition at line 76 of file input_linux.c.

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

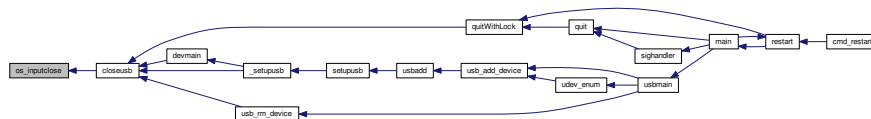
Referenced by closeusb().

```

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:



5.17.2.9 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:



5.17.2.10 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`.

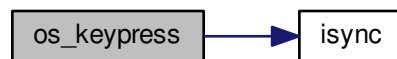
Referenced by `inputupdate_keys()`.

```

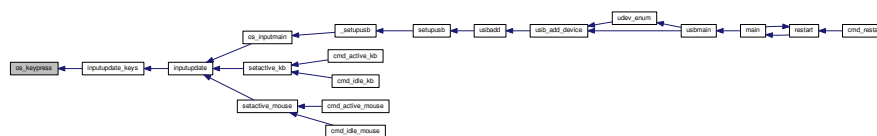
118                                     {
119     struct input_event event;
120     memset(&event, 0, sizeof(event));
121     int is_mouse = 0;
122     if(scancode == BTN_WHEELUP || scancode == BTN_WHEELDOWN){
123         // The mouse wheel is a relative axis
124         if(!down)
125             return;
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:



5.17.2.11 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()`, and `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     }

```


Here is the caller graph for this function:



5.17.2.13 void updateindicators_kb (usbdevice * kb, int force)

Definition at line 142 of file input.c.

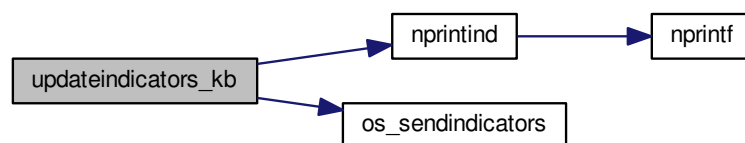
References `usbdevice::active`, `usbprofile::currentmode`, `DELAY_SHORT`, `usbdevice::hw_ileds`, `usbdevice::hw_ileds_old`, `I_CAPS`, `I_NUM`, `I_SCROLL`, `usbdevice::ileds`, `usbmode::inotify`, `usbmode::ioff`, `usbmode::ion`, `nprintind()`, `os_sendindicators()`, `OUTFIFO_MAX`, and `usbdevice::profile`.

```

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:



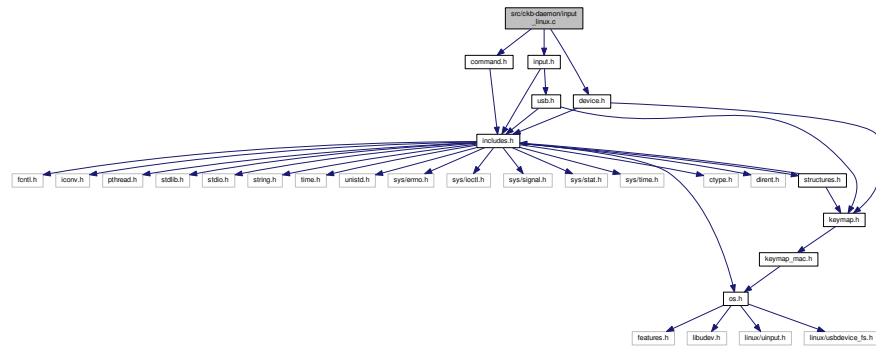
5.18 src/ckb-daemon/input_linux.c File Reference

```

#include "command.h"
#include "device.h"
#include "input.h"

```

Include dependency graph for `input_linux.c`:



Functions

- `int uinputopen` (`struct uinput_user_dev *indev`, `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 `void* _ledthread (void * ctx)`

Definition at line 165 of file `input_linux.c`.

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

Referenced by `os_setupindicators()`.

```

165                                     {
166     usbdevice* kb = ctx;
167     uchar ileds = 0;
168     // Read LED events from the uinput device
169     struct input_event;
170     while (read(kb->uinput_kb - 1, &event, sizeof(event)) > 0) {
171         if (event.type == EV_LED && event.code < 8){
172             char which = 1 << event.code;
173             if(event.value)
174                 ileds |= which;
175             else
176                 ileds &= ~which;
177         }
178         // Update them if needed
179         pthread_mutex_lock(&dmutex(kb));
180         if(kb->hw_ileds != ileds){
181             kb->hw_ileds = ileds;
182             kb->vtable->updateindicators(kb, 0);
183         }
184         pthread_mutex_unlock(&dmutex(kb));
185     }
186     return 0;
187 }
```

Here is the caller graph for this function:



5.18.1.2 static void isync (usbdevice * kb) [static]

Definition at line 107 of file input_linux.c.

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

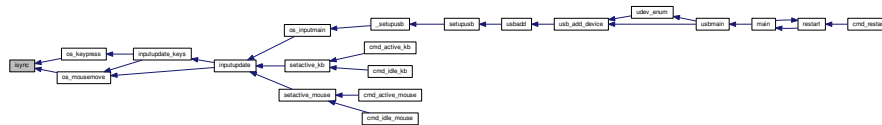
Referenced by os_keypress(), and os_mousemove().

```

107     {
108         struct input_event event;
109         memset(&event, 0, sizeof(event));
110         event.type = EV_SYN;
111         event.code = SYN_REPORT;
112         if(write(kb->uinput_kb - 1, &event, sizeof(event)) <= 0)
113             ckb_warn("uinput write failed: %s\n", strerror(errno));
114         if(write(kb->uinput_mouse - 1, &event, sizeof(event)) <= 0)
115             ckb_warn("uinput write failed: %s\n", strerror(errno));
116     }

```

Here is the caller graph for this function:



5.18.1.3 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;

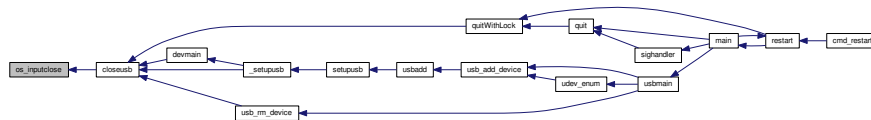
```

```

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 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 void os_keypress (usbdevice * kb, int scandcode, 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()`.

```

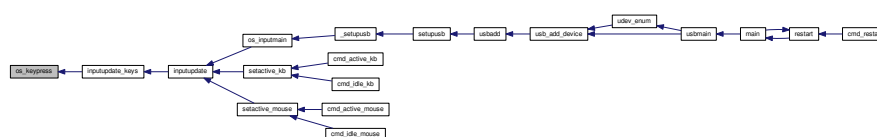
118                                     {
119     struct input_event event;
120     memset(&event, 0, sizeof(event));
121     int is_mouse = 0;
122     if(scancode == BTN_WHEELUP || scancode == BTN_WHEELDOWN){
123         // The mouse wheel is a relative axis
124         if(!down)
125             return;
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:

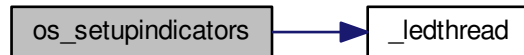



```

196         return err;
197     pthread_detach(thread);
198     return 0;
199 }

```

Here is the call graph for this function:



Here is the caller graph for this function:



5.18.1.8 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().

```

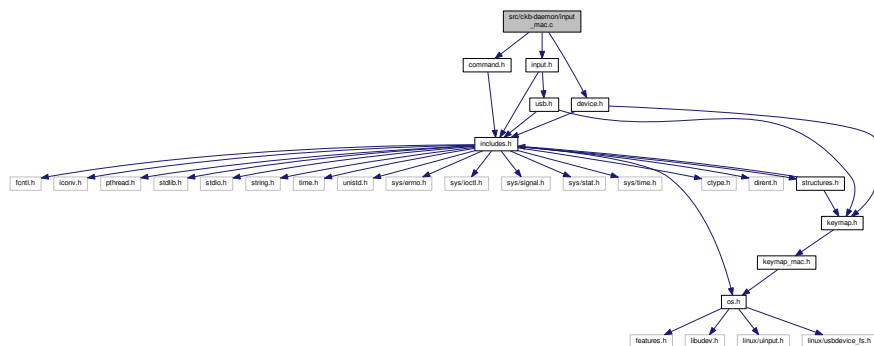
9
10 int fd = open("/dev/uinput", O_RDWR);
11 if(fd < 0){
12     // If that didn't work, try /dev/input/uinput instead
13     fd = open("/dev/input/uinput", O_RDWR);
14     if(fd < 0){
15         ckb_err("Failed to open uinput: %s\n", strerror(errno));
16         return 0;
17     }
18 }
19 // Enable 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 }

```



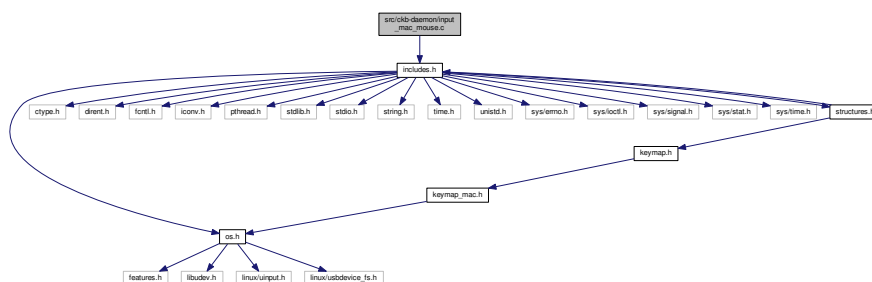
5.19 src/ckb-daemon/input_mac.c File Reference

```
#include "command.h"
#include "device.h"
#include "input.h"
Include dependency graph for input_mac.c:
```



5.20 src/ckb-daemon/input_mac_mouse.c File Reference

```
#include "includes.h"
Include dependency graph for input_mac_mouse.c:
```



5.21 src/ckb-daemon/keymap.c File Reference

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


Here is the caller graph for this function:



5.21.2.2 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 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 scan codes (-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, 98, -2, -2, -2, -2, -2, 97,
237         130, 131, -1, -1, -1, -2, -1, -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, -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); // voldown
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);    // voidn
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 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         if((endpoint != 2 && endpoint != -2) || length < 10)
368             return;
369         // EP 2: mouse input
370         if(urbinput[0] != 1)
371             return;
372         // Byte 1 = mouse buttons (bitfield)
373         for(int bit = 0; bit < BUTTON_HID_COUNT; bit++){
374             if(urbinput[1] & (1 << bit))
375                 SET_KEYBIT(kbinput, MOUSE_BUTTON_FIRST + bit);
376             else
377                 CLEAR_KEYBIT(kbinput, MOUSE_BUTTON_FIRST + bit);
378         }
379         // Bytes 5 - 8: movement
380         *xaxis += *(short*)(urbinput + 5);
381         *yaxis += *(short*)(urbinput + 7);
382         // Byte 9: wheel
383         char wheel = urbinput[9];
384         if(wheel > 0)
385             SET_KEYBIT(kbinput, MOUSE_EXTRA_FIRST);    // wheelup
386         else
387             CLEAR_KEYBIT(kbinput, MOUSE_EXTRA_FIRST);
388         if(wheel < 0)

```

```

389         SET_KEYBIT(kbinput, MOUSE_EXTRA_FIRST + 1);    // wheeldn
390     else
391         CLEAR_KEYBIT(kbinput, MOUSE_EXTRA_FIRST + 1);
392 }

```

Here is the caller graph for this function:



5.21.3 Variable Documentation

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

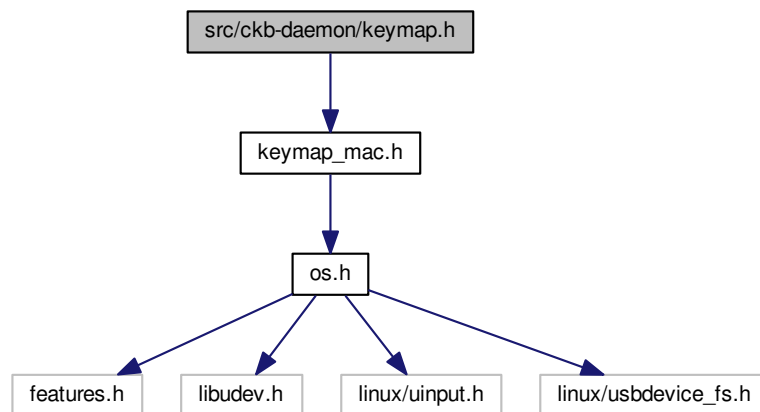
Definition at line 5 of file keymap.c.

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

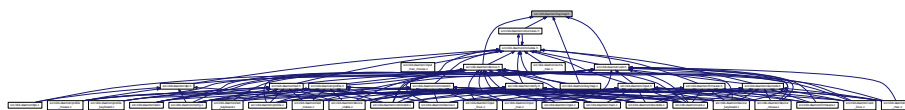
5.22 src/ckb-daemon/keymap.h File Reference

```
#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

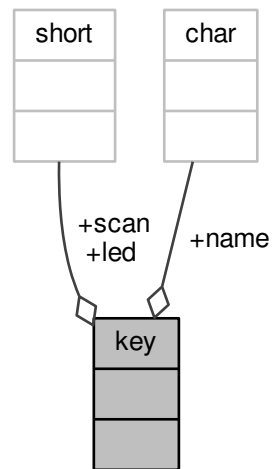
- const [key keymap](#) [(((152+3+12)+25)+11)]

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 #define BTN_WHEELDOWN 0x1f02

Definition at line 13 of file keymap.h.

Referenced by os_keypress().

5.22.2.2 #define BTN_WHEELUP 0x1f01

Definition at line 12 of file keymap.h.

Referenced by os_keypress().

5.22.2.3 #define KEY_BACKSLASH_ISO KEY_BACKSLASH

Definition at line 20 of file keymap.h.

5.22.2.4 #define KEY_CORSAIR -2

Definition at line 8 of file keymap.h.

5.22.2.5 #define KEY_NONE -1

Definition at line 7 of file keymap.h.

5.22.2.6 #define KEY_UNBOUND -3

Definition at line 9 of file keymap.h.

Referenced by cmd_unbind().

5.22.2.7 #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 #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 #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 #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 #define N_BUTTONS_EXTENDED 25

Definition at line 32 of file keymap.h.

5.22.2.12 #define N_BUTTONS_HW 20

Definition at line 31 of file keymap.h.

Referenced by corsair_mousecopy().

5.22.2.13 #define N_KEY_ZONES 3

Definition at line 27 of file keymap.h.

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

Definition at line 46 of file keymap.h.

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

Definition at line 37 of file keymap.h.

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

5.22.2.17 #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 #define N_KEYS_EXTRA 12

Definition at line 29 of file keymap.h.

5.22.2.19 #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 #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 #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 #define N_MOUSE_ZONES_EXTENDED 11

Definition at line 41 of file keymap.h.

5.22.2.23 #define SCAN_KBD 0

Definition at line 57 of file keymap.h.

5.22.2.24 #define SCAN_MOUSE 0x1000

Definition at line 58 of file keymap.h.

Referenced by os_keypress().

5.22.2.25 #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 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 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 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:



5.22.3.4 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 if(wheel < 0)
388                 SET_KEYBIT(kbinput, MOUSE_EXTRA_FIRST + 1); // wheeldn
389             else
390                 CLEAR_KEYBIT(kbinput, MOUSE_EXTRA_FIRST + 1);
391         }
392 }

```

Here is the caller graph for this function:



5.22.4 Variable Documentation

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

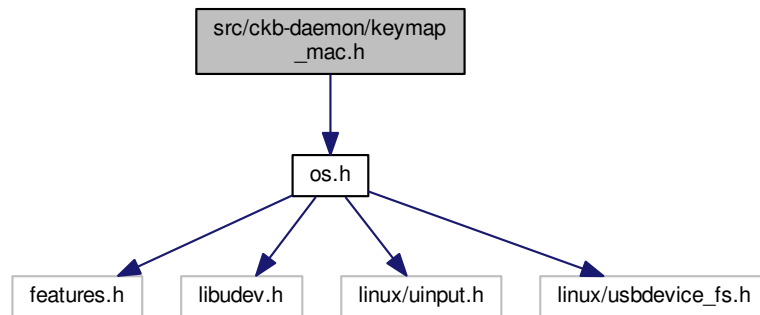
Definition at line 5 of file `keymap.c`.

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

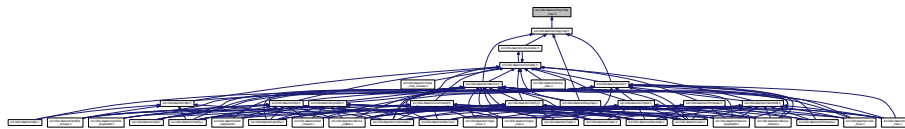
5.23 src/ckb-daemon/keymap_mac.h File Reference

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

Include dependency graph for keymap_mac.h:



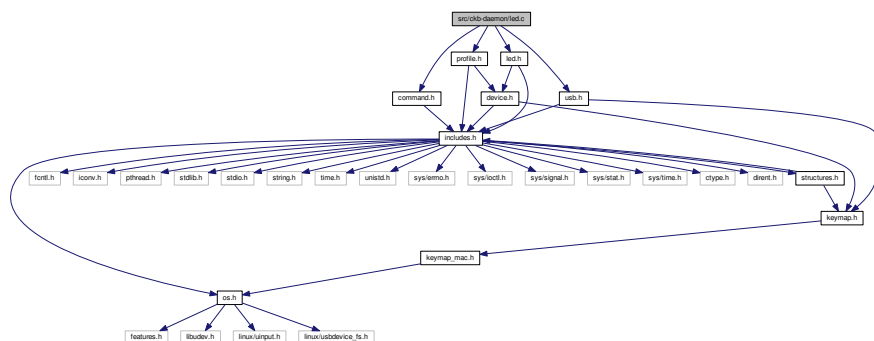
This graph shows which files directly or indirectly include this file:



5.24 src/ckb-daemon/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_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*)
- 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 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 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 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 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 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 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 static uchar iselect (const char * led) [static]

Definition at line 25 of file led.c.

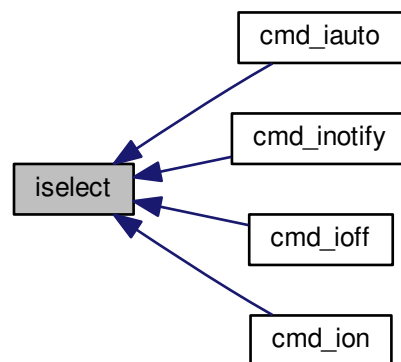
References I_CAPS, I_NUM, and I_SCROLL.

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

```

25         {
26     int result = 0;
27     if(!strcmp(led, "num", 3) || strstr(led, ",num"))
28         result |= I_NUM;
29     if(!strcmp(led, "caps", 4) || strstr(led, ",caps"))
30         result |= I_CAPS;
31     if(!strcmp(led, "scroll", 6) || strstr(led, ",scroll"))
32         result |= I_SCROLL;
33     if(!strcmp(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 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, 10);
125     }
126 }
```

```

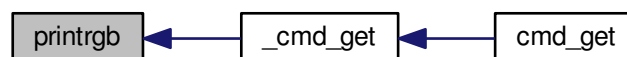
123         strncpy(names[i], name, 11);
124     }
125     // Check to make sure these aren't all the same color
126     int same = 1;
127     for(int i = 1; i < N_KEYS_EXTENDED; i++){
128         if(!names[i][0])
129             continue;
130         if(r[i] != r[0] || g[i] != g[0] || b[i] != b[0]){
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:

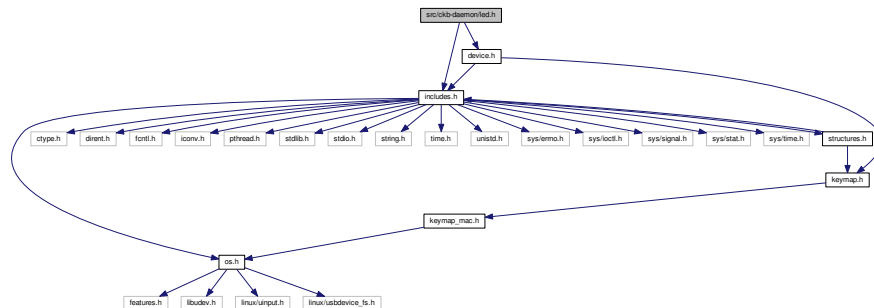


5.25 src/ckb-daemon/led.h File Reference

```
#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 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 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 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 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 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 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
220 ], 4)) {
221                     ckb_err("Bad input header\n");
222                     ckb_err("color = %d, i = %d, mode = %d\nInput (Antwort): %2.2x %2.2x %2.2x %2.2x
223 %2.2x %2.2x %2.2x %2.2x\nOutput (Frage): %2.2x %2.2x %2.2x %2.2x\n", clr, i, mode,
224 in_pkt[i][0], in_pkt[i][1], in_pkt[i][2], in_pkt[i][3], in_pkt[i][4], in_pkt[i][5],
225 in_pkt[i][6], in_pkt[i][7],
226 // data_pkt[i + clr * 4][0], data_pkt[i + clr * 4 ][1], data_pkt[i + clr *
227 4 ][2], data_pkt[i + clr * 4 ][3]);
228 cmp_pkt[i][0], cmp_pkt[i][1], cmp_pkt[i][2], cmp_pkt[i][3]);
229 in_pkt[2][0] = 0x99;
230 in_pkt[2][1] = 0x99;
231 in_pkt[2][2] = 0x99;
232 in_pkt[2][3] = 0x99;
233 usbrecv(kb, in_pkt[2], in_pkt[2]); // just to find it in the wireshark log
234 return -1;
235 }
236 }
237 // Copy colors to lighting. in_pkt[0] is irrelevant.
238 memcpy(colors[clr], in_pkt[1] + 4, 60);
239 memcpy(colors[clr] + 60, in_pkt[2] + 4, 60);
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 },

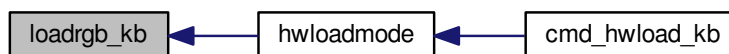
```

```

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:



5.25.1.7 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 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;
131             if(r[i] != r[0] || g[i] != g[0] || b[i] != b[0]){
132                 same = 0;
133                 break;
134             }
135         }
136     }

```

```

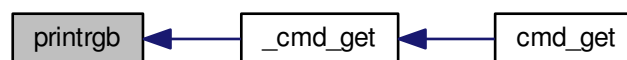
135 // If they are, just output that color
136 if(same){
137     char* buffer = malloc(7);
138     snprintf(buffer, 7, "%02x%02x%02x", r[0], g[0], b[0]);
139     return buffer;
140 }
141 const int BUFFER_LEN = 4096; // Should be more than enough to fit all keys
142 char* buffer = malloc(BUFFER_LEN);
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 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 `usbsend`.

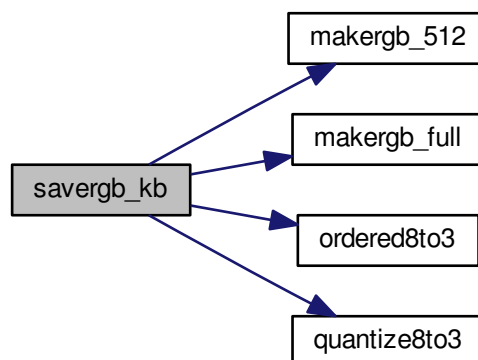
Referenced by `cmd_hwsave_kb()`.

```

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(!usbSEND(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(!usbSEND(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(!usbSEND(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 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:



5.25.1.11 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.

```

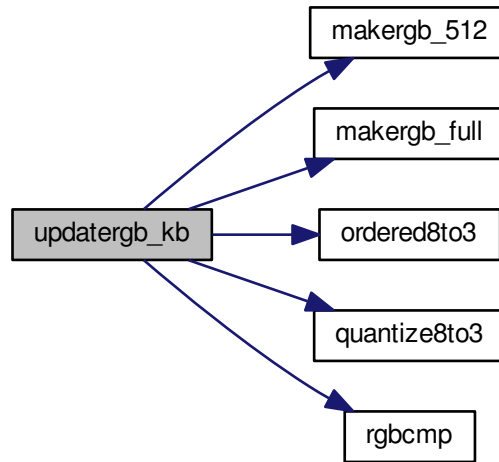
77
{
  
```

```

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 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 `usb send`.

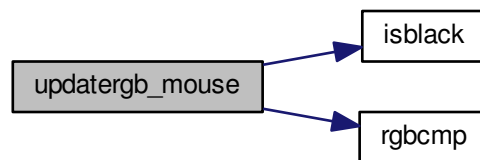
```

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
    && !rgbcmp(lastlight, newlight))
27         return 0;
28     lastlight->forceupdate = newlight->forceupdate = 0;
29
30     // Send the RGB values for each zone to the mouse
31     uchar data_pkt[2][MSG_SIZE] = {
32         { 0x07, 0x22, N_MOUSE_ZONES, 0x01, 0 }, // RGB colors
33         { 0x07, 0x05, 0x02, 0 } // Lighting on/off
34     };
35     uchar* rgb_data = &data_pkt[0][4];
36     for(int i = 0; i < N_MOUSE_ZONES; i++){
37         *rgb_data++ = i + 1;
38         *rgb_data++ = newlight->r[LED_MOUSE + i];
39         *rgb_data++ = newlight->g[LED_MOUSE + i];
40         *rgb_data++ = newlight->b[LED_MOUSE + i];
41     }
42     // Send RGB data
43     if(!usb send(kb, data_pkt[0], 1))
44         return -1;
45     int was_black = isblack(kb, lastlight), is_black = isblack(kb, newlight);
46     if(is_black){
47         // If the lighting is black, send the deactivation packet (M65 only)
48         if(!usb send(kb, data_pkt[1], 1))
49             return -1;
50     } else if(was_black || force){
51         // If the lighting WAS black, or if we're on forced update, send the activation packet
52         data_pkt[1][4] = 1;
53         if(!usb send(kb, data_pkt[1], 1))
54

```

```
55         return -1;
56     }
57
58     memcpy(lastlight, newlight, sizeof(lighting));
59     return 0;
60 }
```

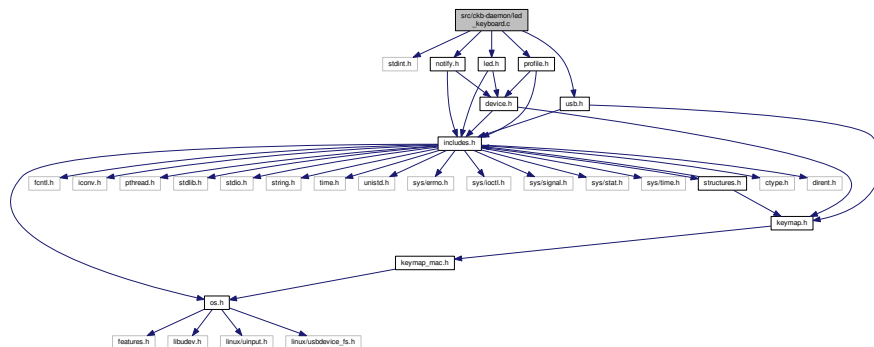
Here is the call graph for this function:



5.26 src/ckb-daemon/led_keyboard.c File Reference

```
#include <stdint.h>
#include "led.h"
#include "notify.h"
#include "profile.h"
#include "usb.h"
```

Include dependency graph for led_keyboard.c:



Macros

- #define BR1(x) (((x) & 0xaa) >> 1) | (((x) & 0x55) << 1))
- #define BR2(x) (((BR1(x) & 0xcc) >> 2) | ((BR1(x) & 0x33) << 2))
- #define BR4(x) (((BR2(x) & 0xf0) >> 4) | ((BR2(x) & 0x0f) << 4))
- #define O0(i) BR4(i),
- #define O1(i) O0(i) O0((i) + 1)
- #define O2(i) O1(i) O1((i) + 2)
- #define O3(i) O2(i) O2((i) + 4)
- #define O4(i) O3(i) O3((i) + 8)
- #define O5(i) O4(i) O4((i) + 16)

Generated on Thu Apr 27 2017 21:48:46 for ckb-next by Doxygen

Generated on Thu Apr 27 2017 21:48:46 for ckb-next by Doxygen

Generated on Thu Apr 27 2017 21:48:46 for ckb-next by Doxygen

Generated on Thu Apr 27 2017 21:48:46 for ckb-next by Doxygen

Generated on Thu Apr 27 2017 21:48:46 for ckb-next by Doxygen

Generated on Thu Apr 27 2017 21:48:46 for ckb-next by Doxygen

Generated on Thu Apr 27 2017 21:48:46 for ckb-next by Doxygen

Generated on Thu Apr 27 2017 21:48:46 for ckb-next by Doxygen

Generated on Thu Apr 27 2017 21:48:46 for ckb-next by Doxygen

Generated on Thu Apr 27 2017 21:48:46 for ckb-next by Doxygen

Generated on Thu Apr 27 2017 21:48:46 for ckb-next by Doxygen

5.26.1 Macro Definition Documentation

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

Definition at line 9 of file `led_keyboard.c`.

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

Definition at line 10 of file `led_keyboard.c`.

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

Definition at line 11 of file `led_keyboard.c`.

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

Definition at line 12 of file `led_keyboard.c`.

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

Definition at line 13 of file `led_keyboard.c`.

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

Definition at line 14 of file `led_keyboard.c`.

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

Definition at line 15 of file `led_keyboard.c`.

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

Definition at line 16 of file `led_keyboard.c`.

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

Definition at line 17 of file `led_keyboard.c`.

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

Definition at line 18 of file `led_keyboard.c`.

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

Definition at line 19 of file `led_keyboard.c`.

5.26.1.12 `#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 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         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                 memcpy(colors[clr] + 120, in_pkt[3] + 4, 24);
238             }
239         } else {
240             uchar data_pkt[5][MSG_SIZE] = {
241                 { 0x0e, 0x14, 0x02, 0x01, 0x01, mode + 1, 0 },
242                 { 0xff, 0x01, 60, 0 },
243                 { 0xff, 0x02, 60, 0 },
244                 { 0xff, 0x03, 60, 0 },
245                 { 0xff, 0x04, 36, 0 },
246             };

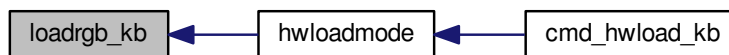
```

```

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 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];

```

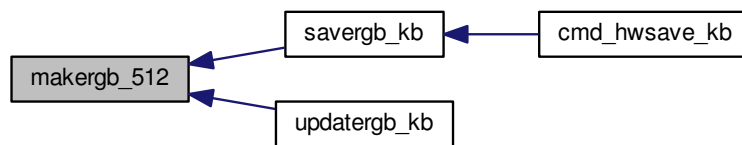


```

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 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`.

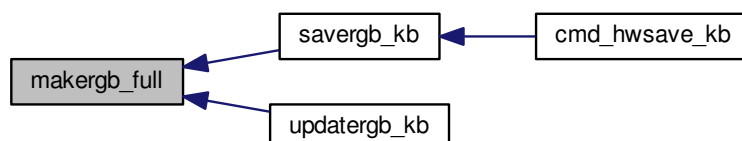
Referenced by `savergb_kb()`, and `updatergb_kb()`.

```

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

Definition at line 24 of file led_keyboard.c.

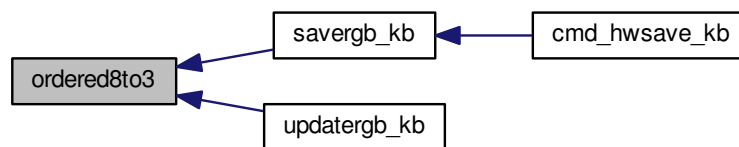
References `bit_reverse_table`.

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

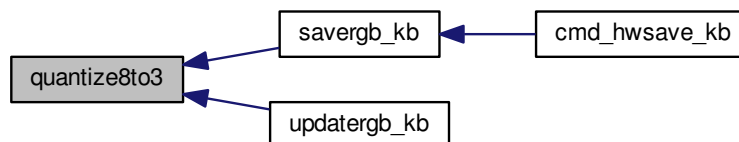
Definition at line 32 of file led_keyboard.c.

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 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);

```

Here is the caller graph for this function:



5.26.2.7 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 usbsend.

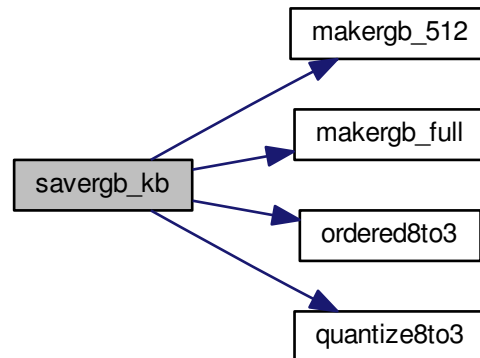
Referenced by cmd_hwsave_kb().

```

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(!usbsend(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(!usbsend(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 :
175         quantize8to3);
176         if(!usbsend(kb, data_pkt[0], 5))
177             return -1;
178     }
179     return 0;

```

Here is the call graph for this function:



Here is the caller graph for this function:



5.26.2.8 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 `usb send`.

```

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)

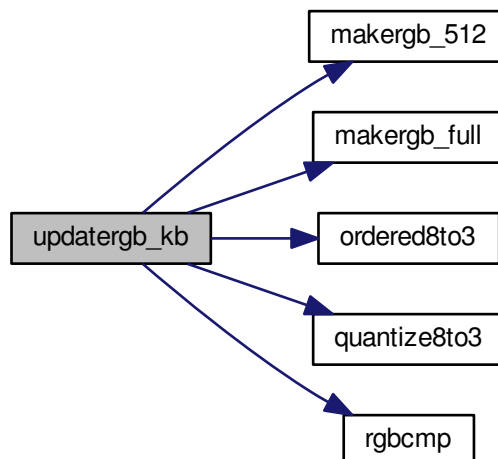
```

```

96         data_pkt[0][4]=1;    // turn on
97         if(!usb_send(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(!usb_send(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(!usb_send(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:



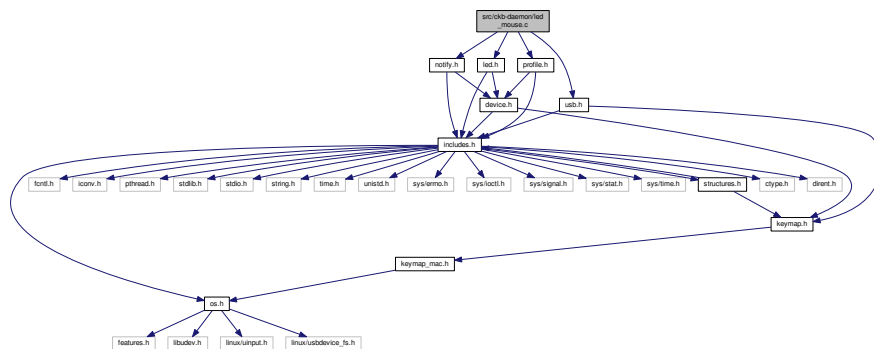
5.26.3 Variable Documentation

Referenced by ordered8to3().

5.27 src/ckb-daemon/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 `rbgcmp` (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 static int isblack (const usbdevice * kb, const lighting * light) [static]

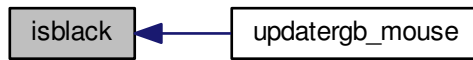
Definition at line 13 of file led_mouse.c.

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 `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 `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 `updaterrgb_mouse()`.


```

7
8     return memcmp(lhs->r + LED_MOUSE, rhs->r + LED_MOUSE,
    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);
9 }

```

Here is the caller graph for this function:



5.27.1.4 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 int updatergb_mouse (usbdevice * kb, int force)

Definition at line 20 of file led_mouse.c.

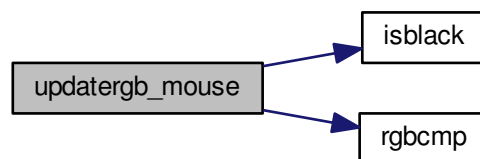
References `usbdevice::active`, `lighting::b`, `usbprofile::currentmode`, `lighting::forceupdate`, `lighting::g`, `isblack()`, `usbprofile::lastlight`, `LED_MOUSE`, `usbmode::light`, `MSG_SIZE`, `N_MOUSE_ZONES`, `usbdevice::profile`, `lighting::r`, `rgbcmp()`, and `usbsend`.

```

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:



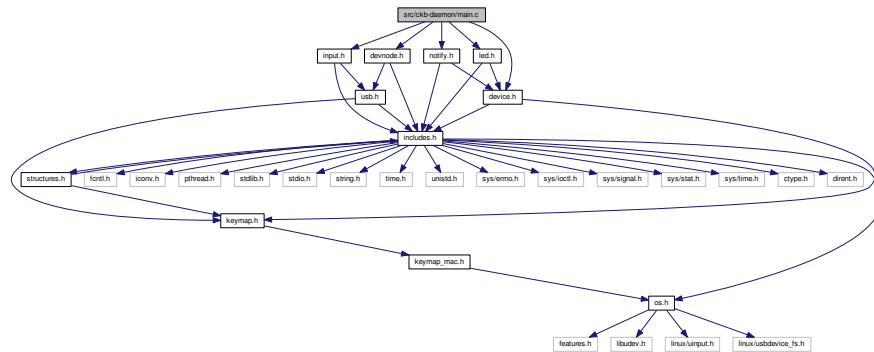
5.28 src/ckb-daemon/main.c File Reference

```

#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 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         *ldst++ = s;
80         if(dst == ldst){
81             dst--;

```

```

82         break;
83     }
84 }
85 *dst = 0;
86 }

```

5.28.1.2 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"
114                     "    --hwload=<always|try|never>\n"
115                     "        --hwload=always will force loading of stored hardware profiles on
116                     compatible devices. May result in long start up times.\n"
117                     "        --hwload=try will try to load the profiles, but give up if not immediately
118                     successful (default).\n"
119                     "        --hwload=never will ignore hardware profiles completely.\n"
120                     "    --nonotify\n"
121                     "        Disables key monitoring/notifications.\n"
122                     "    Note that this makes reactive lighting impossible.\n"
123                     "    --nobind\n"
124                     "        Disables all key rebinding, macros, and notifications. Implies --nonotify.
125                     \n"
126 #ifdef OS_MAC
127                     "    --nomouseaccel\n"
128                     "        Disables mouse acceleration, even if the system preferences enable it.\n"
129 #endif
130                     "    --nonroot\n"
131                     "        Allows running ckb-daemon as a non root user.\n"
132                     "        This will almost certainly not work. Use only if you know what you're
133                     doing.\n"
134                     "\n", devpath);
135                 exit(0);
136             }
137         }
138
139         // Check PID, quit if already running
140         char pidpath[strlen(devpath) + 6];
141         snprintf(pidpath, sizeof(pidpath), "%s0/pid", devpath);
142         FILE* pidfile = fopen(pidpath, "r");
143         if(pidfile){
144             pid_t pid;
145             fscanf(pidfile, "%d", &pid);
146             fclose(pidfile);
147             if(pid > 0){
148                 // kill -s 0 checks if the PID is active but doesn't send a signal
149                 if(!kill(pid, 0)){
150                     ckb_fatal_nofile("ckb-daemon is already running (PID %d). Try 'killall
151 ckb-daemon'.\n", pid);
152                     ckb_fatal_nofile("(If you're certain the process is dead, delete %s and try

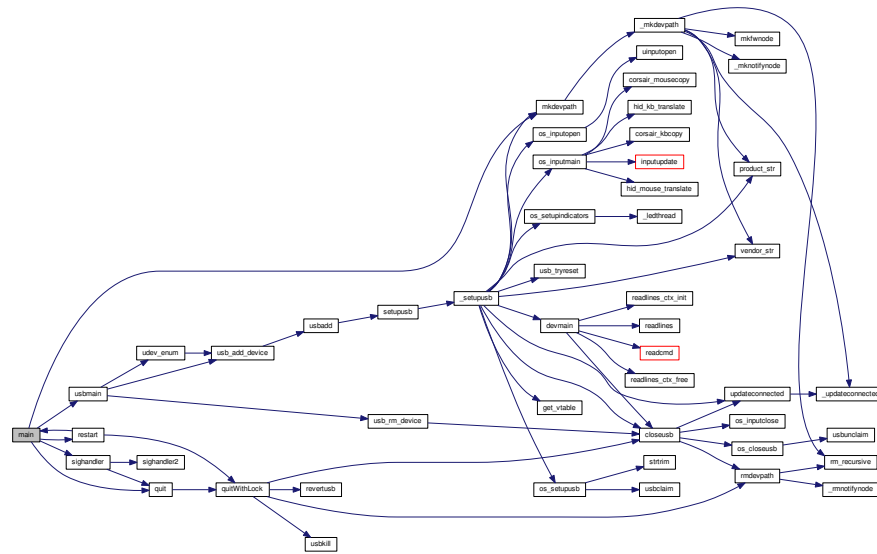
```

```

        again)\n", pidpath);
146         return 0;
147     }
148 }
149 }
150
151 // Read parameters
152 int forceroot = 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         forceroot = 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
191 }
192
193 // Check UID
194 if(getuid() != 0){
195     if(forceroot){
196         ckb_fatal_nofile("ckb-daemon must be run as root. Try 'sudo %s'\n", argv[0]);
197         exit(0);
198     } else
199         ckb_warn_nofile("Warning: not running as root, allowing anyway per command-line
parameter...\n");
200 }
201
202 // Make root keyboard
203 umask(0);
204 memset(keyboard, 0, sizeof(keyboard));
205 if(!mkdevpath(keyboard))
206     ckb_info("Root controller ready at %s0\n", devpath);
207
208 // Set signals
209 sigset_t signals;
210 sigfillset(&signals);
211 sigdelset(&signals, SIGTERM);
212 sigdelset(&signals, SIGINT);
213 sigdelset(&signals, SIGQUIT);
214 sigdelset(&signals, SIGUSR1);
215 // Set up signal handlers for quitting the service.
216 sigprocmask(SIG_SETMASK, &signals, 0);
217 signal(SIGTERM, sighandler);
218 signal(SIGINT, sighandler);
219 signal(SIGQUIT, sighandler);
220 signal(SIGUSR1, (void (*)(void))restart);
221
222 // Start the USB system
223 int result = usbmain();
224 quit();
225 return result;
226 }

```

Here is the call graph for this function:



Here is the caller graph for this function:



5.28.1.3 static void quit () [static]

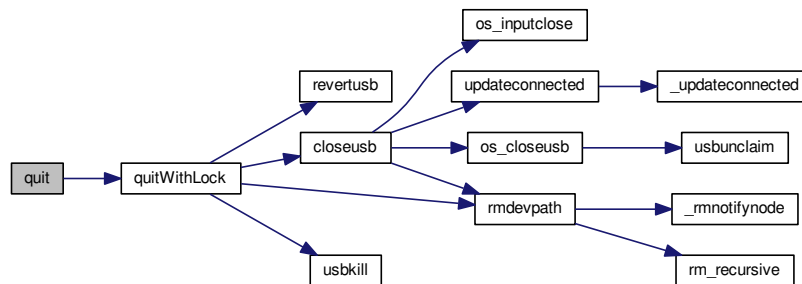
Definition at line 30 of file main.c.

References `quitWithLock()`.

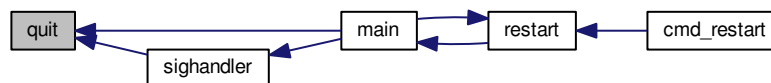
Referenced by `main()`, and `sighandler()`.

```
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 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.

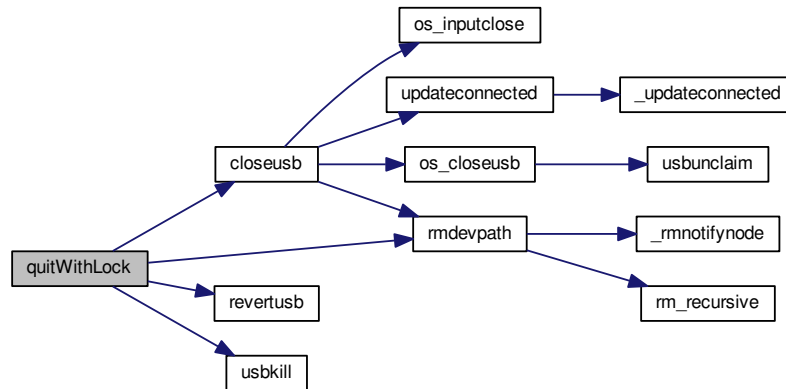
References ckb_info, closeusb(), DEV_MAX, devmutex, IS_CONNECTED, keyboard, reset_stop, revertusb(), rmdevpath(), and usbkill().

Referenced by quit(), and restart().

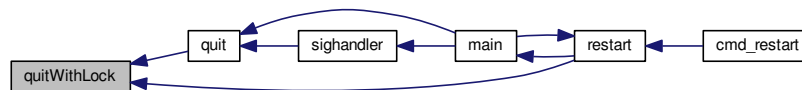
```

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:



Here is the caller graph for this function:



5.28.1.5 int restart ()

Definition at line 228 of file main.c.

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

Referenced by `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     }

```



```

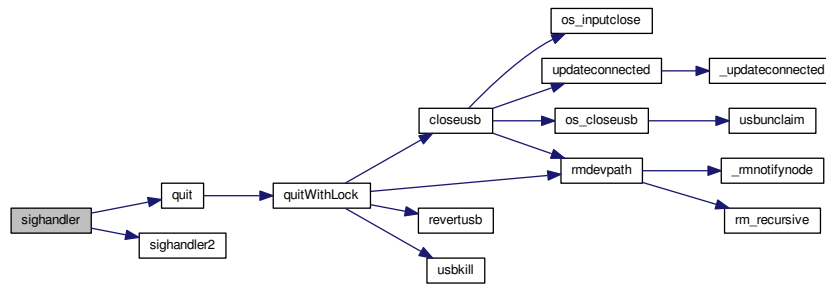
graph LR
    restart_shaded[restart]
    main[main]
    restart_white[restart]
    cmd_restart[cmd_restart]

    cmd_restart --> restart_shaded
    restart_shaded --> main
    main --> restart_white
    restart_white --> cmd_restart
  
```

Definition at line 62 of file main.c.
References quit(), and sighandler2().
Referenced by main().

Generated on Thu Apr 27 2017 21:48:46 for ckb-next by Doxygen

Here is the call graph for this function:



Here is the caller graph for this function:



5.28.1.7 void sighandler2 (int type)

Definition at line 57 of file main.c.

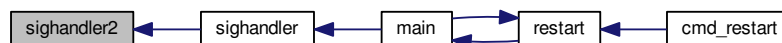
Referenced by sighandler().

```

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 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 int features_mask

features_mask Mask of features to exclude from all devices

That bit mask is 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 int hwload_mode

Definition at line 7 of file device.c.

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

5.28.2.3 int main_ac [static]

Definition at line 7 of file main.c.

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

5.28.2.4 char** main_av [static]

Definition at line 8 of file main.c.

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

5.28.2.5 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\(\)](#), [_usbseend\(\)](#), [quitWithLock\(\)](#), and [usb_tryreset\(\)](#).

5.29 src/ckb-daemon/notify.c File Reference

```
#include "device.h"
#include "devnode.h"
#include "dpi.h"
#include "led.h"
#include "notify.h"
#include "profile.h"
```


Definition at line 73 of file notify.c.

5.29.2 Function Documentation

5.29.2.1 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, ":hwrrgb")){
103        // Get the current hardware RGB settings
104        HW_STANDARD;
105        char* rgb = printrgb(kb->hw->light + index, kb);
106        nprintf(kb, nnumber, mode, "hwrrgb %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);

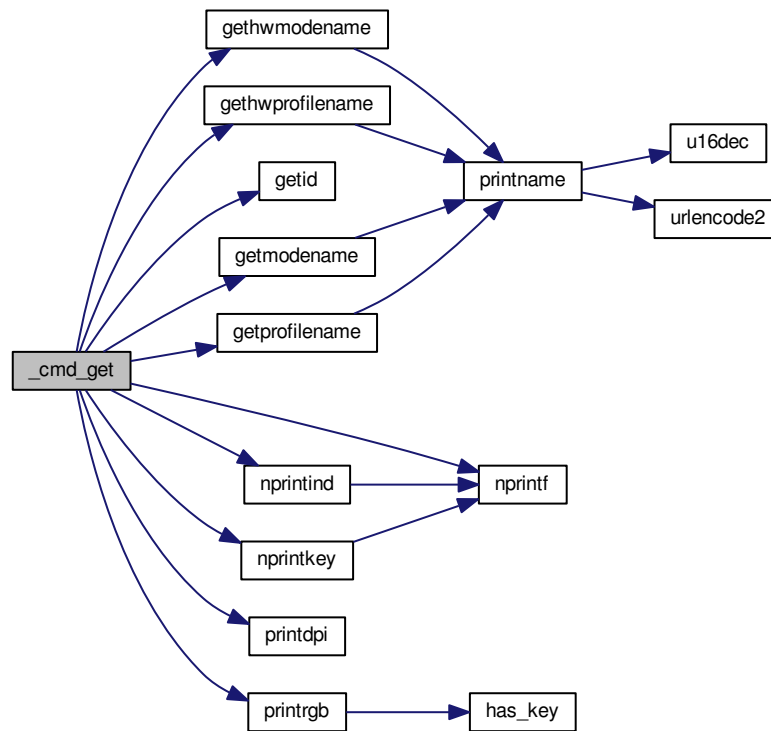
```

```

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, ":dpisel")){
195     // Get the currently-selected DPI
196     nprintf(kb, nnumber, mode, "dpisel %d\n", mode->dpi.current);
197 } else if(!strcmp(setting, ":hwdpisel")){
198     // Get the currently-selected hardware DPI
199     HW_STANDARD;
200     nprintf(kb, nnumber, mode, "hwdpisel %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 void cmd_get (usbdevice * kb, usbmode * mode, int nnumber, int dummy, const char * setting)

Definition at line 215 of file notify.c.

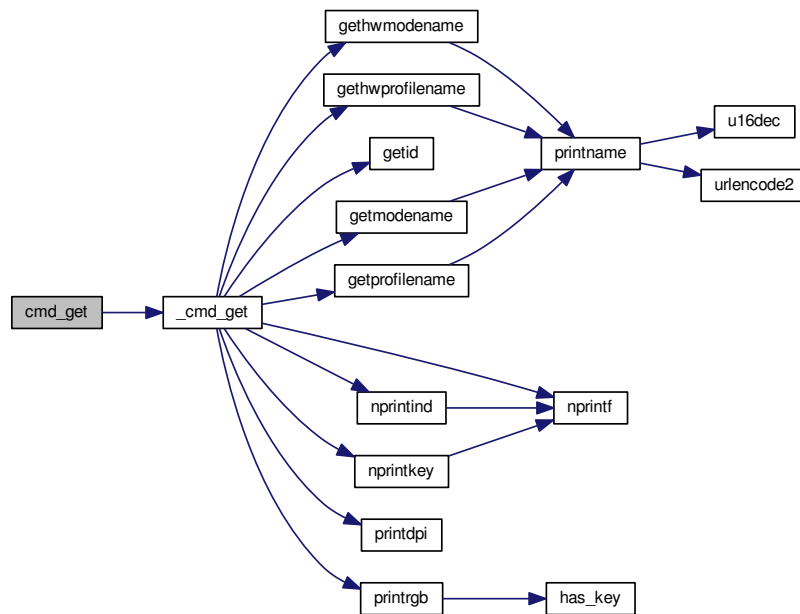
References `_cmd_get()`, and `imutex`.

```

215                                     {
216     pthread_mutex_lock(imutex(kb));
217     _cmd_get(kb, mode, nnumber, setting);
218     pthread_mutex_unlock(imutex(kb));
219 }

```

Here is the call graph for this function:



5.29.2.3 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(imutex(kb));
65     if (!strcmp(toggle, "on") || *toggle == 0)
66         SET_KEYBIT(mode->notify[nnumber], keyindex);
67     else if (!strcmp(toggle, "off"))
68         CLEAR_KEYBIT(mode->notify[nnumber], keyindex);
69     pthread_mutex_unlock(imutex(kb));
70 }

```

5.29.2.4 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 }

```

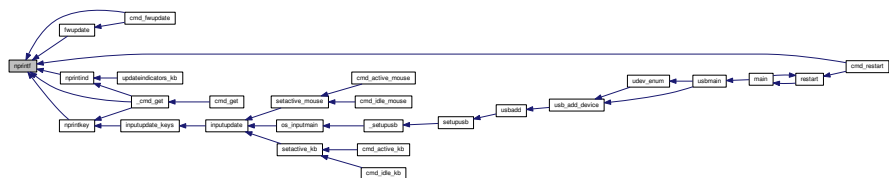

[illegible]

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()`.

Here is the caller graph for this function:



5.29.2.6 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().

```

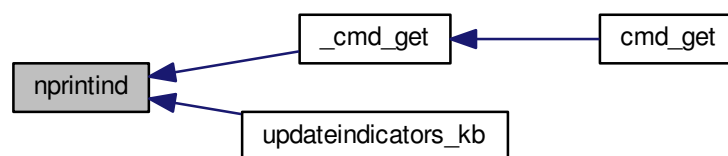
43                                     {
44     const char* name = 0;
45     switch(led) {
46     case I_NUM:
47         name = "num";
48         break;
49     case I_CAPS:
50         name = "caps";
51         break;
52     case I_SCROLL:
53         name = "scroll";
54         break;
55     default:
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:



5.29.2.7 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)

```

```

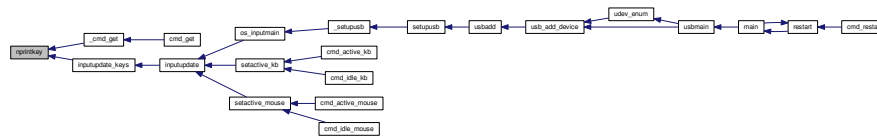
38     nprintf(kb, nnumber, 0, "key %c%s\n", down ? '+' : '-', map->name);
39     else
40     nprintf(kb, nnumber, 0, "key %c#%d\n", down ? '+' : '-', keyindex);
41 }

```

Here is the call graph for this function:



Here is the caller graph for this function:



5.29.2.8 int restart ()

Definition at line 228 of file main.c.

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

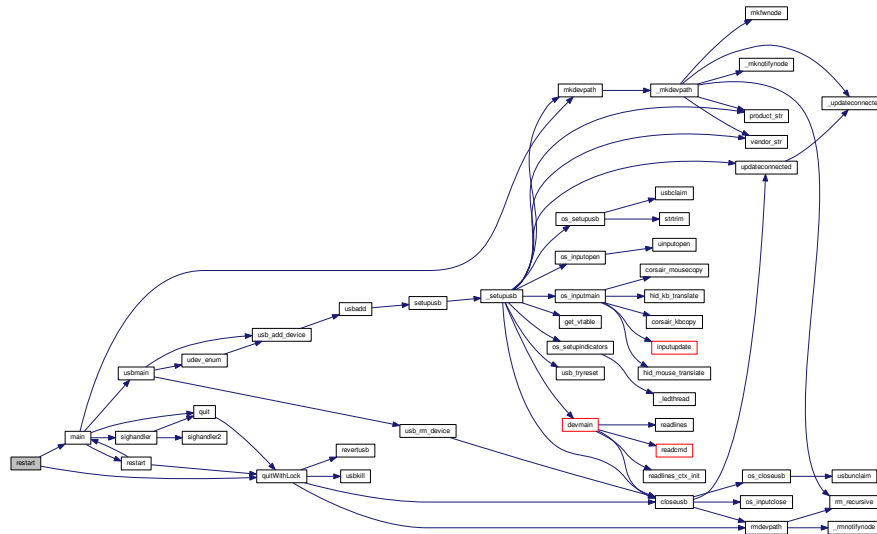
Referenced by `cmd_restart()`, and `main()`.

```

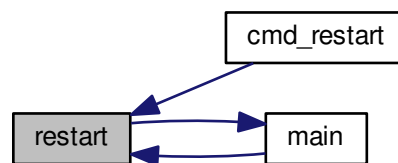
228     {
229     ckb_err("restart called, running quit without mutex-lock.\n");
230     quitWithLock(0);
231     return main(main_ac, main_av);
232 }

```

Here is the call graph for this function:

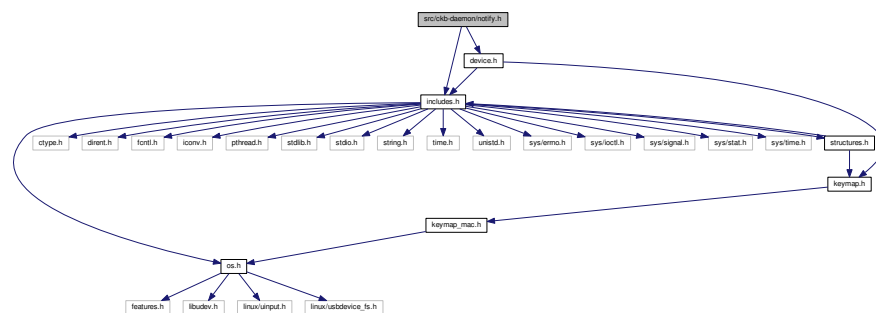


Here is the caller graph for this function:



5.30 src/ckb-daemon/notify.h File Reference

```
#include "includes.h"
#include "device.h"
Include dependency graph for notify.h:
```



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



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

5.30.1 Function Documentation

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

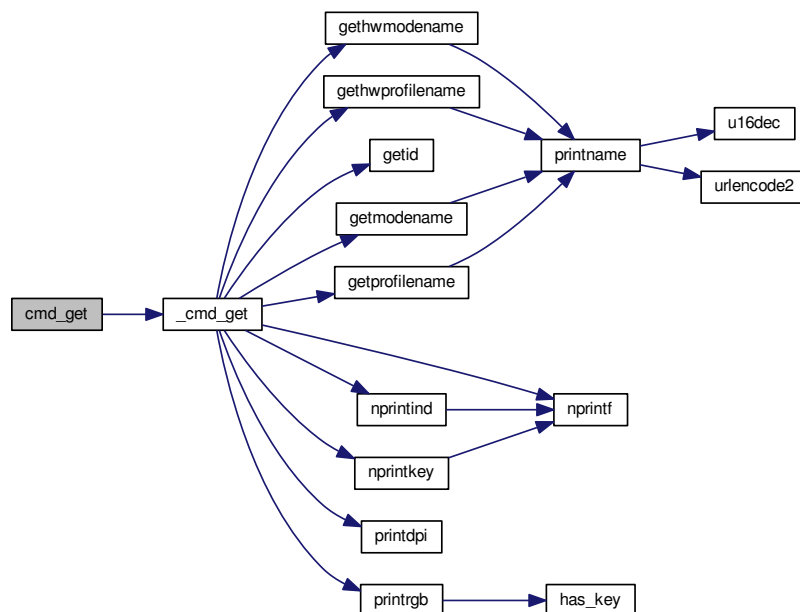
Definition at line 215 of file notify.c.

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

```

215
216     pthread_mutex_lock(&imutex(kb));
217     _cmd_get(kb, mode, nnumber, setting);
218     pthread_mutex_unlock(&imutex(kb));
219 }
```

Here is the call graph for this function:

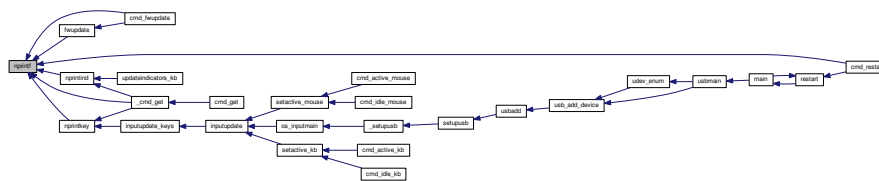



```

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 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()`.

```

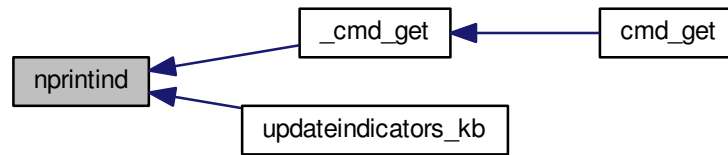
43
44     const char* name = 0;
45     switch(led){
46     case I_NUM:
47         name = "num";
48         break;
49     case I_CAPS:
50         name = "caps";
51         break;
52     case I_SCROLL:
53         name = "scroll";
54         break;
55     default:
56         return;
57     }
58     nprintf(kb, 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 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().

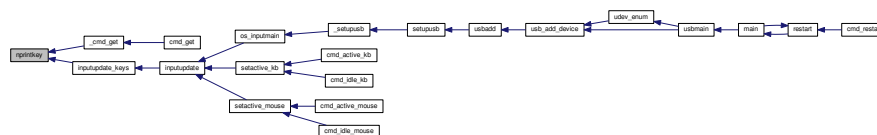
```

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:



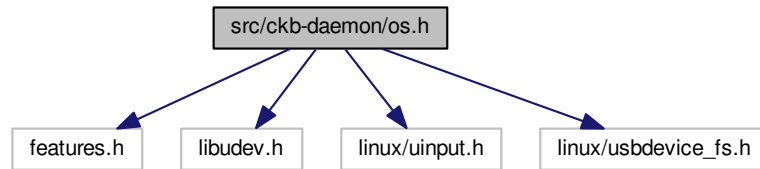
5.31 src/ckb-daemon/os.h File Reference

```
#include <features.h>
```



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

Include dependency graph for os.h:



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



Macros

- `#define _DEFAULT_SOURCE`
- `#define _GNU_SOURCE`
- `#define UINPUT_VERSION 2`
- `#define euid_guard_start`
- `#define euid_guard_stop`

5.31.1 Macro Definition Documentation

5.31.1.1 `#define _DEFAULT_SOURCE`

Definition at line 22 of file os.h.

5.31.1.2 `#define _GNU_SOURCE`

Definition at line 26 of file os.h.

5.31.1.3 `#define euid_guard_start`

Definition at line 40 of file os.h.

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

5.31.1.4 `#define euid_guard_stop`

Definition at line 41 of file os.h.

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

Variables

- static iconv_t `utf8to16` = 0
- static iconv_t `utf16to8` = 0

5.32.1 Function Documentation

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

Definition at line 210 of file profile.c.

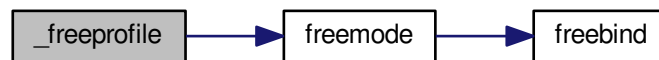
References `freemode()`, `usbprofile::mode`, `MODE_COUNT`, and `usbdevice::profile`.

Referenced by `cmd_eraseprofile()`, and `freeprofile()`.

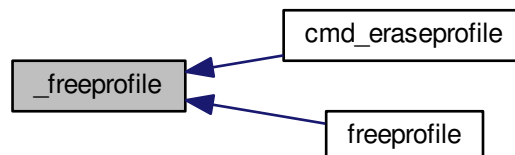
```

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 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`.

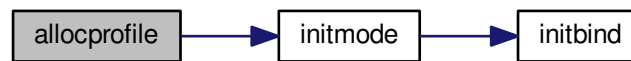
Referenced by `cmd_eraseprofile()`.

```

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

Definition at line 203 of file `profile.c`.

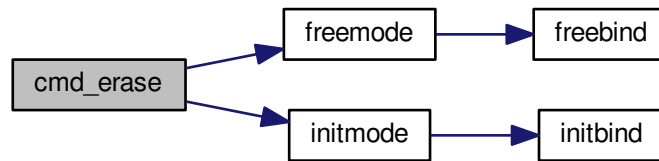
References `freemode()`, `imutex`, and `initmode()`.

```

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

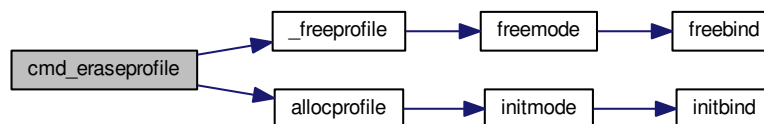
Definition at line 221 of file profile.c.

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

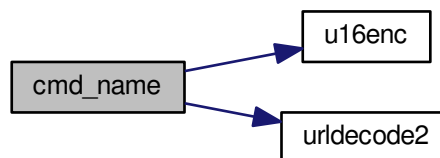
Definition at line 117 of file profile.c.

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

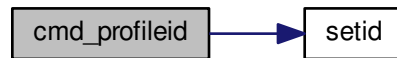
Definition at line 167 of file profile.c.

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

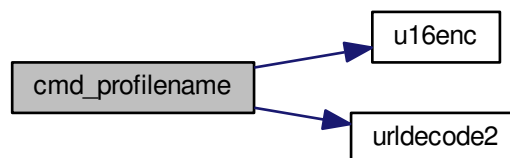
Definition at line 124 of file profile.c.

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 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().

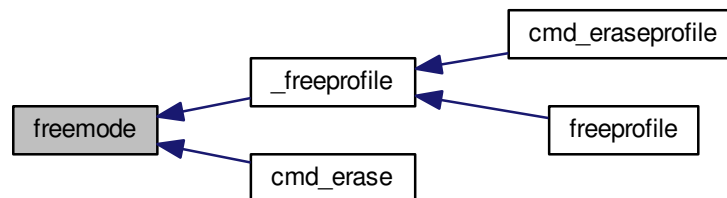
```

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 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 char* gethwmodename (hwprofile * profile, int index)

Definition at line 152 of file profile.c.

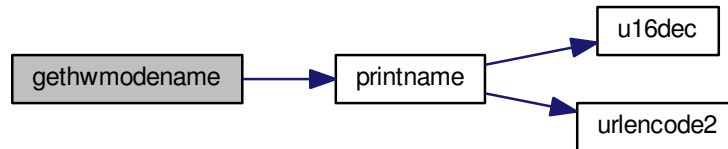
References MD_NAME_LEN, hwprofile::name, and printname().

Referenced by _cmd_get().

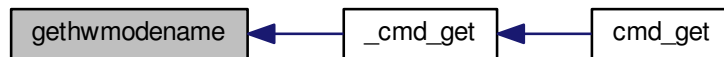
```

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 char* gethwprofilename (hwprofile * profile)

Definition at line 156 of file profile.c.

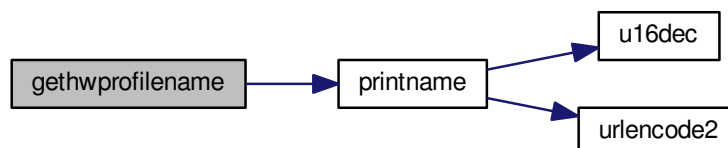
References MD_NAME_LEN, hwprofile::name, and printname().

Referenced by _cmd_get().

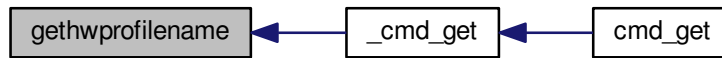
```

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 `char* getid (usbld * id)`

Definition at line 79 of file `profile.c`.

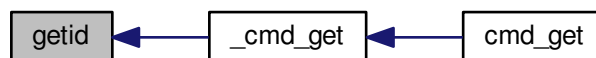
References `usbld::guid`.

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 `char* getmodename (usbmode * mode)`

Definition at line 144 of file `profile.c`.

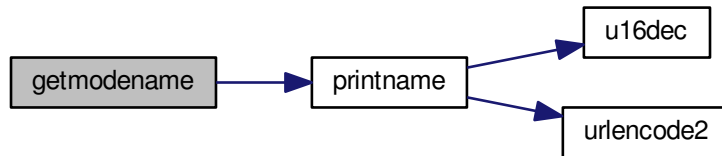
References `MD_NAME_LEN`, `usbmode::name`, and `printname()`.

Referenced by `_cmd_get()`.

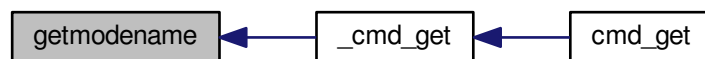
```

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 char* getprofilename (usbprofile * profile)

Definition at line 148 of file profile.c.

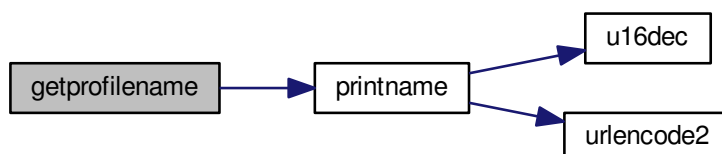
References `usbprofile::name`, `PR_NAME_LEN`, and `printname()`.

Referenced by `_cmd_get()`.

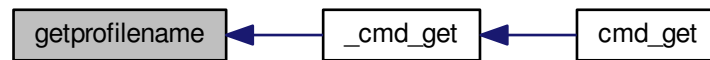
```

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 void hwtonative (usbprofile * profile, hwprofile * hw, int modecount)

Definition at line 235 of file profile.c.

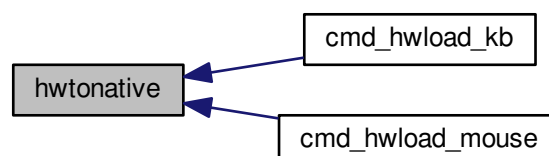
References `usbmode::dpi`, `hwprofile::dpi`, `dpiset::forceupdate`, `lighting::forceupdate`, `usbmode::id`, `usbprofile::id`, `hwprofile::id`, `usbprofile::lastdpi`, `usbprofile::lastlight`, `usbmode::light`, `hwprofile::light`, `MD_NAME_LEN`, `usbprofile::mode`, `usbmode::name`, `usbprofile::name`, `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 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()`.

```

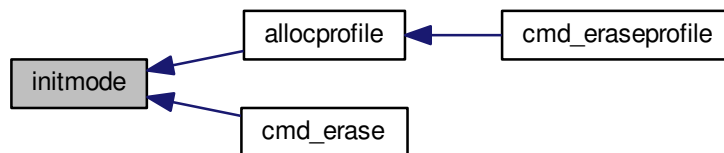
175
176     memset(mode, 0, sizeof(*mode));
177     mode->light.forceupdate = 1;
178     mode->dpi.forceupdate = 1;
179     initbind(&mode->bind);
180 }

```

Here is the call graph for this function:



Here is the caller graph for this function:



5.32.1.18 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 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

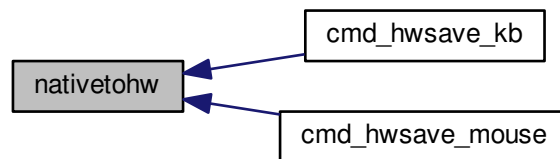
```

```

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 char* printname (ushort * name, int length)

Definition at line 132 of file profile.c.

References `u16dec()`, and `urlencode2()`.

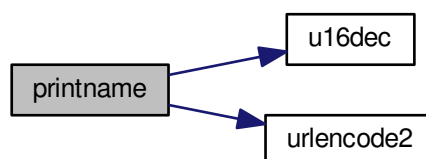
Referenced by `gethwmodename()`, `gethwprofilename()`, `getmodename()`, and `getprofilename()`.

```

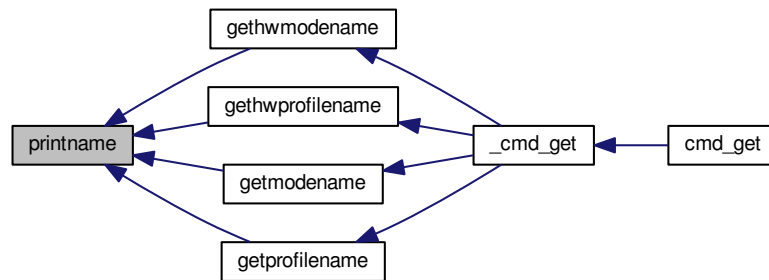
132     {
133         // Convert the name to UTF-8
134         char* buffer = calloc(1, length * 4 - 3);
135         size_t srclen = length, dstlen = length * 4 - 4;
136         u16dec(name, buffer, &srclen, &dstlen);
137         // URL-encode it
138         char* buffer2 = malloc(strlen(buffer) * 3 + 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 `int setid (usbid * id, const char * guid)`

Definition at line 64 of file `profile.c`.

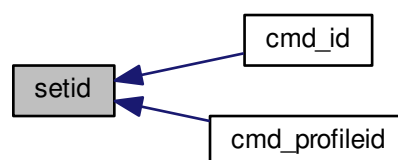
References `usbid::guid`.

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 `void u16dec (ushort * in, char * out, size_t * srclen, size_t * dstlen)`

Definition at line 105 of file `profile.c`.

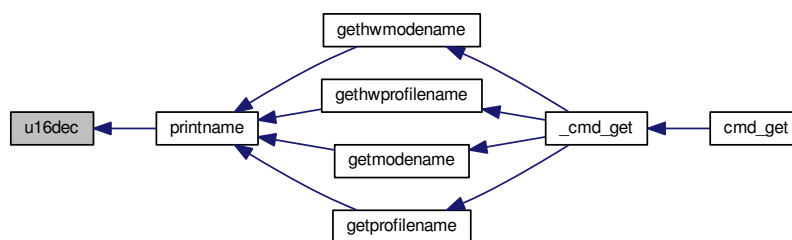
References utf16to8.

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 void u16enc (char * in, ushort * out, size_t * srclen, size_t * dstlen)

Definition at line 97 of file profile.c.

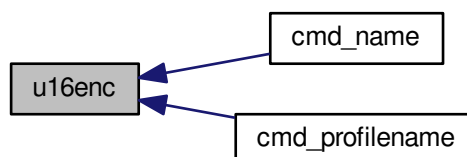
References utf8to16.

Referenced by cmd_name(), and cmd_profilename().

```

97                                     {
98     if(!utf8to16)
99         utf8to16 = iconv_open("UTF-16LE", "UTF-8");
100     memset(out, 0, *dstlen * 2);
101     *dstlen = *dstlen * 2 - 2;
102     iconv(utf8to16, &in, srclen, (char**)&out, dstlen);
103 }
```

Here is the caller graph for this function:



5.32.1.24 void urldecode2 (char * dst, const char * src)

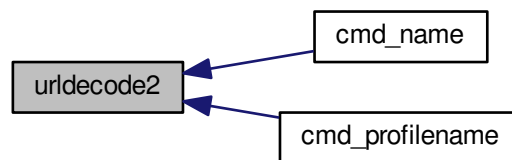
Definition at line 8 of file profile.c.

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 void urlencode2 (char * dst, const char * src)

Definition at line 37 of file profile.c.

Referenced by printname().

```

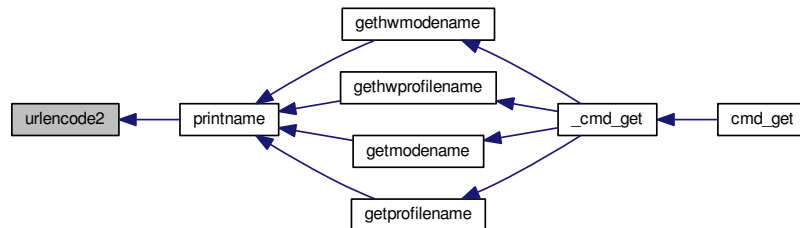
37                                     {
38    char s;
39    while((s = *src++){
40        if(s <= ',' || s == '/' ||
41            (s >= ':' && s <= '@') ||
42            s == '[' || s == ']' ||
43            s >= 0x7F){
44            char a = s >> 4, b = s & 0xF;
45            if(a >= 10)
46                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 `iconv_t utf16to8 = 0` [static]

Definition at line 95 of file `profile.c`.

Referenced by `u16dec()`.

5.32.2.2 `iconv_t utf8to16 = 0` [static]

Definition at line 95 of file `profile.c`.

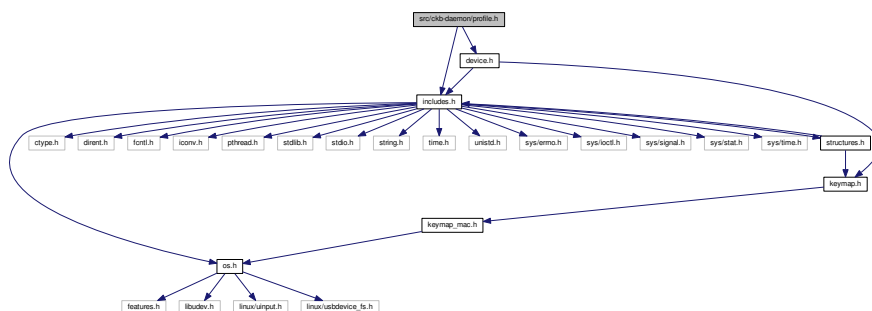
Referenced by `u16enc()`.

5.33 `src/ckb-daemon/profile.h` File Reference

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

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

Include dependency graph for `profile.h`:



```

graph TD
    Root[ECG-Dataset-15] --> C1[ecg-dataset-coronary-1]
    Root --> C2[ecg-dataset-coronary-2]
    Root --> C3[ecg-dataset-coronary-3]
    Root --> C4[ecg-dataset-coronary-4]
    Root --> C5[ecg-dataset-coronary-5]
    Root --> C6[ecg-dataset-coronary-6]
    Root --> C7[ecg-dataset-coronary-7]
    Root --> C8[ecg-dataset-coronary-8]
    Root --> C9[ecg-dataset-coronary-9]
    Root --> C10[ecg-dataset-coronary-10]
    Root --> C11[ecg-dataset-coronary-11]
    Root --> C12[ecg-dataset-coronary-12]
    Root --> C13[ecg-dataset-coronary-13]
    Root --> C14[ecg-dataset-coronary-14]
    Root --> C15[ecg-dataset-coronary-15]
  
```

- `#define hwloadprofile(kb, apply) (kb)->vtable->hwload(kb, 0, 0, apply, 0)`

- 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)

```
5.33.1.1 #define hwloadprofile( kb, apply ) (kb)->vtable->hwload(kb, 0, 0, apply, 0)
```

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

5.33.2.1 void allocprofile (usbdevice * kb)

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

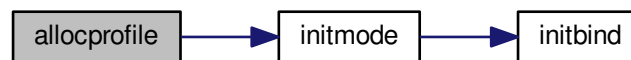
Referenced by cmd eraseprofile().

```

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:



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

Definition at line 203 of file profile.c.

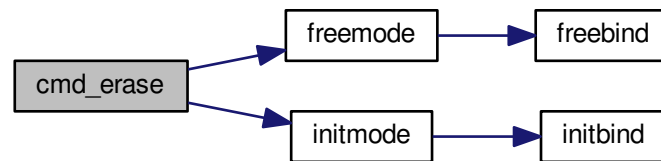
References `freemode()`, `imutex`, and `initmode()`.

```

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

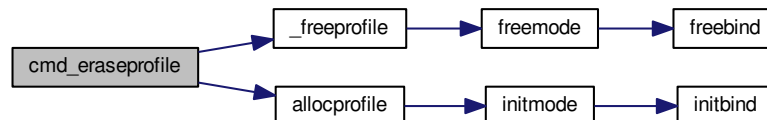
Definition at line 221 of file profile.c.

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 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`.

```

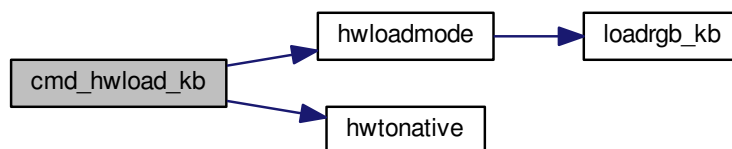
16
17     DELAY_LONG(kb);
18     hwprofile* hw = calloc(1, sizeof(hwprofile));
19     // Ask for profile and mode IDs
20     uchar data_pkt[2][MSG_SIZE] = {
21         { 0x0e, 0x15, 0x01, 0 },
22         { 0x0e, 0x16, 0x01, 0 }
23     };
24     uchar in_pkt[MSG_SIZE];
25     int modes = (IS_K95(kb) ? 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 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        }

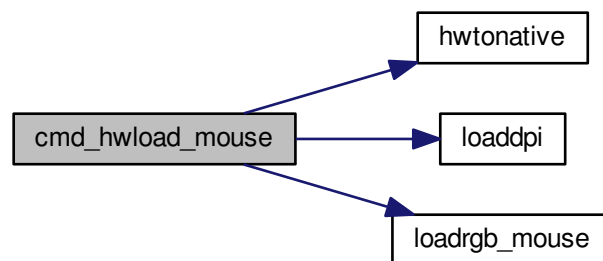
```

```

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 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     }

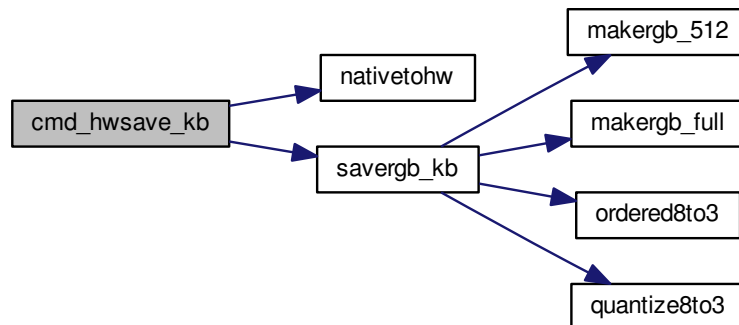
```

```

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:



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

Definition at line 50 of file `profile_mouse.c`.

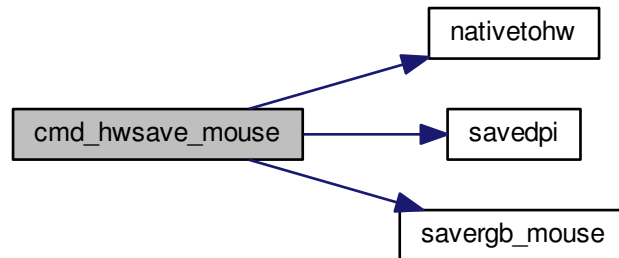
References `DELAY_LONG`, `hwprofile::dpi`, `usbdevice::hw`, `hwprofile::id`, `hwprofile::light`, `MD_NAME_LEN`, `MSG_SIZE`, `hwprofile::name`, `nativetohw()`, `usbdevice::profile`, `savedpi()`, `savergb_mouse()`, and `usb send`.

```

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 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:



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

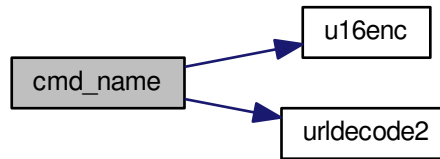
Definition at line 117 of file profile.c.

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

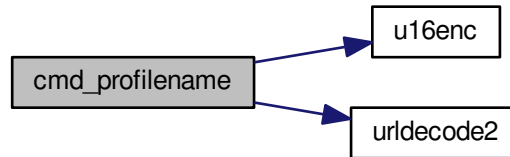
Definition at line 124 of file profile.c.

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 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 char* gethwmodename (hwprofile * profile, int index)

Definition at line 152 of file profile.c.

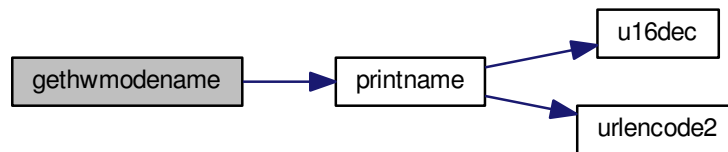
References `MD_NAME_LEN`, `hwprofile::name`, and `printname()`.

Referenced by `_cmd_get()`.

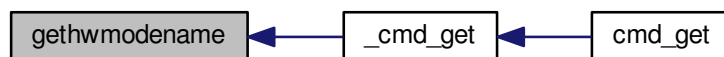
```

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 `char* gethwprofilename (hwprofile * profile)`

Definition at line 156 of file `profile.c`.

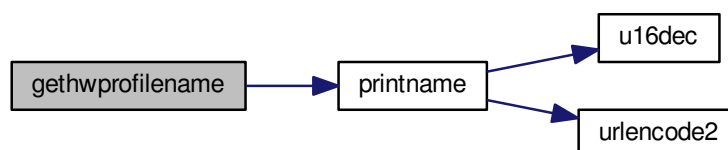
References `MD_NAME_LEN`, `hwprofile::name`, and `printname()`.

Referenced by `_cmd_get()`.

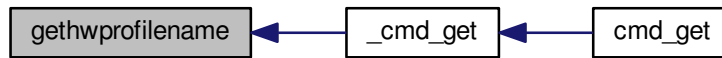
```

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 `char* getid (usbld * id)`

Definition at line 79 of file `profile.c`.

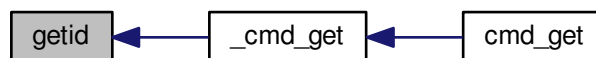
References `usbld::guid`.

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",
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 `char* getmodename (usbmode * mode)`

Definition at line 144 of file `profile.c`.

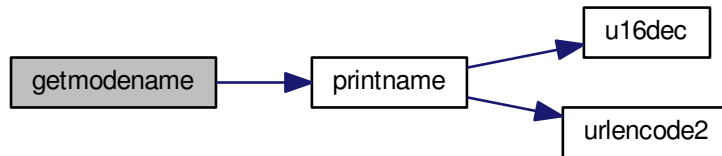
References `MD_NAME_LEN`, `usbmode::name`, and `printname()`.

Referenced by `_cmd_get()`.

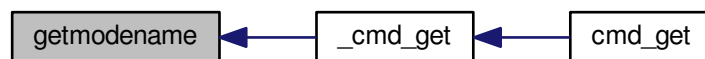
```

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 `char* getprofilename (usbprofile * profile)`

Definition at line 148 of file `profile.c`.

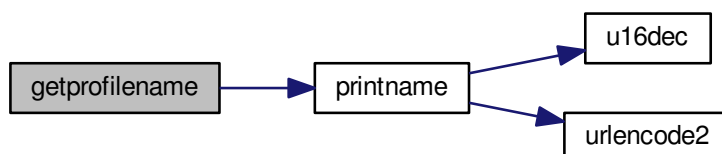
References `usbprofile::name`, `PR_NAME_LEN`, and `printname()`.

Referenced by `_cmd_get()`.

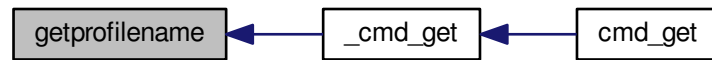
```

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 void hwtonative (usbprofile * profile, hwprofile * hw, int modecount)

Definition at line 235 of file profile.c.

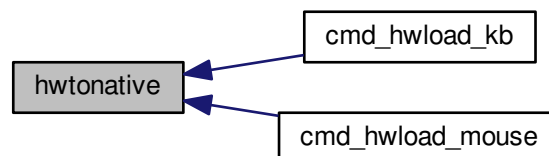
References usbmode::dpi, hwprofile::dpi, dpiset::forceupdate, lighting::forceupdate, usbmode::id, usbprofile::id, hwprofile::id, usbprofile::lastdpi, usbprofile::lastlight, usbmode::light, hwprofile::light, MD_NAME_LEN, usbprofile::mode, usbmode::name, usbprofile::name, hwprofile::name, and PR_NAME_LEN.

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.
250     forceupdate = 1;
251 }
```

Here is the caller graph for this function:



5.33.2.19 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 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.

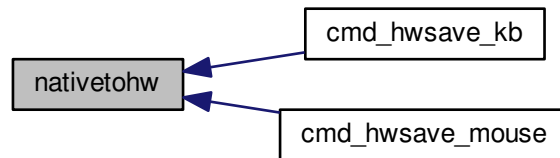
Referenced by cmd_hwsave_kb(), and cmd_hwsave_mouse().

```

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:



5.33.2.21 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,
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     }

```

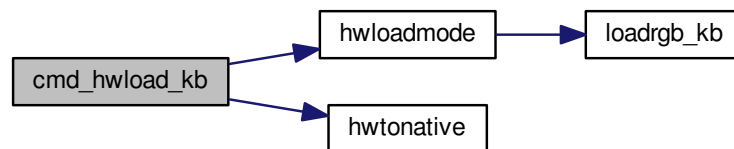


```

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 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 usbsend.

```

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;

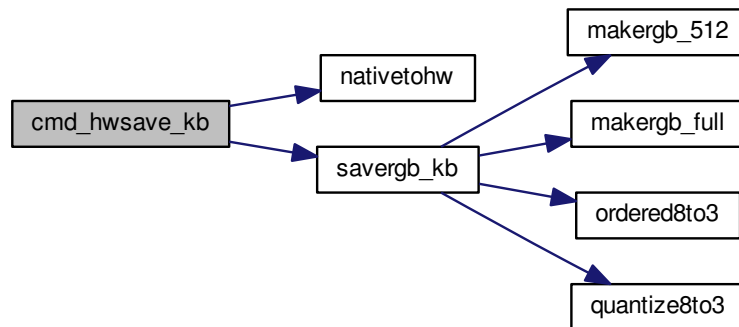
```

```

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 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:



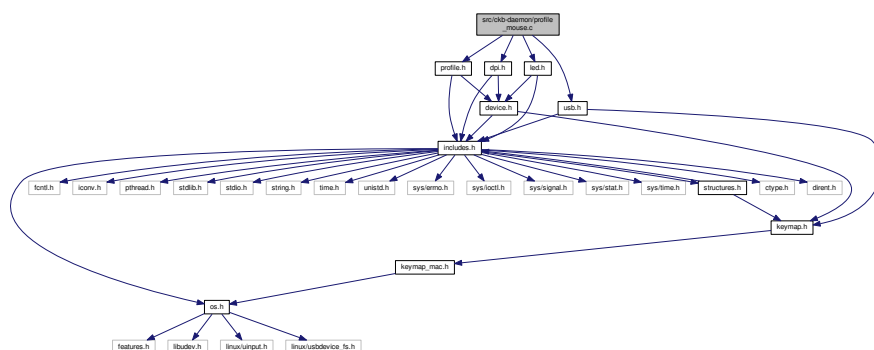
Here is the caller graph for this function:



5.35 src/ckb-daemon/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 int cmd_hwload_mouse (usbdevice * kb, usbmode * dummy1, int dummy2, int apply, const char * dummy3)

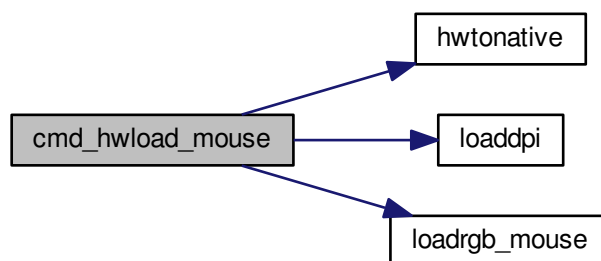
Definition at line 6 of file profile_mouse.c.

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

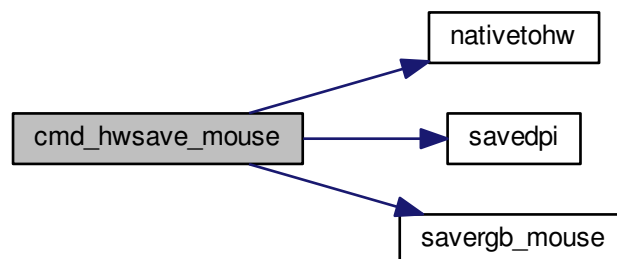
Definition at line 50 of file `profile_mouse.c`.

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(!usbsend(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(usbid));
71         if(!usbsend(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:

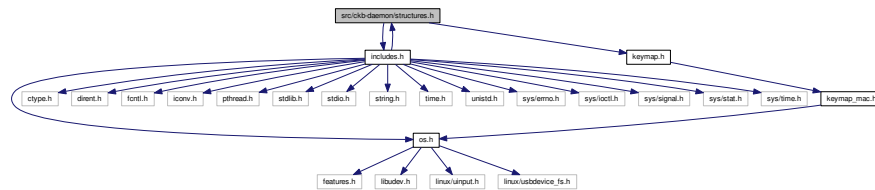


5.36 src/ckb-daemon/structures.h File Reference

```

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

Include dependency graph for structures.h:



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



Data Structures

- struct [usbid](#)
- struct [macroaction](#)
- struct [keymacro](#)
- struct [binding](#)
- struct [dpiset](#)
- struct [lighting](#)
- struct [usbmode](#)
- struct [usbprofile](#)
- struct [hwprofile](#)
- struct [usbinput](#)
- struct [usbdevice](#)

Macros

- #define [SET_KEYBIT](#)(array, index) do { (array)[(index) / 8] |= 1 << ((index) % 8); } while(0)
- #define [CLEAR_KEYBIT](#)(array, index) do { (array)[(index) / 8] &= ~(1 << ((index) % 8)); } while(0)
- #define [I_NUM](#) 1
- #define [I_CAPS](#) 2
- #define [I_SCROLL](#) 4
- #define [OUTFIFO_MAX](#) 10
- #define [MACRO_MAX](#) 1024
- #define [DPI_COUNT](#) 6
- #define [LIFT_MIN](#) 1
- #define [LIFT_MAX](#) 5
- #define [MD_NAME_LEN](#) 16
- #define [PR_NAME_LEN](#) 16
- #define [MODE_COUNT](#) 6
- #define [HWMODE_K70](#) 1
- #define [HWMODE_K95](#) 3
- #define [HWMODE_MAX](#) 3
- #define [FEAT_RGB](#) 0x001
- #define [FEAT_MONOCHROME](#) 0x002
- #define [FEAT_POLLRATE](#) 0x004

- `#define FEAT_ADJRATE 0x008`
- `#define FEAT_BIND 0x010`
- `#define FEAT_NOTIFY 0x020`
- `#define FEAT_FWVERSION 0x040`
- `#define FEAT_FWUPDATE 0x080`
- `#define FEAT_HWLOAD 0x100`
- `#define FEAT_ANSI 0x200`
- `#define FEAT_ISO 0x400`
- `#define FEAT_MOUSEACCEL 0x800`
- `#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

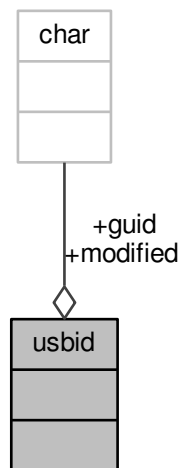
- `const union devcmd vtable_keyboard`
- `const union devcmd vtable_keyboard_nonrgb`
- `const union devcmd vtable_mouse`

5.36.1 Data Structure Documentation

5.36.1.1 struct usbid

Definition at line 8 of file structures.h.

Collaboration diagram for usbid:



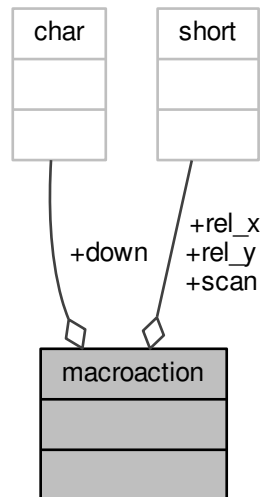
Data Fields

char	guid[16]	
char	modified[4]	

5.36.1.2 struct macroaction

Definition at line 27 of file structures.h.

Collaboration diagram for macroaction:



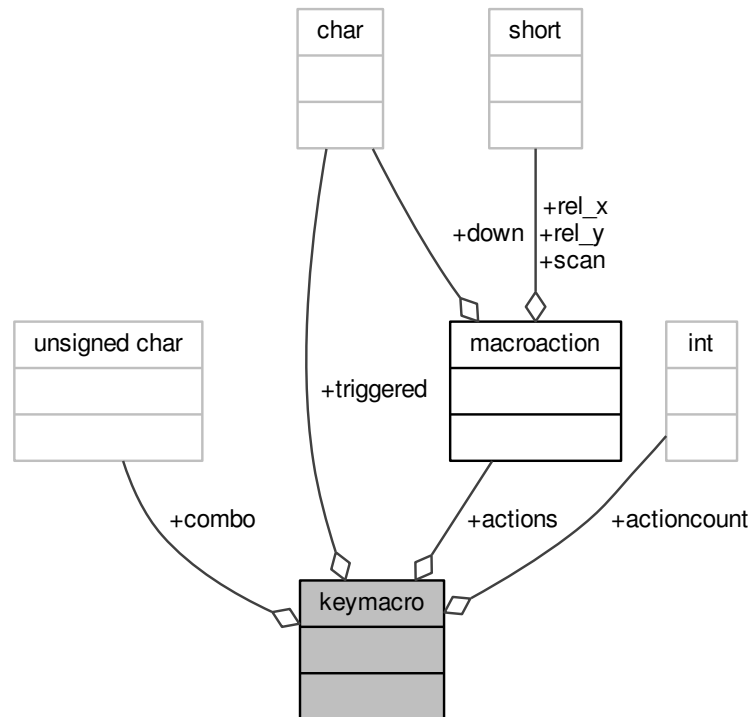
Data Fields

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:



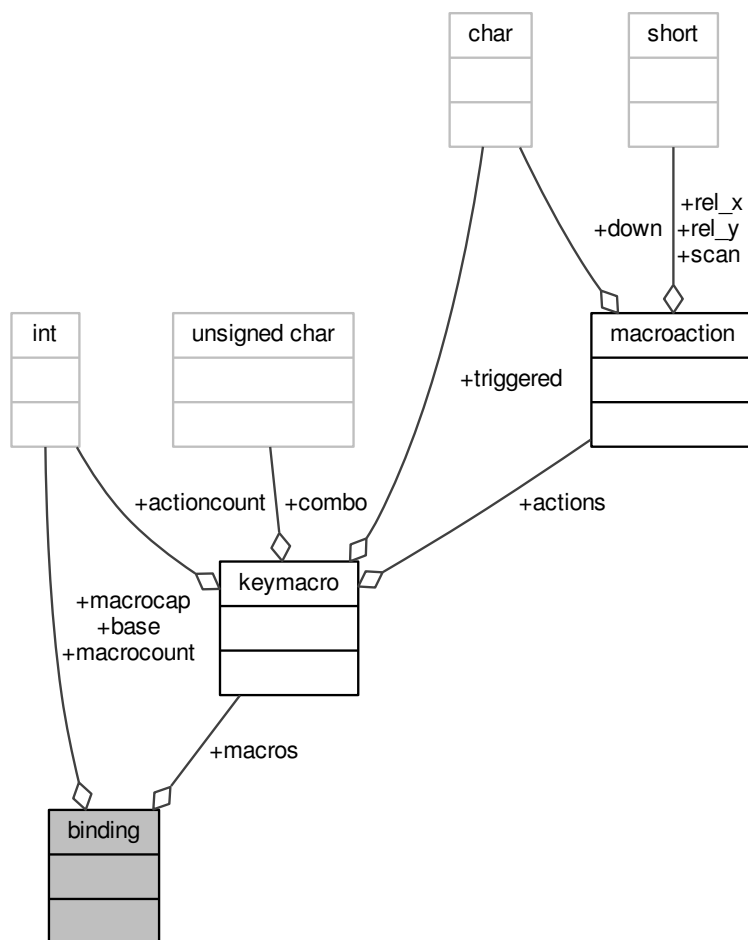
Data Fields

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:



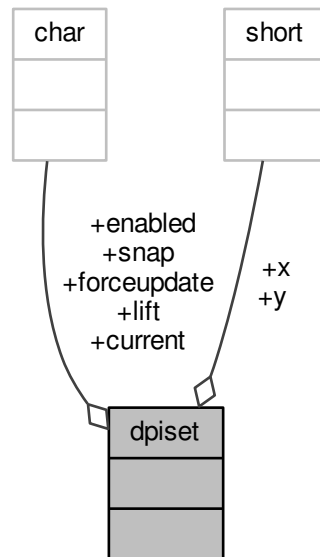
Data Fields

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:



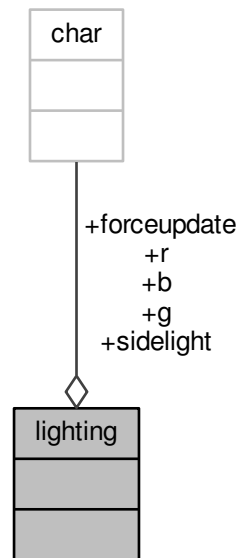
Data Fields

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:



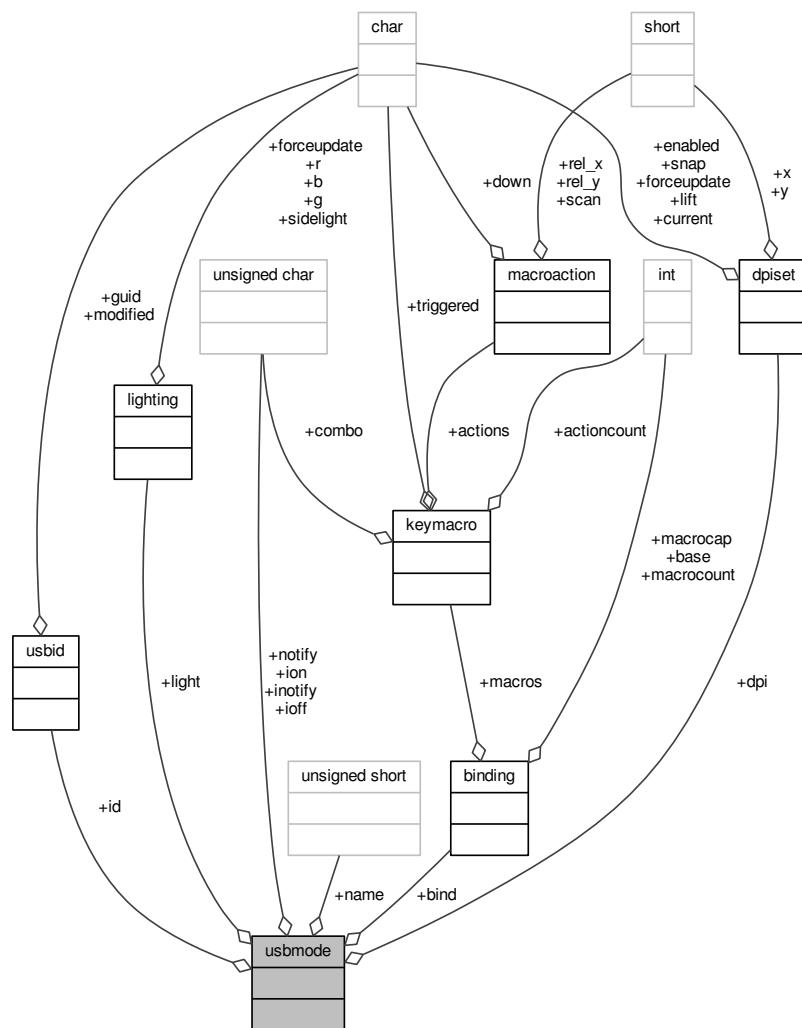
Data Fields

uchar	b[152+11]	
uchar	forceupdate	
uchar	g[152+11]	
uchar	r[152+11]	
uchar	sidelight	

5.36.1.7 struct usbmode

Definition at line 82 of file structures.h.

Collaboration diagram for usbmode:



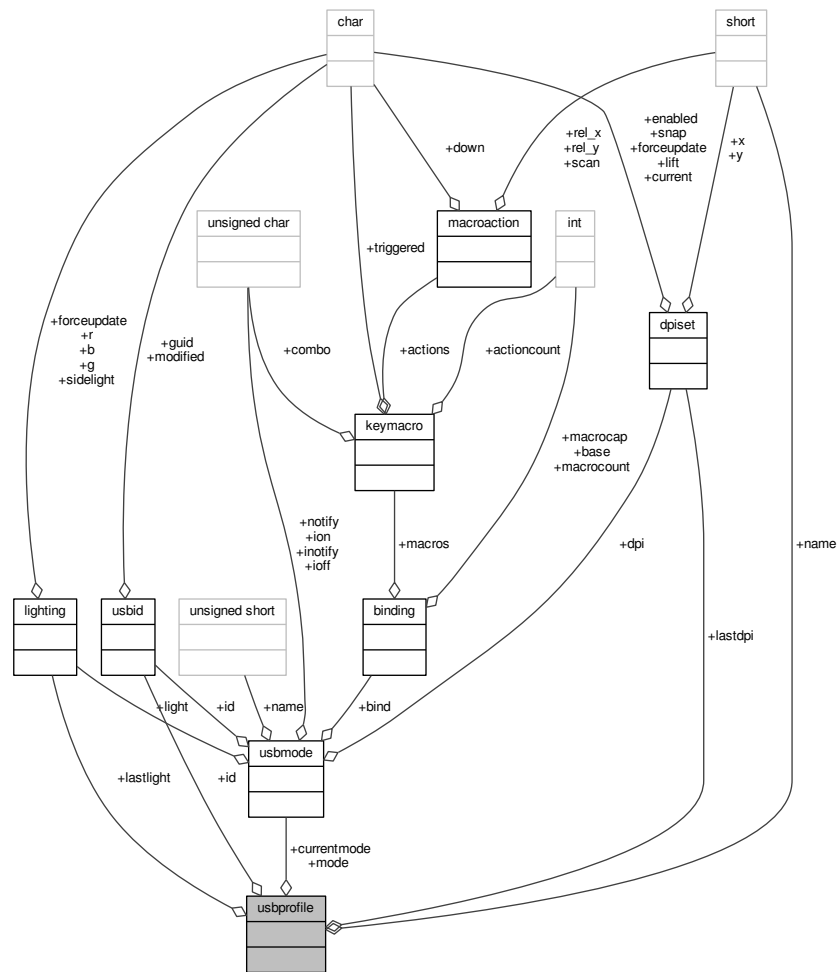
Data Fields

binding	bind	
dpiset	dpi	
usbuid	id	
uchar	inotify[10]	
uchar	ioff	
uchar	ion	
lighting	light	
ushort	name[16]	
uchar	notify[10][((((152+3+12)+25)+7)/8)]	

5.36.1.8 struct usbprofile

Definition at line 100 of file structures.h.

Collaboration diagram for usbprofile:



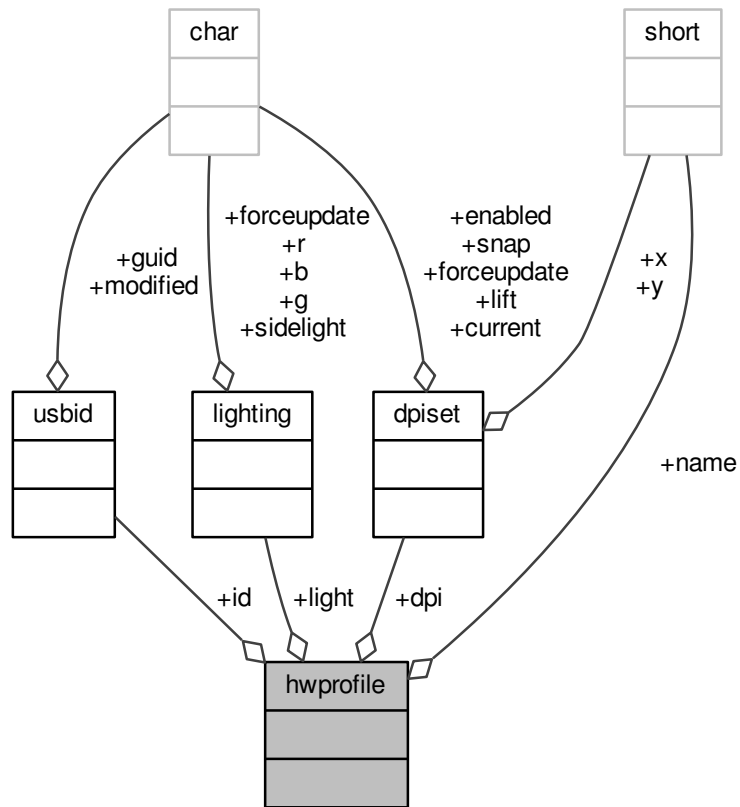
Data Fields

usbmode *	currentmode	
usbid	id	
dpiset	lastdpi	
lighting	lastlight	
usbmode	mode[6]	
ushort	name[16]	

5.36.1.9 struct hwprofile

Definition at line 117 of file structures.h.

Collaboration diagram for hwprofile:



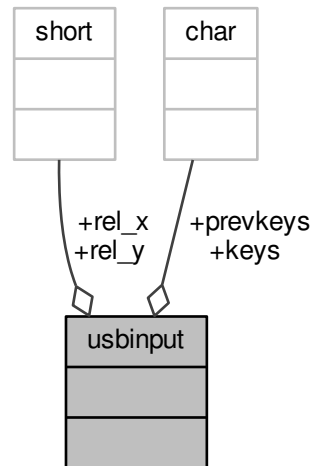
Data Fields

dpiset	<code>dpi[3]</code>	
usbid	<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.

```

classDiagram
    class lib {
        +do_io
    }
    class char {
        +down
        +rel_x
        +rel_y
        +scan
    }
    class short {
        +prevkeys
        +keys
        +rel_x
        +rel_y
    }
    class keymacro {
        +macrocap
        +base
        +macrocount
    }
    class unsigned_char {
        +macroaction
    }
    class unsigned_short {
    }
    class usbmode {
        +binding
        +usbmode
        +usbprofile
    }
    class usbprofile {
        +usbprofile
        +usbdevice
    }
    class usbdevice {
        +usbdevice
        +devcmd
    }
    class devcmd {
        +devcmd
    }
    class binding {
        +binding
    }
    class usb_id {
        +usb_id
    }
    class lighting {
        +lighting
    }
    class dpset {
        +dpset
    }
    class hwprofile {
        +hwprofile
    }
    class struct_udev_device {
        +struct_udev_device
    }
    class pthread_t {
        +pthread_t
    }
    lib --> char
    lib --> short
    lib --> keymacro
    lib --> unsigned_char
    lib --> unsigned_short
    lib --> usbmode
    lib --> usbprofile
    lib --> usbdevice
    lib --> devcmd
    char --> keymacro
    char --> unsigned_char
    char --> usb_id
    char --> lighting
    char --> dpset
    char --> hwprofile
    short --> keymacro
    short --> unsigned_char
    short --> usb_id
    short --> lighting
    short --> dpset
    short --> hwprofile
    keymacro --> unsigned_char
    keymacro --> usbmode
    keymacro --> usbprofile
    keymacro --> usbdevice
    unsigned_char --> usbmode
    unsigned_char --> usbprofile
    unsigned_char --> usbdevice
    unsigned_short --> usbmode
    unsigned_short --> usbprofile
    unsigned_short --> usbdevice
    usbmode --> usbprofile
    usbmode --> usbdevice
    usbprofile --> usbdevice
    usbdevice --> devcmd
    binding --> usbmode
    binding --> usbprofile
    binding --> usbdevice
    usb_id --> usbmode
    usb_id --> usbprofile
    usb_id --> usbdevice
    lighting --> usbmode
    lighting --> usbprofile
    lighting --> usbdevice
    dpset --> usbmode
    dpset --> usbprofile
    dpset --> usbdevice
    hwprofile --> usbmode
    hwprofile --> usbprofile
    hwprofile --> usbdevice
    struct_udev_device --> usbprofile
    struct_udev_device --> usbdevice
    pthread_t --> usbdevice
    
```

char	active	
char	delay	
char	dither	
int	epcount	
ushort	features	
ushort	fwversion	
int	handle	
hwprofile *	hw	
uchar	hw_ileds	
uchar	hw_ileds_old	
uchar	ileds	
int	infifo	
usbinput	input	
pthread_t	inputthread	
char	name[34+1]	

int	outfifo[10]	
char	pollrate	
short	product	
usbprofile *	profile	
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 `#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 `#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 `#define FEAT_ADJRATE 0x008`

Definition at line 138 of file structures.h.

Referenced by `_mkdevpath()`, `_setupusb()`, and `_start_dev()`.

5.36.2.4 `#define FEAT_ANSI 0x200`

Definition at line 145 of file structures.h.

Referenced by `readcmd()`.

5.36.2.5 `#define FEAT_BIND 0x010`

Definition at line 139 of file structures.h.

Referenced by `_mkdevpath()`, `main()`, and `readcmd()`.

5.36.2.6 `#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 `#define FEAT_FWUPDATE 0x080`

Definition at line 142 of file structures.h.

Referenced by `_mkdevpath()`, `_start_dev()`, and `cmd_fwupdate()`.

5.36.2.8 `#define FEAT_FWVERSION 0x040`

Definition at line 141 of file `structures.h`.

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

5.36.2.9 `#define FEAT_HWLOAD 0x100`

Definition at line 143 of file `structures.h`.

Referenced by `_start_dev()`.

5.36.2.10 `#define FEAT_ISO 0x400`

Definition at line 146 of file `structures.h`.

Referenced by `readcmd()`.

5.36.2.11 `#define FEAT_LMASK (FEAT_ANSI | FEAT_ISO)`

Definition at line 153 of file `structures.h`.

Referenced by `readcmd()`.

5.36.2.12 `#define FEAT_MONOCHROME 0x002`

Definition at line 136 of file `structures.h`.

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

5.36.2.13 `#define FEAT_MOUSEACCEL 0x800`

Definition at line 147 of file `structures.h`.

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

5.36.2.14 `#define FEAT_NOTIFY 0x020`

Definition at line 140 of file `structures.h`.

Referenced by `_mkdevpath()`, `main()`, and `readcmd()`.

5.36.2.15 `#define FEAT_POLLRATE 0x004`

Definition at line 137 of file `structures.h`.

Referenced by `_mkdevpath()`, `_start_dev()`, and `getfwversion()`.

5.36.2.16 `#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 `#define FEAT_STD_NRGB (FEAT_COMMON)`

Definition at line 152 of file structures.h.

Referenced by `_setupusb()`.

5.36.2.18 `#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 `#define HAS_ANY_FEATURE(kb, feat) (!!(kb)->features & (feat))`

Definition at line 157 of file structures.h.

5.36.2.20 `#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 `#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 `#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 `#define HWMODE_MAX 3`

Definition at line 116 of file structures.h.

5.36.2.24 `#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 `#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 `#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 #define IFACE_MAX 4

Definition at line 176 of file structures.h.

5.36.2.28 #define KB_NAME_LEN 34

Definition at line 173 of file structures.h.

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

5.36.2.29 #define LIFT_MAX 5

Definition at line 55 of file structures.h.

Referenced by `cmd_lift()`, and `loaddpi()`.

5.36.2.30 #define LIFT_MIN 1

Definition at line 54 of file structures.h.

Referenced by `cmd_lift()`, and `loaddpi()`.

5.36.2.31 #define MACRO_MAX 1024

Definition at line 50 of file structures.h.

Referenced by `_cmd_macro()`.

5.36.2.32 #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 #define MODE_COUNT 6

Definition at line 99 of file structures.h.

Referenced by `_freeprofile()`, `allocprofile()`, and `readcmd()`.

5.36.2.34 #define MSG_SIZE 64

Definition at line 175 of file structures.h.

Referenced by `_usbSend()`, `cmd_hwload_kb()`, `cmd_hwload_mouse()`, `cmd_hwsave_kb()`, `cmd_hwsave_mouse()`, `cmd_pollrate()`, `fwupdate()`, `getfwversion()`, `hwloadmode()`, `loaddpi()`, `loadrgb_kb()`, `loadrgb_mouse()`, `os_inputmain()`, `os_usbreceive()`, `os_usbSend()`, `savedpi()`, `savergb_kb()`, `savergb_mouse()`, `setactive_kb()`, `setactive_mouse()`, `updatedpi()`, `update_rgb_kb()`, and `update_rgb_mouse()`.

5.36.2.35 #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 `#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 `#define PR_NAME_LEN 16`

Definition at line 98 of file structures.h.

Referenced by `cmd_hwload_kb()`, `cmd_hwload_mouse()`, `cmd_profilename()`, `getprofilename()`, `hwtonative()`, and `nativetohw()`.

5.36.2.38 `#define SCROLL_ACCELERATED 0`

Definition at line 163 of file structures.h.

Referenced by `readcmd()`.

5.36.2.39 `#define SCROLL_MAX 10`

Definition at line 165 of file structures.h.

Referenced by `readcmd()`.

5.36.2.40 `#define SCROLL_MIN 1`

Definition at line 164 of file structures.h.

Referenced by `readcmd()`.

5.36.2.41 `#define SERIAL_LEN 34`

Definition at line 174 of file structures.h.

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

5.36.2.42 `#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 `const union devcmd vtable_keyboard`

Definition at line 28 of file device_vtable.c.

Referenced by `get_vtable()`.

5.36.3.2 `const union devcmd vtable_keyboard_nonrgb`

Definition at line 75 of file device_vtable.c.

Referenced by `get_vtable()`.

5.36.3.3 const union devcmd vtable_mouse

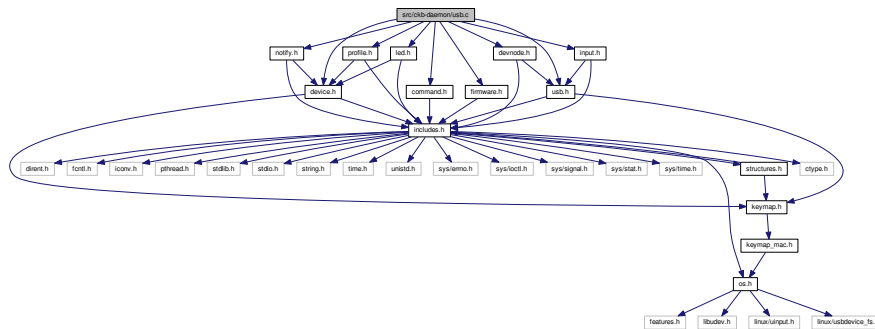
Definition at line 122 of file device_vtable.c.

Referenced by get_vtable().

5.37 src/ckb-daemon/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 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 `updatergb()` with `force=true`.

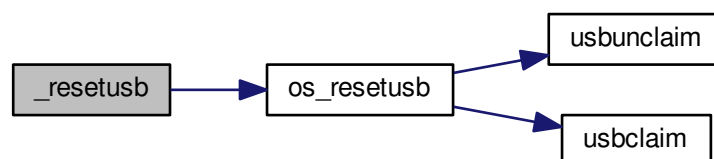
Definition at line 426 of file `usb.c`.

References `usbdevice::active`, `DELAY_LONG`, `os_resetusb()`, and `usbdevice::vtable`.

```

426                                     {
427     // Perform a USB reset
428     DELAY_LONG(kb);
429     int res = os_resetusb(kb, file, line);
430     if(res)
431         return res;
432     DELAY_LONG(kb);
433     // Re-initialize the device
434     if(kb->vtable->start(kb, kb->active) != 0)
435         return -1;
436     if(kb->vtable->updatergb(kb, 1) != 0)
437         return -1;
438     return 0;
439 }
```

Here is the call graph for this function:



5.37.1.2 static void* _setupusb (void * context) [static]

`_setupusb` A horrible function for setting up an usb device

Parameters

<i>context</i>	As _setupusb() is called as a new thread, the kb* is transferred as void*
----------------	---

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 vtable (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	cmd_active_kb() means: start the device with a lot of kb-specific initializers (software controlled mode)	cmd_idle_kb() set the device with a lot of kb-specific initializers into the hardware controlled mode)
non RGB Keyboard	cmd_io_none() means: Do nothing	cmd_io_none() means: Do nothing
Mouse	cmd_active_mouse() similar to cmd_active_kb()	cmd_idle_mouse similar to cmd_idle_kb()

- 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 214 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\(\)](#).

```

214                                     {
227     usbdevice* kb = context;
228     // Set standard fields
229     short vendor = kb->vendor, product = kb->product;
230     const devcmd* vt = kb->vtable = get_vtable(vendor, product);
231     kb->features = (IS_RGB(vendor, product) ? FEAT_STD_RGB :
FEAT_STD_NRGB) & features_mask;
232     if(IS_MOUSE(vendor, product)) kb->features |= FEAT_ADJRATE;
233     if(IS_MONOCHROME(vendor, product)) kb->features |=
FEAT_MONOCHROME;
234     kb->usbdelay = USB_DELAY_DEFAULT;
235
236     // 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);

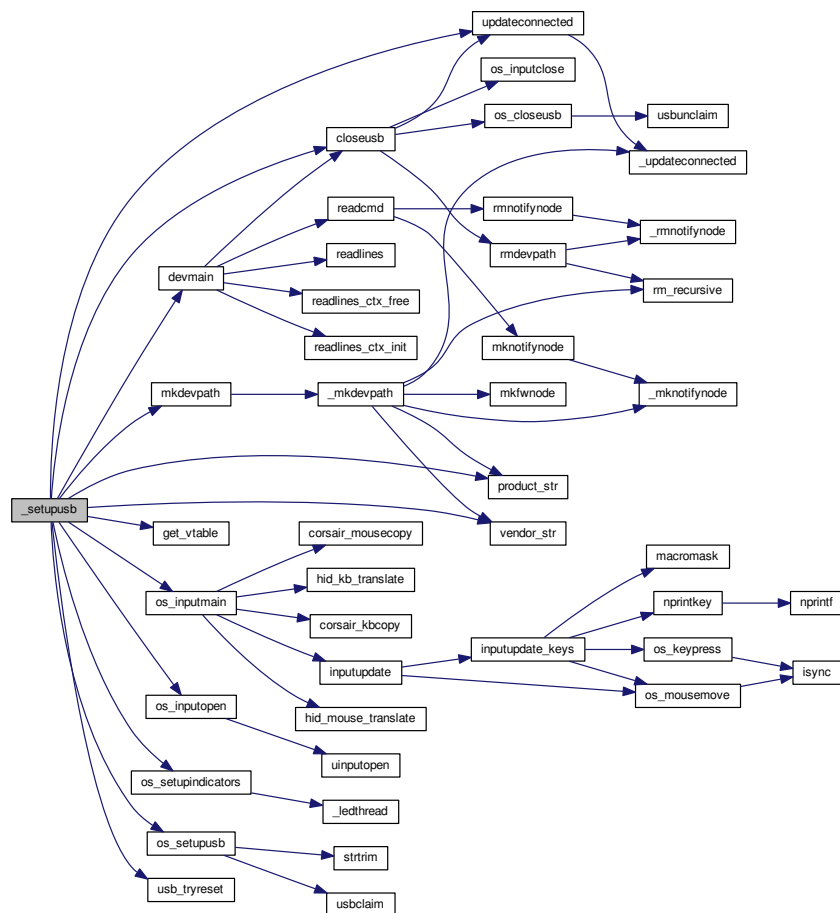
```

```

309     vt->updateindicators(kb, 1);
314     pthread_mutex_unlock(&imutex(kb));
348     if(vt->start(kb, 0) && usb_tryreset(kb))
349         goto fail_noinput;
355     // Make /dev path
356     if(mkdevpath(kb))
357         goto fail_noinput;
363     // Finished. Enter main loop
364     int index = INDEX_OF(kb, keyboard);
365     ckb_info("Setup finished for %s%d\n", devpath, index);
366     updateconnected();
369     return devmain(kb);
372 fail:
373     pthread_mutex_unlock(&imutex(kb));
374     fail_noinput:
375     closeusb(kb);
376     pthread_mutex_unlock(&dmutex(kb));
377     return 0;
378 }

```

Here is the call graph for this function:



Here is the caller graph for this function:



5.37.1.3 `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_usbsend()` as it is a machine depending implementation. The usb target device is as always determined over `kb`.

For `os_usbsend()` to know that it is a receive request, the `is_recv` parameter is set to true (1). With this, `os_usbsend()` 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.

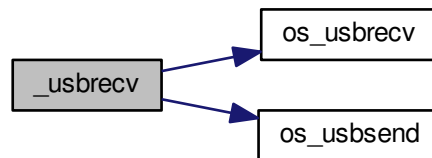
Definition at line 599 of file `usb.c`.

References `ckb_err_fn`, `DELAY_LONG`, `DELAY_MEDIUM`, `DELAY_SHORT`, `hwload_mode`, `os_usbrecv()`, `os_usbsend()`, and `reset_stop`.

```

599                                     {
600     // Try a maximum of 3 times
601     for(int try = 0; try < 5; try++){
602         // Send the output message
603         DELAY_SHORT(kb);
604         int res = os_usbsend(kb, out_msg, 1, file, line);
605         if(res == 0)
606             return 0;
607         else if(res == -1){
608             // Retry on temporary failure
609             if(reset_stop)
610                 return 0;
611             DELAY_LONG(kb);
612             continue;
613         }
614         // Wait for the response
615         DELAY_MEDIUM(kb);
616         res = os_usbrecv(kb, in_msg, file, line);
617         if(res == 0)
618             return 0;
619         else if(res != -1)
620             return res;
621         if(reset_stop || hwload_mode != 2)
622             return 0;
623         DELAY_LONG(kb);
624     }
625     // Give up
626     ckb_err_fn("Too many send/recv failures. Dropping.\n", file, line);
627     return 0;
628 }
```

Here is the call graph for this function:



5.37.1.4 `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 532 of file usb.c.

References `DELAY_LONG`, `DELAY_SHORT`, `hwload_mode`, `MSG_SIZE`, `os_usbSEND()`, and `reset_stop`.

```

532                                     {
533     int total_sent = 0;
534     for(int i = 0; i < count; i++){
535         // Send each message via the OS function
536         while(1){
537             DELAY_SHORT(kb);
538             int res = os_usbSEND(kb, messages + i * MSG_SIZE, 0, file, line);
539             if(res == 0)
540                 return 0;
541             else if(res != -1){
542                 total_sent += res;
543                 break;
544             }
545             // Stop immediately if the program is shutting down or hardware load is set to tryonce
546             if(reset_stop || hwload_mode != 2)
547                 return 0;
548             // Retry as long as the result is temporary failure
549             DELAY_LONG(kb);
550         }
551     }
552     return total_sent;
553 }
```

Here is the call graph for this function:



5.37.1.5 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 673 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`.

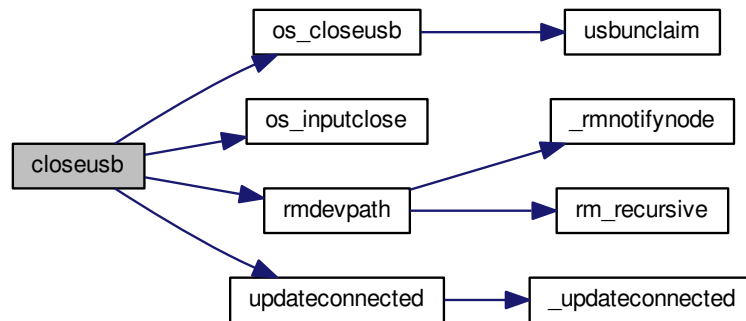
Referenced by `_setupusb()`, `devmain()`, `quitWithLock()`, and `usb_rm_device()`.

```

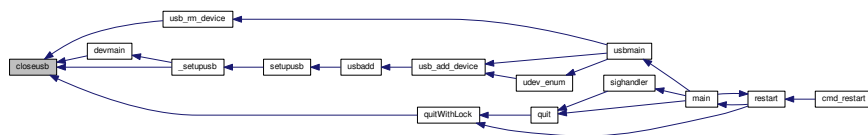
673         {
674     pthread_mutex_lock(imutex(kb));
675     if(kb->handle){
676         int index = INDEX_OF(kb, keyboard);
677         ckb_info("Disconnecting %s%d\n", devpath, index);
678         os_inputclose(kb);
679         updateconnected();
680         // Close USB device
681         os_closeusb(kb);
682     } else
683         updateconnected();
684     rmdevpath(kb);
685
686     // Wait for thread to close
687     pthread_mutex_unlock(imutex(kb));
688     pthread_mutex_unlock(dmutex(kb));
689     pthread_join(kb->thread, 0);
690     pthread_mutex_lock(dmutex(kb));
691
692     // Delete the profile and the control path
693     if(!kb->vtable)
694         return 0;
695     kb->vtable->freeprofile(kb);
696     memset(kb, 0, sizeof(usbdevice));
697     return 0;
698 }

```

Here is the call graph for this function:



Here is the caller graph for this function:



5.37.1.6 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 `mutex` 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 `closeusb()`, `dmutex`, `usbdevice::infifo`, `IS_CONNECTED`, `readcmd()`, `readlines()`, `readlines_ctx_free()`, and `readlines_ctx_init()`.

Referenced by `_setupusb()`.

```

135                                     {
137     int kbfifo = kb->infifo - 1;
140     readlines_ctx 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             break;
164         if(lines){
167             if(readcmd(kb, line)){
173                 // USB transfer failed; destroy device
174                 closeusb(kb);
175                 break;

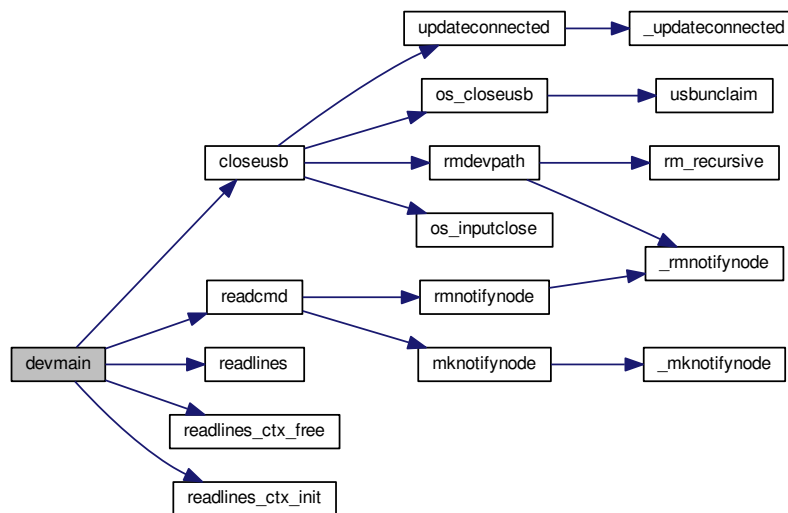
```

```

176         }
177     }
178 }
179 pthread_mutex_unlock (dmutex (kb));
182 readlines_ctx_free (linectx);
183 return 0;
184 }

```

Here is the call graph for this function:



Here is the caller graph for this function:



5.37.1.7 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 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().

```

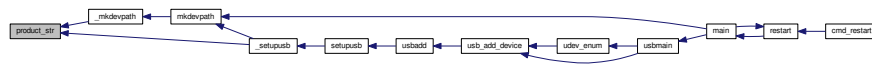
70                                     {
71     if (product == P_K95 || product == P_K95_NRGB || product ==
       P_K95_PLATINUM)
72         return "k95";
73     if (product == P_K70 || product == P_K70_NRGB || product ==
       P_K70_LUX || product == P_K70_LUX_NRGB || product ==
       P_K70_RFIRE || 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:



5.37.1.9 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 407 of file `usb.c`.

References `FEAT_RGB`, `HAS_FEATURES`, `NEEDS_FW_UPDATE`, `NK95_HWON`, `nk95cmd`, and `setactive`.

Referenced by `quitWithLock()`.

```

407     {
408         if (NEEDS_FW_UPDATE(kb))
409             return 0;
410         if (!HAS_FEATURES(kb, FEAT_RGB)) {
411             nk95cmd(kb, NK95_HWON);
412             return 0;
413         }
414         if (setactive(kb, 0))
415             return -1;
416         return 0;
417     }

```

Here is the caller graph for this function:



5.37.1.10 void setupusb (usbdevice * kb)

setupusb starts a thread with kb as parameter and [_setupusb\(\)](#) as entrypt.

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 386 of file usb.c.

References [_setupusb\(\)](#), [ckb_err](#), [imutex](#), and [usbdevice::thread](#).

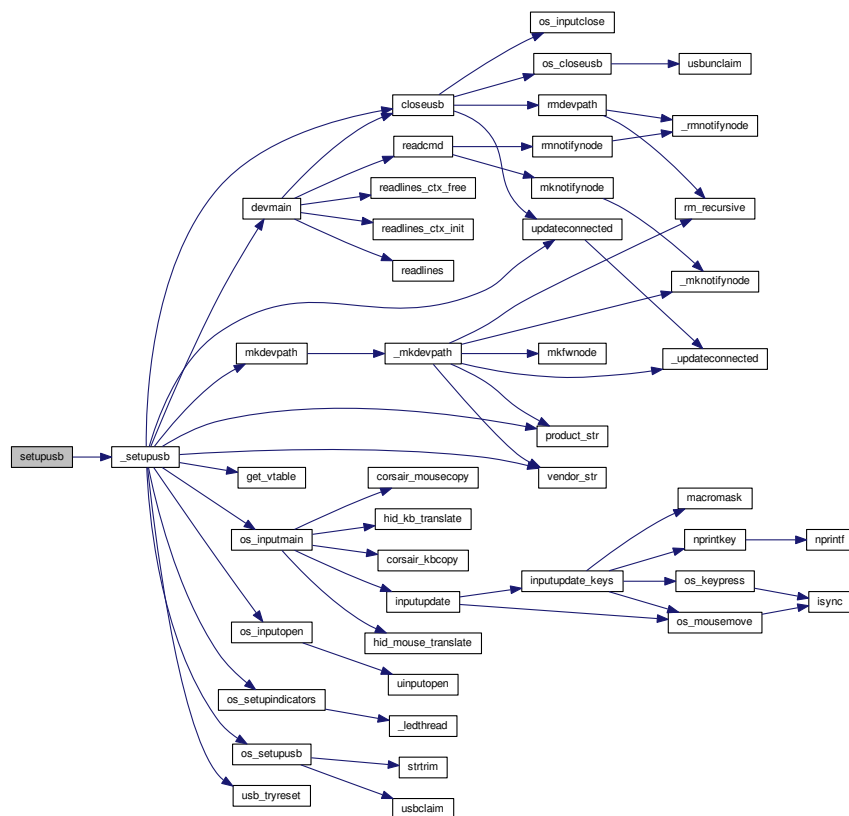
Referenced by [usbadd\(\)](#).

```

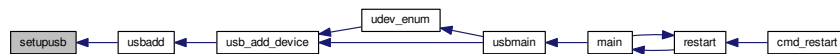
386 {
387     pthread_mutex_lock(&imutex(kb));
388     if(pthread_create(&kb->thread, 0, _setupusb, kb))
389         ckb_err("Failed to create USB thread\n");
390 }

```

Here is the call graph for this function:



Here is the caller graph for this function:



5.37.1.11 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 465 of file usb.c.

References [ckb_err](#), [ckb_info](#), [reset_stop](#), and [resetusb](#).

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

```

465         {
466     if(reset_stop)
467         return -1;
468     ckb_info("Attempting reset...\n");
469     while(1){
470         int res = resetusb(kb);
471         if(!res){
472             ckb_info("Reset success\n");
473             return 0;
474         }
475         if(res == -2 || reset_stop)
476             break;
477     }
478     ckb_err("Reset failed. Disconnecting.\n");
479     return -1;
480 }

```

Here is the caller graph for this function:



5.37.1.12 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*.

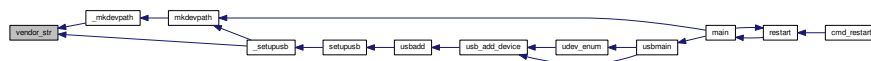
Referenced by *_mkdevpath()*, and *_setupusb()*.

```

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 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 int hwload_mode

Definition at line 7 of file device.c.

Referenced by *_start_dev()*, *_usbrecv()*, and *_usbSEND()*.


```

• #define P_K65_LUX_STR "1b37"
• #define P_K65_RFIRE 0x1b39
• #define P_K65_RFIRE_STR "1b39"
• #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))
• #define P_K70 0x1b13
• #define P_K70_STR "1b13"
• #define P_K70_NRGB 0x1b09
• #define P_K70_NRGB_STR "1b09"
• #define P_K70_LUX 0x1b33
• #define P_K70_LUX_STR "1b33"
• #define P_K70_LUX_NRGB 0x1b36
• #define P_K70_LUX_NRGB_STR "1b36"
• #define P_K70_RFIRE 0x1b38
• #define P_K70_RFIRE_STR "1b38"
• #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() .
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()

Definition in file [usb.h](#).

5.38.2 Macro Definition Documentation

5.38.2.1 `#define DELAY_LONG(kb) usleep(100000)`

Definition at line 156 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 `#define DELAY_MEDIUM(kb) usleep((int)(kb)->usbdelay * 10000)`

Definition at line 153 of file [usb.h](#).

Referenced by [_usbrecv\(\)](#), and [setactive_kb\(\)](#).

5.38.2.3 `#define DELAY_SHORT(kb) usleep((int)(kb)->usbdelay * 1000)`

Definition at line 150 of file [usb.h](#).

Referenced by [_usbrecv\(\)](#), [_usbseend\(\)](#), and [updateindicators_kb\(\)](#).

5.38.2.4 `#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 139 of file [usb.h](#).

Referenced by [readcmd\(\)](#), and [updatergb_kb\(\)](#).

```
5.38.2.5 #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 #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 #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 #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 #define IS_MONOCHROME( vendor, product ) ((vendor) == (V_CORSAIR) && (product) == (P_STRAFE_NRGB))
```

Definition at line 130 of file usb.h.

Referenced by _setupusb().

```
5.38.2.10 #define IS_MONOCHROME_DEV( kb ) IS_MONOCHROME((kb)->vendor, (kb)->product)
```

Definition at line 136 of file usb.h.

```
5.38.2.11 #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 142 of file usb.h.

Referenced by _setupusb(), get_vtable(), has_key(), and os_inputmain().

```
5.38.2.12 #define IS_MOUSE_DEV( kb ) IS_MOUSE((kb)->vendor, (kb)->product)
```

Definition at line 145 of file usb.h.

Referenced by readcmd().

```
5.38.2.13 #define IS_RGB( vendor, product ) ((vendor) == (V_CORSAIR) && (product) != (P_K65_NRGB) && (product) !=
(P_K70_NRGB) && (product) != (P_K95_NRGB))
```

Definition at line 125 of file usb.h.

Referenced by `_setupusb()`, `get_vtable()`, and `os_inputmain()`.

5.38.2.14 `#define IS_RGB_DEV(kb) IS_RGB((kb)->vendor, (kb)->product)`

Definition at line 133 of file `usb.h`.

5.38.2.15 `#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 `#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 `#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 `#define NK95_HWOFF 0x020030`

See Also

[usb2.0 documentation for details](#). Set Hardware playback off

Definition at line 302 of file `usb.h`.

Referenced by `start_kb_nrgb()`.

5.38.2.19 `#define NK95_HWON 0x020001`

Definition at line 305 of file `usb.h`.

Referenced by `revertusb()`.

5.38.2.20 `#define NK95_M1 0x140001`

Definition at line 308 of file `usb.h`.

Referenced by `setmodeindex_nrgb()`.

5.38.2.21 `#define NK95_M2 0x140002`

Definition at line 311 of file `usb.h`.

Referenced by `setmodeindex_nrgb()`.

5.38.2.22 `#define NK95_M3 0x140003`

Definition at line 314 of file usb.h.

Referenced by `setmodeindex_nrgb()`.

5.38.2.23 `#define nk95cmd(kb, command) _nk95cmd(kb, (command) >> 16 & 0xFF, (command) & 0xFFFF, __FILE__NOPATH__, __LINE__)`

Definition at line 297 of file usb.h.

Referenced by `revertusb()`, `setmodeindex_nrgb()`, and `start_kb_nrgb()`.

5.38.2.24 `#define P_HARPOON 0x1b3c /* Harpoon test */`

Definition at line 93 of file usb.h.

Referenced by `product_str()`.

5.38.2.25 `#define P_HARPOON_STR "1b3c"`

Definition at line 94 of file usb.h.

5.38.2.26 `#define P_K65 0x1b17`

Definition at line 41 of file usb.h.

Referenced by `product_str()`.

5.38.2.27 `#define P_K65_LUX 0x1b37`

Definition at line 45 of file usb.h.

Referenced by `product_str()`.

5.38.2.28 `#define P_K65_LUX_STR "1b37"`

Definition at line 46 of file usb.h.

5.38.2.29 `#define P_K65_NRGB 0x1b07`

Definition at line 43 of file usb.h.

Referenced by `product_str()`.

5.38.2.30 `#define P_K65_NRGB_STR "1b07"`

Definition at line 44 of file usb.h.

5.38.2.31 `#define P_K65_RFIRE 0x1b39`

Definition at line 47 of file usb.h.

Referenced by `product_str()`.

5.38.2.32 `#define P_K65_RFIRE_STR "1b39"`

Definition at line 48 of file usb.h.

5.38.2.33 `#define P_K65_STR "1b17"`

Definition at line 42 of file usb.h.

5.38.2.34 `#define P_K70 0x1b13`

Definition at line 51 of file usb.h.

Referenced by `product_str()`.

5.38.2.35 `#define P_K70_LUX 0x1b33`

Definition at line 55 of file usb.h.

Referenced by `product_str()`.

5.38.2.36 `#define P_K70_LUX_NRGB 0x1b36`

Definition at line 57 of file usb.h.

Referenced by `product_str()`.

5.38.2.37 `#define P_K70_LUX_NRGB_STR "1b36"`

Definition at line 58 of file usb.h.

5.38.2.38 `#define P_K70_LUX_STR "1b33"`

Definition at line 56 of file usb.h.

5.38.2.39 `#define P_K70_NRGB 0x1b09`

Definition at line 53 of file usb.h.

Referenced by `product_str()`.

5.38.2.40 `#define P_K70_NRGB_STR "1b09"`

Definition at line 54 of file usb.h.

5.38.2.41 `#define P_K70_RFIRE 0x1b38`

Definition at line 59 of file usb.h.

Referenced by `product_str()`.

5.38.2.42 `#define P_K70_RFIRE_NRGB 0x1b3a`

Definition at line 61 of file usb.h.

Referenced by `product_str()`.

5.38.2.43 `#define P_K70_RFIRE_NRGB_STR "1b3a"`

Definition at line 62 of file usb.h.

5.38.2.44 `#define P_K70_RFIRE_STR "1b38"`

Definition at line 60 of file usb.h.

5.38.2.45 `#define P_K70_STR "1b13"`

Definition at line 52 of file usb.h.

5.38.2.46 `#define P_K95 0x1b11`

Definition at line 65 of file usb.h.

Referenced by `product_str()`.

5.38.2.47 `#define P_K95_NRGB 0x1b08`

Definition at line 67 of file usb.h.

Referenced by `_nk95cmd()`, and `product_str()`.

5.38.2.48 `#define P_K95_NRGB_STR "1b08"`

Definition at line 68 of file usb.h.

5.38.2.49 `#define P_K95_PLATINUM 0x1b2d`

Definition at line 69 of file usb.h.

Referenced by `product_str()`.

5.38.2.50 `#define P_K95_PLATINUM_STR "1b2d"`

Definition at line 70 of file usb.h.

5.38.2.51 `#define P_K95_STR "1b11"`

Definition at line 66 of file usb.h.

5.38.2.52 `#define P_M65 0x1b12`

Definition at line 79 of file usb.h.

Referenced by `product_str()`.

5.38.2.53 #define P_M65_PRO 0x1b2e

Definition at line 81 of file usb.h.

Referenced by product_str().

5.38.2.54 #define P_M65_PRO_STR "1b2e"

Definition at line 82 of file usb.h.

5.38.2.55 #define P_M65_STR "1b12"

Definition at line 80 of file usb.h.

5.38.2.56 #define P_SABRE_L 0x1b19 /* laser */

Definition at line 87 of file usb.h.

Referenced by product_str().

5.38.2.57 #define P_SABRE_L_STR "1b19"

Definition at line 88 of file usb.h.

5.38.2.58 #define P_SABRE_N 0x1b2f /* new? */

Definition at line 89 of file usb.h.

Referenced by product_str().

5.38.2.59 #define P_SABRE_N_STR "1b2f"

Definition at line 90 of file usb.h.

5.38.2.60 #define P_SABRE_O 0x1b14 /* optical */

Definition at line 85 of file usb.h.

Referenced by product_str().

5.38.2.61 #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 #define P_SABRE_O2_STR "1b32"

Definition at line 92 of file usb.h.

5.38.2.63 #define P_SABRE_O_STR "1b14"

Definition at line 86 of file usb.h.

5.38.2.64 `#define P_SCIMITAR 0x1b1e`

Definition at line 97 of file usb.h.

Referenced by `product_str()`.

5.38.2.65 `#define P_SCIMITAR_PRO 0x1b3e`

Definition at line 99 of file usb.h.

Referenced by `product_str()`.

5.38.2.66 `#define P_SCIMITAR_PRO_STR "1b3e"`

Definition at line 100 of file usb.h.

5.38.2.67 `#define P_SCIMITAR_STR "1b1e"`

Definition at line 98 of file usb.h.

5.38.2.68 `#define P_STRAFE 0x1b20`

Definition at line 73 of file usb.h.

Referenced by `product_str()`.

5.38.2.69 `#define P_STRAFE_NRGB 0x1b15`

Definition at line 75 of file usb.h.

Referenced by `product_str()`.

5.38.2.70 `#define P_STRAFE_NRGB_STR "1b15"`

Definition at line 76 of file usb.h.

5.38.2.71 `#define P_STRAFE_STR "1b20"`

Definition at line 74 of file usb.h.

5.38.2.72 `#define resetusb(kb) _resetusb(kb, __FILE__ __NOPATH__, __LINE__)`

Definition at line 215 of file usb.h.

Referenced by `usb_tryreset()`.

5.38.2.73 `#define USB_DELAY_DEFAULT 5`

Definition at line 161 of file usb.h.

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

5.38.2.74 `#define usbrecv(kb, out_msg, in_msg) _usbrecv(kb, out_msg, in_msg, __FILE__ __NOPATH__, __LINE__)`

Parameters

<i>kb</i>	THE usbdevice*
<i>IN]</i>	out_msg What information does the caller want from the device?
<i>OUT]</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 257 of file usb.h.

Referenced by cmd_hwload_kb(), cmd_hwload_mouse(), getfwversion(), hwloadmode(), loaddpi(), loadrgb_kb(), and loadrgb_mouse().

5.38.2.75 `#define usbsend(kb, messages, count) _usbsend(kb, messages, count, __FILE__NOPATH__, __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 240 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 `#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 `#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 `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_ctransfer` 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 187 of file `usb_linux.c`.

References `ckb_err_fn`, `usbdevice::handle`, `P_K95_NRGB`, and `usbdevice::product`.

```

187
188     if (kb->product != P_K95_NRGB)
189         return 0;

```



```

190     struct usbdevfs_ctrltransfer transfer = { 0x40, bRequest, wValue, 0, 0, 5000, 0 };
191     int res = ioctl(kb->handle - 1, USBDEVFS_CONTROL, &transfer);
192     if(res <= 0){
193         ckb_err_fn("%s\n", file, line, res ? strerror(errno) : "No data written");
194         return 1;
195     }
196     return 0;
197 }

```

5.38.3.2 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 dependent `updatergb()` with `force=true`.

Definition at line 426 of file `usb.c`.

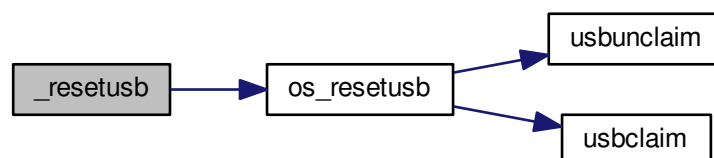
References `usbdevice::active`, `DELAY_LONG`, `os_resetusb()`, and `usbdevice::vtable`.

```

426                                     {
427     // Perform a USB reset
428     DELAY_LONG(kb);
429     int res = os_resetusb(kb, file, line);
430     if(res)
431         return res;
432     DELAY_LONG(kb);
433     // Re-initialize the device
434     if(kb->vtable->start(kb, kb->active) != 0)
435         return -1;
436     if(kb->vtable->updatergb(kb, 1) != 0)
437         return -1;
438     return 0;
439 }

```

Here is the call graph for this function:



5.38.3.3 int _usbrecv (usbdevice * kb, const uchar * out_msg, uchar * in_msg, const char * file, int line)

Parameters

	<i>kb</i>	THE usbdevice*
	<i>IN]</i>	out_msg What information does the caller want from the device?
	<i>OUT]</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.
	<i>IN]</i>	file for debugging
	<i>IN]</i>	line for debugging
<i>in</i>	<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_usbseend()` as it is a machine depending implementation. The usb target device is as always determined over *kb*.

For `os_usbseend()` to know that it is a receive request, the **is_recv** parameter is set to true (1). With this, `os_usbseend()` 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 599 of file *usb.c*.

References *ckb_err_fn*, *DELAY_LONG*, *DELAY_MEDIUM*, *DELAY_SHORT*, *hwload_mode*, `os_usbrecv()`, `os_usbseend()`, and *reset_stop*.

```

599
600     // Try a maximum of 3 times
601     for(int try = 0; try < 5; try++){
602         // Send the output message
603         DELAY_SHORT(kb);
604         int res = os_usbseend(kb, out_msg, 1, file, line);
605         if(res == 0)
606             return 0;
607         else if(res == -1){
608             // Retry on temporary failure
609             if(reset_stop)
610                 return 0;
611             DELAY_LONG(kb);

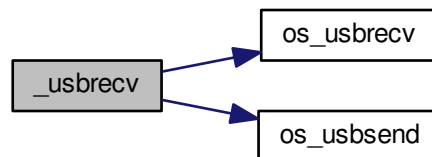
```

```

612         continue;
613     }
614     // Wait for the response
615     DELAY_MEDIUM(kb);
616     res = os_usbrecv(kb, in_msg, file, line);
617     if(res == 0)
618         return 0;
619     else if(res != -1)
620         return res;
621     if(reset_stop || hwload_mode != 2)
622         return 0;
623     DELAY_LONG(kb);
624 }
625 // Give up
626 ckb_err_fn("Too many send/recv failures. Dropping.\n", file, line);
627 return 0;
628 }

```

Here is the call graph for this function:



5.38.3.4 int _usbsend (usbdevice * kb, const uchar * messages, int count, const char * file, int 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?
	<i>IN]</i>	file for debugging
	<i>IN]</i>	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 numer 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 532 of file `usb.c`.

References `DELAY_LONG`, `DELAY_SHORT`, `hwload_mode`, `MSG_SIZE`, `os_usbsend()`, and `reset_stop`.

```

532                                     {
533     int total_sent = 0;
534     for(int i = 0; i < count; i++){
535         // Send each message via the OS function
536         while(1){
537             DELAY_SHORT(kb);
538             int res = os_usbsend(kb, messages + i * MSG_SIZE, 0, file, line);
539             if(res == 0)
540                 return 0;
541             else if(res != -1){
542                 total_sent += res;
543                 break;
544             }
545             // Stop immediately if the program is shutting down or hardware load is set to tryonce
546             if(reset_stop || hwload_mode != 2)
547                 return 0;
548             // Retry as long as the result is temporary failure
549             DELAY_LONG(kb);
550         }
551     }
552     return total_sent;
553 }
```

Here is the call graph for this function:



5.38.3.5 int closeusb (usbdevice * kb)

Parameters

<i>IN,OUT]</i>	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 673 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`.

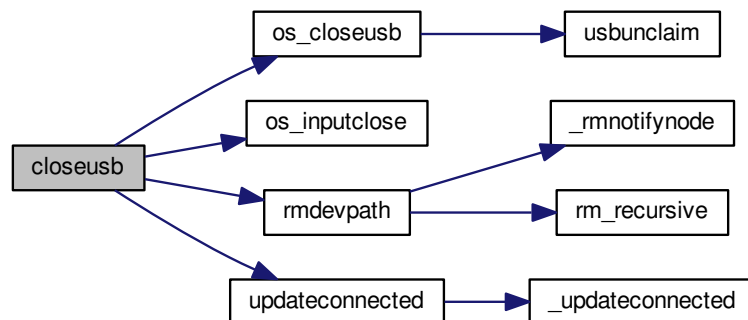
Referenced by `_setupusb()`, `devmain()`, `quitWithLock()`, and `usb_rm_device()`.

```

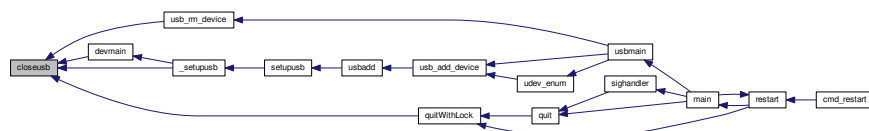
673     {
674         pthread_mutex_lock(imutex(kb));
675         if(kb->handle){
676             int index = INDEX_OF(kb, keyboard);
677             ckb_info("Disconnecting %s%d\n", devpath, index);
678             os_inputclose(kb);
679             updateconnected();
680             // Close USB device
681             os_closeusb(kb);
682         } else
683             updateconnected();
684         rmdevpath(kb);
685
686         // Wait for thread to close
687         pthread_mutex_unlock(imutex(kb));
688         pthread_mutex_unlock(dmutex(kb));
689         pthread_join(kb->thread, 0);
690         pthread_mutex_lock(dmutex(kb));
691
692         // Delete the profile and the control path
693         if(!kb->vtable)
694             return 0;
695         kb->vtable->freeprofile(kb);
696         memset(kb, 0, sizeof(usbdevice));
697         return 0;
698     }

```

Here is the call graph for this function:



Here is the caller graph for this function:



5.38.3.6 void os_closeusb (usbdevice * kb)

Parameters

<i>IN,OUT]</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 422 of file usb_linux.c.

References `usbdevice::handle`, `INDEX_OF`, `kbsyspath`, `keyboard`, `usbdevice::udev`, and `usbunclaim()`.

Referenced by closeusb().

```

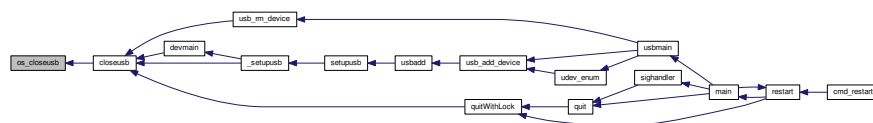
422                                     {
423     if (kb->handle) {
424         usbunclaim(kb, 0);
425         close(kb->handle - 1);
426     }
427     if (kb->udev)
428         udev_device_unref(kb->udev);
429     kb->handle = 0;
430     kb->udev = 0;
431     kbsyspath[INDEX_OF(kb, keyboard)][0] = 0;
432 }

```

Here is the call graph for this function:



Here is the caller graph for this function:



5.38.3.7 void* os_inputmain (void * context)

Parameters

<i>context</i>	THE usbdevice* ; Because <code>os_inputmain()</code> is started as a new thread, its formal parameter is named "context".
----------------	---

Returns

null

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.

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

1. set up an URB (Userspace Ressource Buffer) to communicate with the USBDEVFS_* ioctl()s
2. perform the ioctl()
3. interpretate the information got into the URB buffer or handle error situations and retry operation or leave the endless loop
4. inform the os about the data
5. loop endless via 2.
6. if endless loop has gone, deinitalize the interface, free buffers etc.
7. return null

Here the actions in detail:

Monitor input transfers on all endpoints for non-RGB devices For RGB, monitor all but the last, as it's used for input/output

Get an usbdevfs_urb data structure and clear it via memset()

Hopefully the buffer lengths are equal for all devices with congruent types. You can find out the correctness for your device with lsusb -v or similar on macOS. Currently the following combinations are known and implemented:

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 232 of file `usb_linux.c`.

References `usbdevice::active`, `ckb_info`, `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()`.

```

232                                     {
233     usbdevice* kb = context;
234     int fd = kb->handle - 1;
235     short vendor = kb->vendor, product = kb->product;
236     int index = INDEX_OF(kb, keyboard);
237     ckb_info("Starting input thread for %s%d\n", devpath, index);
238
239     int urbcount = IS_RGB(vendor, product) ? (kb->epcount - 1) : kb->
240     epcount;
241
242     struct usbdevfs_urb urbs[urbcount];
243     memset(urbs, 0, sizeof(urbs));
244
245     urbs[0].buffer_length = 8;
246     if(IS_RGB(vendor, product)){
247         if(IS_MOUSE(vendor, product))
248             urbs[1].buffer_length = 10;
249         else
250             urbs[1].buffer_length = 21;
251         urbs[2].buffer_length = MSG_SIZE;
252         if(urbcount != 3)
253             urbs[urbcount - 1].buffer_length = MSG_SIZE;
254     } else {
255         urbs[1].buffer_length = 4;
256         urbs[2].buffer_length = 15;
257     }
258
259     for(int i = 0; i < urbcount; i++){
260         urbs[i].type = USBDEVFS_URB_TYPE_INTERRUPT;
261         urbs[i].endpoint = 0x80 | (i + 1);
262         urbs[i].buffer = malloc(urbs[i].buffer_length);
263         ioctl(fd, USBDEVFS_SUBMITURB, urbs + i);
264     }
265
266     while (1) {
267         struct usbdevfs_urb* urb = 0;
268
269         if (ioctl(fd, USBDEVFS_REAPURB, &urb)){
270             if (errno == ENODEV || errno == ENOENT || errno == ESHUTDOWN)
271                 // Stop the thread if the handle closes
272                 break;
273             else if(errno == EPIPE && urb){
274                 ioctl(fd, USBDEVFS_CLEAR_HALT, &urb->endpoint);
275                 // Re-submit the URB
276                 if(urb)
277                     ioctl(fd, USBDEVFS_SUBMITURB, urb);
278                 urb = 0;
279             }
280         }
281     }
282 }

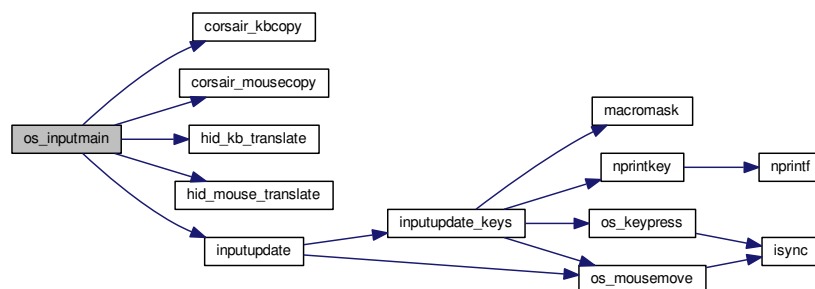
```

```

304
308     if (urb) {
320         pthread_mutex_lock(&mutex(kb));
321         if(IS_MOUSE(vendor, product)){
322             switch(urb->actual_length){
323                 case 8:
324                 case 10:
325                 case 11:
326                 // HID mouse input
327                 hid_mouse_translate(kb->input.keys, &kb->
input.rel_x, &kb->input.rel_y, -(urb->endpoint & 0xF), urb->actual_length, urb->buffer)
;
328                 break;
329                 case MSG_SIZE:
330                 // Corsair mouse input
331                 corsair_mousecopy(kb->input.keys, -(urb->endpoint & 0xF), urb
->buffer);
332                 break;
333             }
334         } else if(IS_RGB(vendor, product)){
335             switch(urb->actual_length){
336                 case 8:
337                 // RGB EP 1: 6KRO (BIOS mode) input
338                 hid_kb_translate(kb->input.keys, -1, urb->actual_length, urb->
buffer);
339                 break;
340                 case 21:
341                 case 5:
342                 // RGB EP 2: NKRO (non-BIOS) input. Accept only if keyboard is inactive
343                 if(!kb->active)
344                 hid_kb_translate(kb->input.keys, -2, urb->actual_length,
urb->buffer);
345                 break;
346                 case MSG_SIZE:
347                 // RGB EP 3: Corsair input
348                 corsair_kbcopy(kb->input.keys, -(urb->endpoint & 0xF), urb->
buffer);
349                 break;
350             }
351         } else {
352             // Non-RGB input
353             hid_kb_translate(kb->input.keys, urb->endpoint & 0xF, urb->
actual_length, urb->buffer);
354         }
355         inputupdate(kb);
356         pthread_mutex_unlock(&mutex(kb));
357         ioctl(fd, USBDEVFS_SUBMITURB, urb);
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 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 480 of file usb_linux.c.

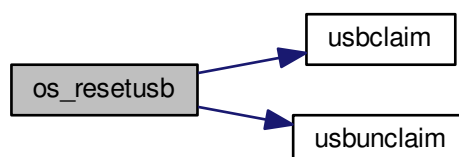
References usbdevice::handle, TEST_RESET, usbclaim(), and usbunclaim().

Referenced by _resetusb().

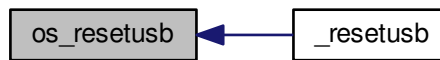
```

480                                     {
481     TEST_RESET(usbunclaim(kb, 1));
482     TEST_RESET(ioctl(kb->handle - 1, USBDEVFS_RESET));
483     TEST_RESET(usbclaim(kb));
484     // Success!
485     return 0;
486 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



5.38.3.9 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 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 endpoint?)	9 = SEND?	specific	0	1	500	struct* kb->ileds

The ioctl command is USBDEVFS_CONTROL.

Definition at line 212 of file usb_linux.c.

References ckb_err, usbdevice::handle, and usbdevice::ileds.

Referenced by updateindicators_kb().

```

212     {
213         struct usbdevfs_ctrltransfer transfer = { 0x21, 0x09, 0x0200, 0x00, 1, 500, &kb->
ileds };
214         int res = ioctl(kb->handle - 1, USBDEVFS_CONTROL, &transfer);
215         if(res <= 0)
216             ckb_err("%s\n", res ? strerror(errno) : "No data written");
217     }
  
```

Here is the caller graph for this function:



5.38.3.10 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 518 of file usb_linux.c.

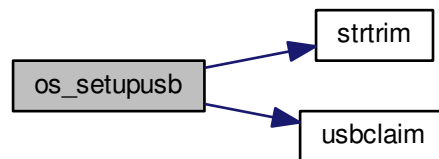
References [ckb_err](#), [ckb_info](#), [ckb_warn](#), [devpath](#), [usbdevice::epcount](#), [FEAT_RGB](#), [usbdevice::fwversion](#), [HAS_FEATURES](#), [INDEX_OF](#), [KB_NAME_LEN](#), [keyboard](#), [usbdevice::name](#), [usbdevice::serial](#), [SERIAL_LEN](#), [strtrim\(\)](#), [usbdevice::udev](#), and [usbclaim\(\)](#).

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

```

518                                     {
521     struct udev_device* dev = kb->udev;
522     const char* name = udev_device_get_sysattr_value(dev, "product");
523     if(name)
524         strncpy(kb->name, name, KB_NAME_LEN);
525     strtrim(kb->name);
526     const char* serial = udev_device_get_sysattr_value(dev, "serial");
527     if(serial)
528         strncpy(kb->serial, serial, SERIAL_LEN);
529     strtrim(kb->serial);
532     const char* firmware = udev_device_get_sysattr_value(dev, "bcdDevice");
533     if(firmware)
534         sscanf(firmware, "%hx", &kb->fwversion);
535     else
536         kb->fwversion = 0;
537     int index = INDEX_OF(kb, keyboard);
540     ckb_info("Connecting %s at %s%d\n", kb->name, devpath, index);
541
547     const char* ep_str = udev_device_get_sysattr_value(dev, "bNumInterfaces");
548     kb->epcount = 0;
549     if(ep_str)
550         sscanf(ep_str, "%d", &kb->epcount);
551     if(kb->epcount == 0){
552         // This shouldn't happen, but if it does, assume EP count based on what the device is supposed to
have
553         kb->epcount = (HAS_FEATURES(kb, FEAT_RGB) ? 4 : 3);
554         ckb_warn("Unable to read endpoint count from udev, assuming %d...\n", kb->
epcount);
555     }
556     if(usbclaim(kb)){
557         ckb_err("Failed to claim interfaces: %s\n", strerror(errno));
558         return -1;
559     }
560     return 0;
561 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



5.38.3.11 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_usbSEND ()).

So all the receiving is done via an ioctl() like in os_usbSEND. 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 (ETIMEOUT), `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->
epcount - 1, MSG_SIZE, 5000, in_msg };
140         res = ioctl(kb->handle - 1, USBDEVFS_CONTROL, &transfer);
141     /*}
142     if(res <= 0){
143         ckb_err_fn("%s\n", file, line, res ? strerror(errno) : "No data read");
144         if(res == -1 && errno == ETIMEDOUT)
145             return -1;
146         else
147             return 0;
148     } else if(res != MSG_SIZE)
149         ckb_warn_fn("Read %d bytes (expected %d)\n", file, line, res,
MSG_SIZE);
150 #ifdef DEBUG_USB_RECV
151     char converted[MSG_SIZE*3 + 1];
152     for(int i=0;i<MSG_SIZE;i++)
153         sprintf(&converted[i*3], "%02x ", in_msg[i]);
154     ckb_warn_fn("Recv %s\n", file, line, converted);
155 #endif
156     return res;
157 }
```

Here is the caller graph for this function:



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

Parameters

<i>kb</i>	THE usbdevice*
<i>out_msg</i>	the MSGSIZE char long buffer to send
<i>is_rcv</i>	if true, just send an ioctl for further reading packets. If false, send the data at <i>out_msg</i> .
<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_usbsend has two functions:

- if *is_rcv* == 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_rcv* = true).

For a more recent firmware and *is_rcv* = 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.

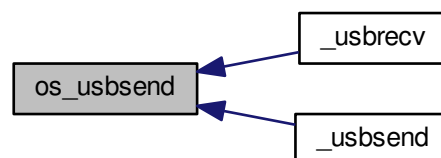
Referenced by _usbrecv(), and _usbSEND().

```

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_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 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_P-RO`, `P_STRAFE`, and `P_STRAFE_NRGB`.

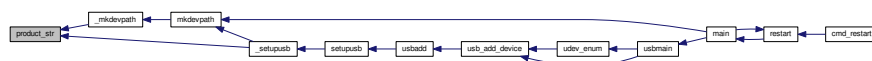
Referenced by `_mkdevpath()`, and `_setupusb()`.

```

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 `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 407 of file `usb.c`.

References `FEAT_RGB`, `HAS_FEATURES`, `NEEDS_FW_UPDATE`, `NK95_HWON`, `nk95cmd`, and `setactive`.

Referenced by `quitWithLock()`.

```

407     {
408         if (NEEDS_FW_UPDATE(kb))
409             return 0;
410         if (!HAS_FEATURES(kb, FEAT_RGB)) {
411             nk95cmd(kb, NK95_HWON);
412             return 0;
413         }
414         if (setactive(kb, 0))
415             return -1;
416         return 0;
417     }

```

Here is the caller graph for this function:



5.38.3.15 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>OUT</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 386 of file usb.c.

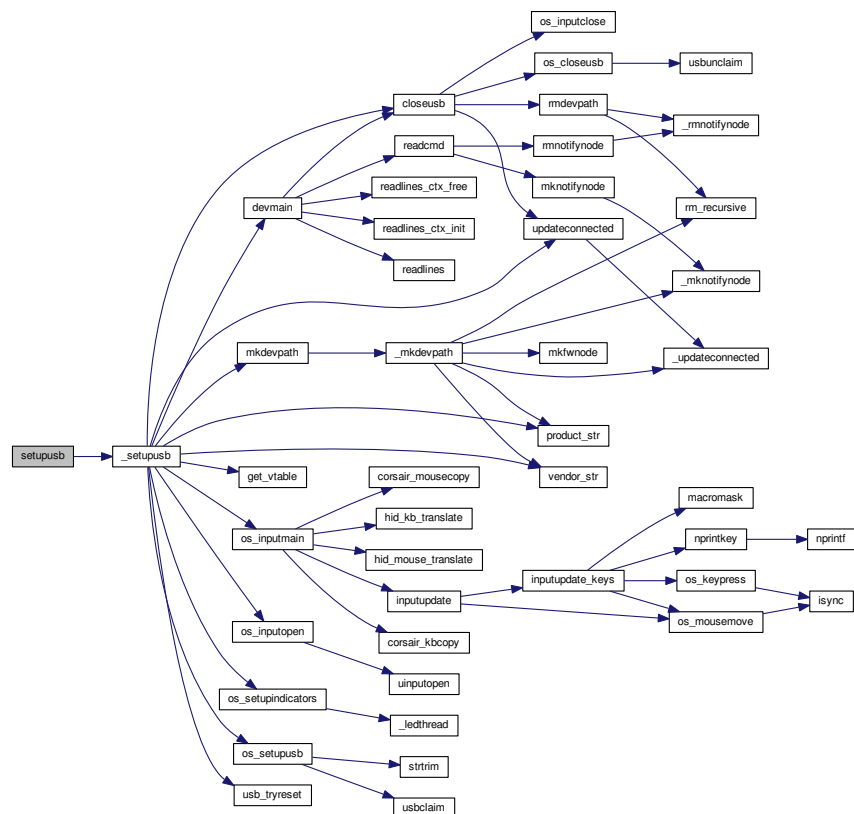
References `_setupusb()`, `ckb_err`, `imutex`, and `usbdevice::thread`.

Referenced by `usbadd()`.

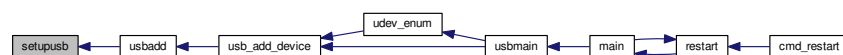
```

386 {
387     pthread_mutex_lock(imutex(kb));
388     if(pthread_create(&kb->thread, 0, _setupusb, kb))
389         ckb_err("Failed to create USB thread\n");
390 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



5.38.3.16 int usb_tryreset (usbdevice * kb)

Parameters

in, out	<i>kb</i>	THE usbdevice*
in	<i>reset_stop</i>	global variable is read

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 465 of file usb.c.

References [ckb_err](#), [ckb_info](#), [reset_stop](#), and [resetusb](#).

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

```

465                                     {
466     if(reset_stop)
467         return -1;
468     ckb_info("Attempting reset...\n");
469     while(1){
470         int res = resetusb(kb);
471         if(!res){
472             ckb_info("Reset success\n");
473             return 0;
474         }
475         if(res == -2 || reset_stop)
476             break;
477     }
478     ckb_err("Reset failed. Disconnecting.\n");
479     return -1;
480 }
```

Here is the caller graph for this function:



5.38.3.17 void usbskill ()

Definition at line 803 of file usb_linux.c.

Referenced by quitWithLock().

```

803     {
804         udev_unref(udev);
805         udev = 0;
806     }
  
```

Here is the caller graph for this function:



5.38.3.18 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 743 of file usb_linux.c.

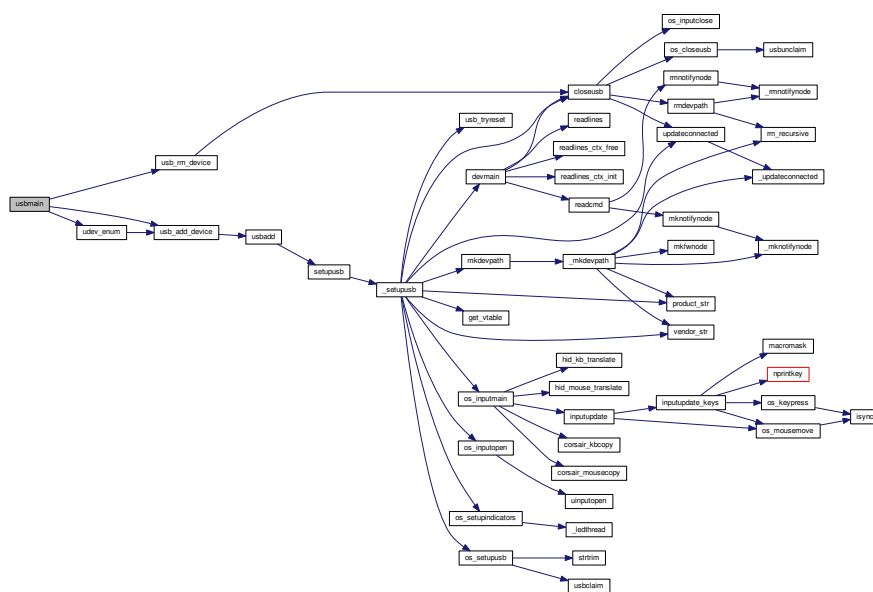
References [ckb_fatal](#), [ckb_warn](#), [udev_enum\(\)](#), [usb_add_device\(\)](#), and [usb_rm_device\(\)](#).

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

```

743     {
744         // Load the uinput module (if it's not loaded already)
745         if(system("modprobe uinput") != 0)
  
```


Here is the call graph for this function:



Here is the caller graph for this function:



5.38.3.19 const char* vendor_str (short vendor)

vendor_str Vendor/product string representations

Parameters

<i>vendor</i>	<i>short vendor ID</i>
---------------	------------------------

Returns

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.

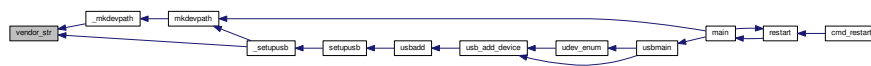
References *V_CORSAIR*.

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 src/ckb-daemon/usb_linux.c File Reference

```

#include "device.h"
#include "devnode.h"
#include "input.h"
#include "notify.h"
#include "usb.h"
```

- struct_model

- #define TEST_RESET(op)

$$TEST_RESET \text{ does a "try / catch" for resetting the usb interface.}$$
- #define N_MODELS (sizeof(models) / sizeof(_model))

- 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. Returns 0 if device was recognized/added.
- 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` ()
Stop the USB system.

Variables

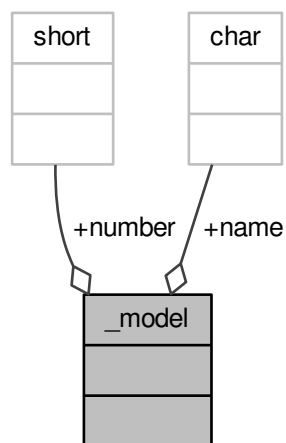
- static char [kbsyspath](#) [9][FILENAME_MAX]
all open usb devices have their system path names here in this array.
- static struct udev * [udev](#)
- pthread_t [usbthread](#)
struct udev is defined in /usr/include/libudev.h
- pthread_t [udevthread](#)
- static [_model](#) [models](#) []

5.39.1 Data Structure Documentation

5.39.1.1 struct _model

Definition at line 612 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 #define N_MODELS (sizeof(models) / sizeof(_model))

Definition at line 650 of file usb_linux.c.

Referenced by `usb_add_device()`.

5.39.2.2 #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 462 of file usb_linux.c.

Referenced by os_resetusb().

5.39.3 Function Documentation

5.39.3.1 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 187 of file `usb_linux.c`.

References `ckb_err_fn`, `usbdevice::handle`, `P_K95_NRGB`, and `usbdevice::product`.

```

187                                     {
188     if(kb->product != P_K95_NRGB)
189         return 0;
190     struct usbdevfs_ctrltransfer transfer = { 0x40, bRequest, wValue, 0, 0, 5000, 0 };
191     int res = ioctl(kb->handle - 1, USBDEVFS_CONTROL, &transfer);
192     if(res <= 0){
193         ckb_err_fn("%s\n", file, line, res ? strerror(errno) : "No data written");
194         return 1;
195     }
196     return 0;
197 }
```

5.39.3.2 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 422 of file `usb_linux.c`.

References `usbdevice::handle`, `INDEX_OF`, `kbsyspath`, `keyboard`, `usbdevice::udev`, and [usbunclaim\(\)](#).

Referenced by `closeusb()`.

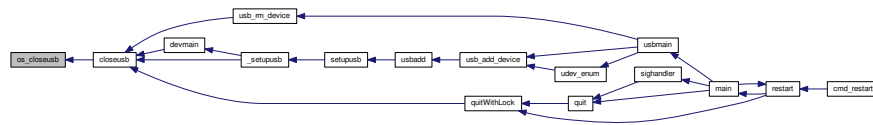
```

422                                     {
423     if(kb->handle){
424         usbunclaim(kb, 0);
425         close(kb->handle - 1);
426     }
427     if(kb->udev)
428         udev_device_unref(kb->udev);
429     kb->handle = 0;
430     kb->udev = 0;
431     kbsyspath[INDEX_OF(kb, keyboard)][0] = 0;
432 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



5.39.3.3 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.

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

1. set up an URB (Userspace Ressource Buffer) to communicate with the USBDEVFS_* ioctl()s
2. perform the ioctl()
3. interpretate the information got into the URB buffer or handle error situations and retry operation or leave the endless loop
4. inform the os about the data
5. loop endless via 2.
6. if endless loop has gone, deinitalize the interface, free buffers etc.
7. return null

Here the actions in detail:

Monitor input transfers on all endpoints for non-RGB devices For RGB, monitor all but the last, as it's used for input/output

Get an `usbdevfs_urb` data structure and clear it via `memset()`

Hopefully the buffer lengths are equal for all devices with congruent types. You can find out the correctness for your device with `lsusb -v` or similar on macOS. Currently the following combinations are known and implemented:

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 232 of file `usb_linux.c`.

References `usbdevice::active`, `ckb_info`, `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()`.

```

232                                     {
233     usbdevice* kb = context;
234     int fd = kb->handle - 1;
235     short vendor = kb->vendor, product = kb->product;
236     int index = INDEX_OF(kb, keyboard);
237     ckb_info("Starting input thread for %s%d\n", devpath, index);
238
239     int urbcount = IS_RGB(vendor, product) ? (kb->epcount - 1) : kb->
240     epcount;
241
242     struct usbdevfs_urb urbs[urbcount];
243     memset(urbs, 0, sizeof(urbs));
244
245     urbs[0].buffer_length = 8;
246     if(IS_RGB(vendor, product)){
247         if(IS_MOUSE(vendor, product))
248             urbs[1].buffer_length = 10;
249         else
250             urbs[1].buffer_length = 21;
251         urbs[2].buffer_length = MSG_SIZE;
252         if(urbcount != 3)
253             urbs[urbcount - 1].buffer_length = MSG_SIZE;
254     } else {
255         urbs[1].buffer_length = 4;
256         urbs[2].buffer_length = 15;
257     }
258
259     for(int i = 0; i < urbcount; i++){
260         urbs[i].type = USBDEVFS_URB_TYPE_INTERRUPT;
261         urbs[i].endpoint = 0x80 | (i + 1);
262         urbs[i].buffer = malloc(urbs[i].buffer_length);
263         ioctl(fd, USBDEVFS_SUBMITURB, urbs + i);
264     }
265
266     while (1) {
267         struct usbdevfs_urb* urb = 0;
268
269         if (ioctl(fd, USBDEVFS_REAPURB, &urb)){
270             if (errno == ENODEV || errno == ENOENT || errno == ESHUTDOWN)
271                 // Stop the thread if the handle closes

```

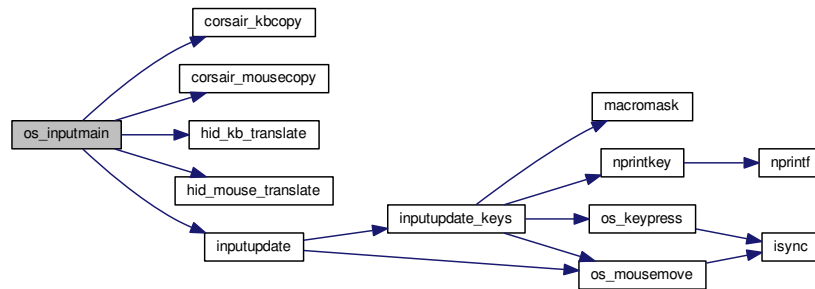


```

294         break;
295     else if(errno == EPIPE && urb){
296         ioctl(fd, USBDEVFS_CLEAR_HALT, &urb->endpoint);
297         // Re-submit the URB
298         if(urb)
299             ioctl(fd, USBDEVFS_SUBMITURB, urb);
300         urb = 0;
301     }
302 }
303
304 if (urb) {
305     pthread_mutex_lock(&mutex(kb));
306     if(IS_MOUSE(vendor, product)){
307         switch(urb->actual_length){
308             case 8:
309             case 10:
310             case 11:
311                 // HID mouse input
312                 hid_mouse_translate(kb->input.keys, &kb->
313 input.rel_x, &kb->input.rel_y, -(urb->endpoint & 0xF), urb->actual_length, urb->buffer)
314 ;
315                 break;
316             case MSG_SIZE:
317                 // Corsair mouse input
318                 corsair_mousecopy(kb->input.keys, -(urb->endpoint & 0xF), urb
319 ->buffer);
320                 break;
321         }
322     } else if(IS_RGB(vendor, product)){
323         switch(urb->actual_length){
324             case 8:
325                 // RGB EP 1: 6KRO (BIOS mode) input
326                 hid_kb_translate(kb->input.keys, -1, urb->actual_length, urb->
327 buffer);
328                 break;
329             case 21:
330             case 5:
331                 // RGB EP 2: NKRO (non-BIOS) input. Accept only if keyboard is inactive
332                 if(!kb->active)
333                     hid_kb_translate(kb->input.keys, -2, urb->actual_length,
334 urb->buffer);
335                 break;
336             case MSG_SIZE:
337                 // RGB EP 3: Corsair input
338                 corsair_kbcopy(kb->input.keys, -(urb->endpoint & 0xF), urb->
339 buffer);
340                 break;
341         }
342     } else {
343         // Non-RGB input
344         hid_kb_translate(kb->input.keys, urb->endpoint & 0xF, urb->
345 actual_length, urb->buffer);
346     }
347     inputupdate(kb);
348     pthread_mutex_unlock(&mutex(kb));
349     ioctl(fd, USBDEVFS_SUBMITURB, urb);
350     urb = 0;
351 }
352 }
353
354 ckb_info("Stopping input thread for %s%d\n", devpath, index);
355 for(int i = 0; i < urbcount; i++){
356     ioctl(fd, USBDEVFS_DISCARDURB, urbs + i);
357     free(urbs[i].buffer);
358 }
359 return 0;
360 }

```

Here is the call graph for this function:



Here is the caller graph for this function:



5.39.3.4 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 480 of file usb_linux.c.

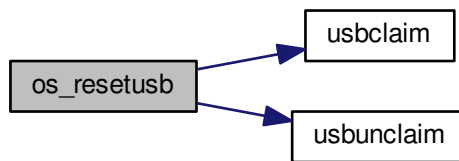
References usbdevice::handle, TEST_RESET, usbclaim(), and usbunclaim().

Referenced by _resetusb().

```

480                                     {
481     TEST_RESET(usbunclaim(kb, 1));
482     TEST_RESET(ioctl(kb->handle - 1, USBDEVFS_RESET));
483     TEST_RESET(usbclaim(kb));
484     // Success!
485     return 0;
486 }
  
```

Here is the call graph for this function:



Here is the caller graph for this function:



5.39.3.5 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 endpoint?)	9 = SEND?	specific	0	1	500	struct* kb->ileds

The ioctl command is USBDEVFS_CONTROL.

Definition at line 212 of file usb_linux.c.

References ckb_err, usbdevice::handle, and usbdevice::ileds.

Referenced by updateindicators_kb().

```

212                                     {
213     struct usbdevfs_ctrltransfer transfer = { 0x21, 0x09, 0x0200, 0x00, 1, 500, &kb->
ileds };
214     int res = ioctl(kb->handle - 1, USBDEVFS_CONTROL, &transfer);
215     if(res <= 0)
216         ckb_err("%s\n", res ? strerror(errno) : "No data written");
217 }
  
```

Here is the caller graph for this function:



5.39.3.6 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 518 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\(\)](#).

```

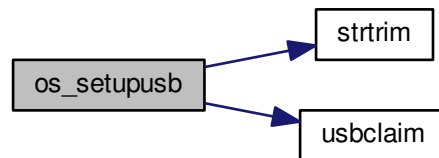
518     {
521     struct udev_device* dev = kb->udev;
522     const char* name = udev_device_get_sysattr_value(dev, "product");
523     if(name)
524         strncpy(kb->name, name, KB_NAME_LEN);
525     strtrim(kb->name);
526     const char* serial = udev_device_get_sysattr_value(dev, "serial");
527     if(serial)
528         strncpy(kb->serial, serial, SERIAL_LEN);
529     strtrim(kb->serial);
532     const char* firmware = udev_device_get_sysattr_value(dev, "bcdDevice");
533     if(firmware)
534         sscanf(firmware, "%hx", &kb->fwversion);
535     else
536         kb->fwversion = 0;
537     int index = INDEX_OF(kb, keyboard);
540     ckb_info("Connecting %s at %s%d\n", kb->name, devpath, index);
541
547     const char* ep_str = udev_device_get_sysattr_value(dev, "bNumInterfaces");
548     kb->epcount = 0;
549     if(ep_str)
550         sscanf(ep_str, "%d", &kb->epcount);
551     if(kb->epcount == 0){
552         // This shouldn't happen, but if it does, assume EP count based on what the device is supposed to
553         have
554         kb->epcount = (HAS_FEATURES(kb, FEAT_RGB) ? 4 : 3);
555         ckb_warn("Unable to read endpoint count from udev, assuming %d...\n", kb->
556         epcount);
555     }
556     if(usbclaim(kb)) {
  
```

```

557         ckb_err("Failed to claim interfaces: %s\n", strerror(errno));
558         return -1;
559     }
560     return 0;
561 }

```

Here is the call graph for this function:



Here is the caller graph for this function:



5.39.3.7 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, 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 (ETIMEOUT), [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 #ifdef DEBUG_USB_RECV
153     char converted[MSG_SIZE*3 + 1];
154     for(int i=0;i<MSG_SIZE;i++)
155         sprintf(&converted[i*3], "%02x ", in_msg[i]);
156     ckb_warn_fn("Recv %s\n", file, line, converted);
157 #endif
158     return res;
159 }

```

Here is the caller graph for this function:



5.39.3.8 int os_usbseend (usbdevice * kb, const uchar * out_msg, int is_rcv, const char * file, int line)

os_usbseend has two functions:

- if is_rcv == 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 (ETIMEOUT), [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 [_usbseend\(\)](#).

```

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;

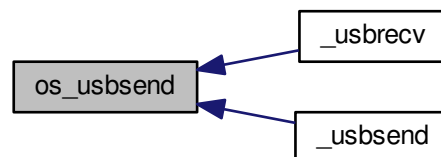
```

```

75     res = ioctl(kb->handle - 1, USBDEVFS_BULK, &transfer);
76 } else {
77     struct usbdevfs_ctrltransfer transfer = { 0x21, 0x09, 0x0200, kb->
epcount - 1, MSG_SIZE, 5000, (void*)out_msg };
78     res = ioctl(kb->handle - 1, USBDEVFS_CONTROL, &transfer);
79 }
80 if(res <= 0){
81     ckb_err_fn("%s\n", file, line, res ? strerror(errno) : "No data written");
82     if(res == -1 && errno == ETIMEDOUT)
83         return -1;
84     else
85         return 0;
86 } else if(res != MSG_SIZE)
87     ckb_warn_fn("Wrote %d bytes (expected %d)\n", file, line, res,
MSG_SIZE);
88 #ifdef DEBUG_USB
89     char converted[MSG_SIZE*3 + 1];
90     for(int i=0;i<MSG_SIZE;i++)
91         sprintf(&converted[i*3], "%02x ", out_msg[i]);
92     ckb_warn_fn("Sent %s\n", file, line, converted);
93 #endif
94     return res;
95 }

```

Here is the caller graph for this function:



5.39.3.9 void strtrim (char * *string*)

strtrim trims a string by removing leading and trailing spaces.

Parameters

<i>string</i>	
---------------	--

Definition at line 493 of file usb_linux.c.

Referenced by os_setupusb().

```

493     {
494         // Find last non-space
495         char* last = string;
496         for(char* c = string; *c != 0; c++){
497             if(!isspace(*c))
498                 last = c;
499         }
500         last[1] = 0;
501         // Find first non-space
502         char* first = string;
503         for(; *first != 0; first++){
504             if(!isspace(*first))
505                 break;
506         }
507         if(first != string)
508             memmove(string, first, last - first);
509     }

```


Here is the caller graph for this function:



5.39.3.10 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 715 of file `usb_linux.c`.

References `usb_add_device()`, and `V_CORSAIR_STR`.

Referenced by `usbmain()`.

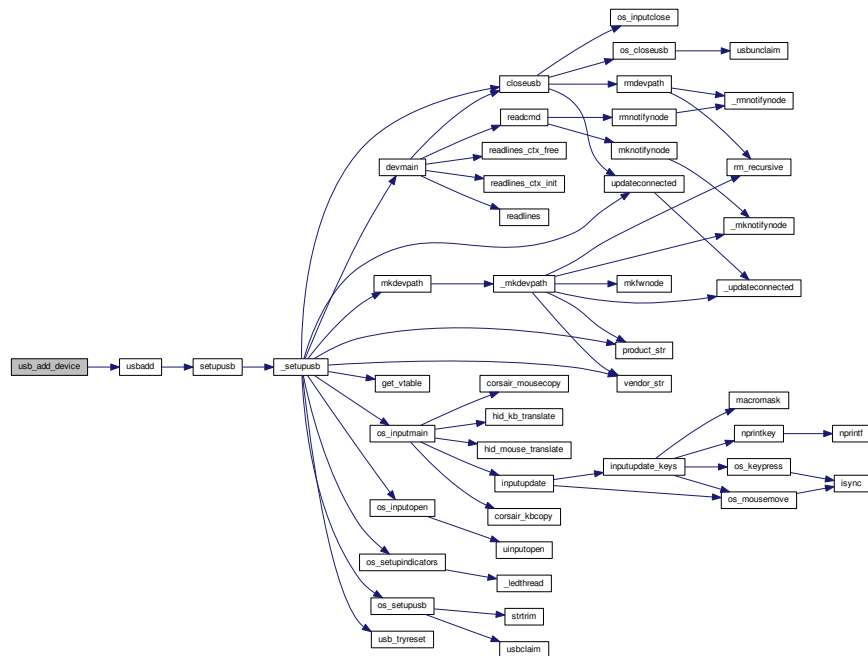
```

715         {
716     struct udev_enumerate* enumerator = udev_enumerate_new(udev);
717     udev_enumerate_add_match_subsystem(enumerator, "usb");
718     udev_enumerate_add_match_sysattr(enumerator, "idVendor", V_CORSAIR_STR);
719     udev_enumerate_scan_devices(enumerator);
720     struct udev_list_entry* devices, *dev_list_entry;
721     devices = udev_enumerate_get_list_entry(enumerator);
722
723     udev_list_entry_foreach(dev_list_entry, devices){
724         const char* path = udev_list_entry_get_name(dev_list_entry);
725         if(!path)
726             continue;
727         struct udev_device* dev = udev_device_new_from_syspath(udev, path);
728         if(!dev)
729             continue;
730         // If the device matches a recognized device ID, open it
731         if(usb_add_device(dev))
732             // Release device if not
733             udev_device_unref(dev);
734     }
735     udev_enumerate_unref(enumerator);
736 }

```


Referenced by `udev_enum()`, and `usbmain()`.

Here is the call graph for this function:



```

graph LR
    cmd_restart --> restart
    restart --> main
    main --> usbmain
    usbmain --> udev_enum
    udev_enum --> usb_add_device
    usbmain --> usb_add_device

```

Generated on Thu Apr 27 2017 21:48:46 for ckb-next by Doxygen

Parameters

<i>dev</i>	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 688 of file `usb_linux.c`.

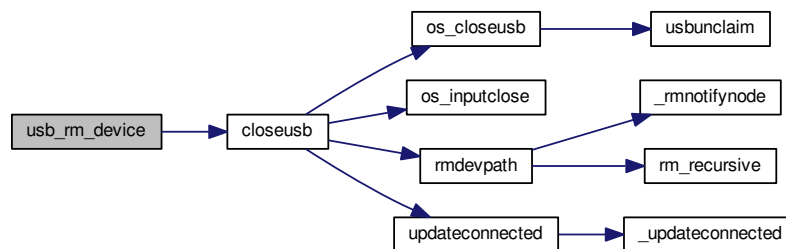
References `closeusb()`, `DEV_MAX`, `devmutex`, `kbsyspath`, and `keyboard`.

Referenced by `usbmain()`.

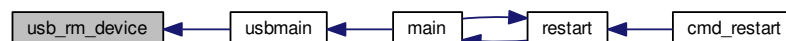
```

688                                     {
689     // Device removed. Look for it in our list of keyboards
690     const char* syspath = udev_device_get_syspath(dev);
691     if(!syspath || syspath[0] == 0)
692         return;
693     for(int i = 1; i < DEV_MAX; i++){
694         pthread_mutex_lock(devmutex + i);
695         if(!strcmp(syspath, kbsyspath[i]))
696             closeusb(keyboard + i);
697         pthread_mutex_unlock(devmutex + i);
698     }
699 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



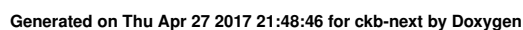
5.39.3.13 int usbadd (struct udev_device * dev, short vendor, short product)

Definition at line 563 of file `usb_linux.c`.

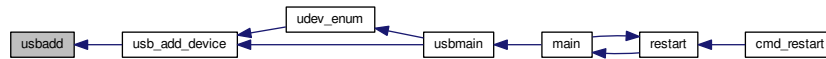
References `ckb_err`, `DEV_MAX`, `dmutex`, `usbdevice::handle`, `IS_CONNECTED`, `kbsyspath`, `keyboard`, `usbdevice::product`, `setupusb()`, `usbdevice::udev`, and `usbdevice::vendor`.

Referenced by `usb_add_device()`.

Here is the call graph for this function:



Here is the caller graph for this function:



5.39.3.14 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 (remeber 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 446 of file [usb_linux.c](#).

References [usbdevice::epcount](#), and [usbdevice::handle](#).

Referenced by [os_resetusb\(\)](#), and [os_setupusb\(\)](#).

```

446                                     {
447     int count = kb->epcount;
448     for (int i = 0; i < count; i++) {
449         struct usbdevfs_ioctl ctl = { i, USBDEVFS_DISCONNECT, 0 };
450         ioctl(kb->handle - 1, USBDEVFS_IOCTL, &ctl);
451     }
452     if (ioctl(kb->handle - 1, USBDEVFS_CLAIMINTERFACE, &i)){
453         return -1;
454     }
455 }
456 return 0;
457 }

```

Here is the caller graph for this function:



5.39.3.15 void usbkill ()

Definition at line 803 of file [usb_linux.c](#).

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

```

803                                     {
804     udev_unref(udev);
805     udev = 0;
806 }

```

Here is the caller graph for this function:



5.39.3.16 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 743 of file `usb_linux.c`.

References `ckb_fatal`, `ckb_warn`, `udev_enum()`, `usb_add_device()`, and `usb_rm_device()`.

Referenced by `main()`.

```

743     {
744     // Load the uinput module (if it's not loaded already)
745     if(system("modprobe uinput") != 0)
746         ckb_warn("Failed to load uinput module\n");
747
748     if(!(udev = udev_new())) {
749         ckb_fatal("Failed to initialize udev in usbmain(), usb_linux.c\n");
750         return -1;
751     }
752
753     udev_enum();
754
755     // Done scanning. Enter a loop to poll for device updates
756     struct udev_monitor* monitor = udev_monitor_new_from_netlink(udev, "udev");
757     udev_monitor_filter_add_match_subsystem_devtype(monitor, "usb", 0);
758     udev_monitor_enable_receiving(monitor);
759     // Get an fd for the monitor
760     int fd = udev_monitor_get_fd(monitor);
761     fd_set fds;
762     while(udev){
763         FD_ZERO(&fds);
764         FD_SET(fd, &fds);
765         // Block until an event is read
766         if(select(fd + 1, &fds, 0, 0, 0) > 0 && FD_ISSET(fd, &fds)){
767             struct udev_device* dev = udev_monitor_receive_device(monitor);
768             if(!dev)
769                 continue;
770             const char* action = udev_device_get_action(dev);
771             if(!action){
772                 udev_device_unref(dev);
773                 continue;
774             }
775             // Add/remove device

```


Parameters

<i>kb</i>	THE usbdevice*
<i>resetting</i>	boolean flag: If ressetting is true, the caller will perform a bus reset command after unclaiming the device.

Returns

always 0.

Unclaim all endpoints for a given device (remember the decrementing of the file descriptor) via `ioctl(USBDEVFS_DISCARDURB)`.

Afterwards - if resetting is false - do a USBDEVFS_CONNECT for EP 0 and 1. If it is a non RGB device, connect EP 2 also. The comment mentions RGB keyboards only, but as I understand the code, this is valid also for RGB mice.

There is no error handling yet. Function is called in `usb_linux.c` only, so it is declared as static now.

Definition at line 393 of file usb linux.c.

References usbdevice::epcount, FEAT_RGB, usbdevice::handle, and HAS_FEATURES.

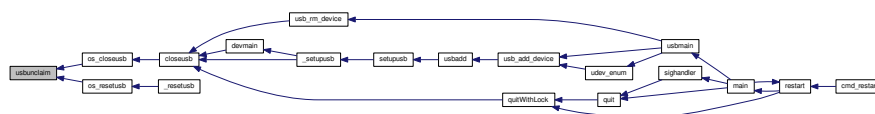
Referenced by `os_closeusb()`, and `os_resetusb()`.

```

393     {
394         int handle = kb->handle - 1;
395         int count = kb->epcount;
396         for (int i = 0; i < count; i++) {
397             ioctl(handle, USBDEVFS_RELEASEINTERFACE, &i);
398         }
399         // For RGB keyboards, the kernel driver should only be reconnected to interfaces 0 and 1 (HID), and
only if we're not about to do a USB reset.
        // Reconnecting any of the others causes trouble.
400         if (!resetting) {
401             struct usbdevfs_ioctl ctl = { 0, USBDEVFS_CONNECT, 0 };
402             ioctl(handle, USBDEVFS_IOCTL, &ctl);
403             ctl.ifno = 1;
404             ioctl(handle, USBDEVFS_IOCTL, &ctl);
405             // Also reconnect iface #2 (HID) for non-RGB keyboards
406             if (!HAS_FEATURES(kb, FEAT_RGB)){
407                 ctl.ifno = 2;
408                 ioctl(handle, USBDEVFS_IOCTL, &ctl);
409             }
410         }
411     }
412     return 0;
413 }

```

Here is the caller graph for this function:



5.39.4 Variable Documentation

5.39.4.1 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 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 has 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 622 of file *usb_linux.c*.

5.39.4.3 struct udev* udev [static]

Definition at line 606 of file *usb_linux.c*.

5.39.4.4 pthread_t udevthread

Definition at line 609 of file *usb_linux.c*.

5.39.4.5 pthread_t usbthread

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

Definition at line 609 of file *usb_linux.c*.

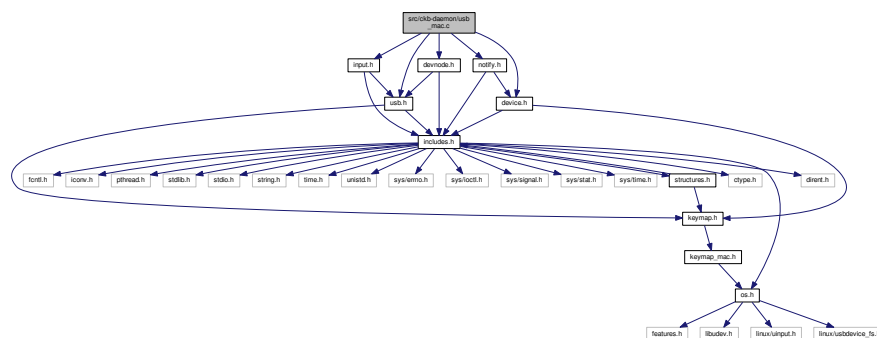
5.40 src/ckb-daemon/usb_mac.c File Reference

```

#include "device.h"
#include "devnode.h"
#include "input.h"
#include "notify.h"
#include "usb.h"

```

Include dependency graph for usb_mac.c:



Index

- [_model](#), [318](#)
- [_readlines_ctx](#), [48](#)
- [_DEFAULT_SOURCE](#)
 - [os.h](#), [203](#)
- [_GNU_SOURCE](#)
 - [os.h](#), [203](#)
- [__FILE_NOPATH__](#)
 - [includes.h](#), [87](#)
- [_cmd_get](#)
 - [notify.c](#), [191](#)
- [_cmd_macro](#)
 - [input.c](#), [90](#)
- [_freeprofile](#)
 - [profile.c](#), [205](#)
- [_ledthread](#)
 - [input_linux.c](#), [108](#)
- [_mkdevpath](#)
 - [devnode.c](#), [49](#)
- [_mknotifynode](#)
 - [devnode.c](#), [51](#)
- [_nk95cmd](#)
 - [usb.h](#), [289](#)
 - [usb_linux.c](#), [319](#)
- [_resetusb](#)
 - [usb.c](#), [260](#)
 - [usb.h](#), [291](#)
- [_rmnotifynode](#)
 - [devnode.c](#), [52](#)
- [_setupusb](#)
 - [usb.c](#), [260](#)
- [_start_dev](#)
 - [device.c](#), [26](#)
- [_updateconnected](#)
 - [devnode.c](#), [52](#)
- [_usbrecv](#)
 - [usb.c](#), [264](#)
 - [usb.h](#), [291](#)
- [_usbseend](#)
 - [usb.c](#), [266](#)
 - [usb.h](#), [293](#)
- [ACCEL](#)
 - [command.h](#), [19](#)
- [ACTIVE](#)
 - [command.h](#), [19](#)
- [ACT_LIGHT](#)
 - [device.h](#), [30](#)
- [ACT_LOCK](#)
 - [device.h](#), [30](#)
- [ACT_M1](#)
 - [device.h](#), [30](#)
- [ACT_M2](#)
 - [device.h](#), [30](#)
- [ACT_M3](#)
 - [device.h](#), [30](#)
- [ACT_MR_RING](#)
 - [device.h](#), [30](#)
- [ACT_NEXT](#)
 - [device.h](#), [30](#)
- [ACT_NEXT_NOWRAP](#)
 - [device.h](#), [30](#)
- [active](#)
 - [devcmd.__unnamed__](#), [8](#)
- [allocprofile](#)
 - [devcmd.__unnamed__](#), [8](#)
 - [profile.c](#), [205](#)
 - [profile.h](#), [221](#)
- [BIND](#)
 - [command.h](#), [20](#)
- [BR1](#)
 - [led_keyboard.c](#), [168](#)
- [BR2](#)
 - [led_keyboard.c](#), [168](#)
- [BR4](#)
 - [led_keyboard.c](#), [168](#)
- [BTN_WHEELDOWN](#)
 - [keymap.h](#), [121](#)
- [BTN_WHEELUP](#)
 - [keymap.h](#), [121](#)
- [BUTTON_HID_COUNT](#)
 - [keymap.c](#), [115](#)
- [bind](#)
 - [devcmd.__unnamed__](#), [8](#)
- [binding](#), [245](#)
- [bit_reverse_table](#)
 - [led_keyboard.c](#), [175](#)
- [CMD_FIRST](#)
 - [command.h](#), [19](#)
- [CMD_LAST](#)
 - [command.h](#), [20](#)
- [CMD_VT_FIRST](#)
 - [command.h](#), [19](#)
- [CLEAR_KEYBIT](#)
 - [structures.h](#), [254](#)
- [CMD_COUNT](#)
 - [command.h](#), [18](#)
- [CMD_DEV_COUNT](#)
 - [command.h](#), [18](#)

- ckb_err
 - includes.h, [87](#)
- ckb_err_fn
 - includes.h, [87](#)
- ckb_err_nofile
 - includes.h, [87](#)
- ckb_fatal
 - includes.h, [87](#)
- ckb_fatal_fn
 - includes.h, [87](#)
- ckb_fatal_nofile
 - includes.h, [87](#)
- ckb_info
 - includes.h, [87](#)
- ckb_info_fn
 - includes.h, [87](#)
- ckb_info_nofile
 - includes.h, [88](#)
- ckb_s_err
 - includes.h, [88](#)
- ckb_s_out
 - includes.h, [88](#)
- ckb_warn
 - includes.h, [88](#)
- ckb_warn_fn
 - includes.h, [88](#)
- ckb_warn_nofile
 - includes.h, [88](#)
- closeusb
 - usb.c, [267](#)
 - usb.h, [295](#)
- cmd
 - command.h, [19](#)
- cmd_active_kb
 - device.h, [31](#)
 - device_keyboard.c, [38](#)
- cmd_active_mouse
 - device.h, [32](#)
 - device_mouse.c, [42](#)
- cmd_bind
 - input.c, [92](#)
 - input.h, [100](#)
- cmd_dpi
 - dpi.c, [68](#)
 - dpi.h, [73](#)
- cmd_dpisel
 - dpi.c, [69](#)
 - dpi.h, [74](#)
- cmd_erase
 - profile.c, [206](#)
 - profile.h, [222](#)
- cmd_eraseprofile
 - profile.c, [207](#)
 - profile.h, [223](#)
- cmd_fwupdate
 - firmware.c, [79](#)
 - firmware.h, [83](#)
- cmd_get
 - notify.c, [193](#)
 - notify.h, [199](#)
- cmd_hwload_kb
 - profile.h, [223](#)
 - profile_keyboard.c, [235](#)
- cmd_hwload_mouse
 - profile.h, [224](#)
 - profile_mouse.c, [238](#)
- cmd_hwsave_kb
 - profile.h, [225](#)
 - profile_keyboard.c, [236](#)
- cmd_hwsave_mouse
 - profile.h, [226](#)
 - profile_mouse.c, [239](#)
- cmd_iauto
 - led.c, [129](#)
 - led.h, [134](#)
- cmd_id
 - profile.c, [207](#)
 - profile.h, [227](#)
- cmd_idle_kb
 - device.h, [32](#)
 - device_keyboard.c, [38](#)
- cmd_idle_mouse
 - device.h, [32](#)
 - device_mouse.c, [42](#)
- cmd_inotify
 - led.c, [129](#)
 - led.h, [135](#)
- cmd_io_none
 - device_vtable.c, [45](#)
- cmd_ioff
 - led.c, [129](#)
 - led.h, [135](#)
- cmd_ion
 - led.c, [130](#)
 - led.h, [136](#)
- cmd_lift
 - dpi.c, [69](#)
 - dpi.h, [74](#)
- cmd_macro
 - input.c, [92](#)
 - input.h, [100](#)
- cmd_macro_none
 - device_vtable.c, [45](#)
- cmd_name
 - profile.c, [208](#)
 - profile.h, [227](#)
- cmd_none
 - device_vtable.c, [45](#)
- cmd_notify
 - notify.c, [194](#)
 - notify.h, [199](#)
- cmd_pollrate
 - device.h, [33](#)
 - device_mouse.c, [43](#)
- cmd_profileid
 - profile.c, [208](#)

- profile.h, 228
- cmd_profilename
 - profile.c, 209
 - profile.h, 228
- cmd_rebind
 - input.c, 93
 - input.h, 100
- cmd_restart
 - notify.c, 194
 - notify.h, 200
- cmd_rgb
 - led.c, 130
 - led.h, 136
- cmd_snap
 - dpi.c, 69
 - dpi.h, 74
- cmd_strings
 - command.c, 16
- cmd_unbind
 - input.c, 93
 - input.h, 101
- cmdhandler
 - command.h, 18
- cmdhandler_io
 - command.h, 19
- cmdhandler_mac
 - command.h, 19
- command.h
 - ACCEL, 19
 - ACTIVE, 19
 - BIND, 20
 - CMD_FIRST, 19
 - CMD_LAST, 20
 - CMD_VT_FIRST, 19
 - DELAY, 19
 - DITHER, 19
 - DPI, 20
 - DPISEL, 20
 - ERASE, 19
 - ERASEPROFILE, 19
 - FPS, 19
 - FWUPDATE, 19
 - GET, 20
 - HWLOAD, 19
 - HWSAVE, 19
 - IAUTO, 20
 - ID, 19
 - IDLE, 19
 - INOTIFY, 20
 - IOFF, 19
 - ION, 19
 - LAYOUT, 19
 - LIFT, 20
 - MACRO, 20
 - MODE, 19
 - NAME, 19
 - NONE, 19
 - NOTIFY, 20
 - NOTIFYOFF, 19
 - NOTIFYON, 19
 - POLLRATE, 19
 - PROFILEID, 19
 - PROFILENAME, 19
 - REBIND, 20
 - RESTART, 20
 - RGB, 19
 - SCROLLSPEED, 19
 - SNAP, 20
 - SWITCH, 19
 - UNBIND, 20
- command.c
 - cmd_strings, 16
 - readcmd, 12
 - TRY_WITH_RESET, 11
- command.h
 - CMD_COUNT, 18
 - CMD_DEV_COUNT, 18
 - cmd, 19
 - cmdhandler, 18
 - cmdhandler_io, 19
 - cmdhandler_mac, 19
 - devcmd, 19
 - readcmd, 21
- corsair_kbcopy
 - keymap.c, 115
 - keymap.h, 124
- corsair_mousecopy
 - keymap.c, 116
 - keymap.h, 124
- DELAY
 - command.h, 19
- DITHER
 - command.h, 19
- DPI
 - command.h, 20
- DPISEL
 - command.h, 20
- DELAY_LONG
 - usb.h, 281
- DELAY_MEDIUM
 - usb.h, 281
- DELAY_SHORT
 - usb.h, 281
- DEV_MAX
 - device.h, 30
- DPI_COUNT
 - structures.h, 254
- devcmd, 17
 - command.h, 19
- devcmd.__unnamed__, 7
 - active, 8
 - allocprofile, 8
 - bind, 8
 - dpi, 8
 - dpisel, 8
 - erase, 8

- eraseprofile, 8
- freeprofile, 8
- fwupdate, 8
- get, 8
- hwload, 8
- hwsave, 8
- iauto, 9
- id, 9
- idle, 9
- inotify, 9
- ioff, 9
- ion, 9
- lift, 9
- loadprofile, 9
- macro, 9
- name, 9
- notify, 9
- pollrate, 9
- profileid, 9
- profilename, 9
- rebind, 9
- restart, 9
- rgb, 9
- setmodeindex, 9
- snap, 9
- start, 9
- unbind, 9
- updatedpi, 9
- updateindicators, 9
- updatergb, 9
- device.c
 - _start_dev, 26
 - devlistmutex, 28
 - devmutex, 28
 - hwload_mode, 28
 - inputmutex, 28
 - keyboard, 28
 - start_dev, 27
- device.h
 - ACT_LIGHT, 30
 - ACT_LOCK, 30
 - ACT_M1, 30
 - ACT_M2, 30
 - ACT_M3, 30
 - ACT_MR_RING, 30
 - ACT_NEXT, 30
 - ACT_NEXT_NOWRAP, 30
 - cmd_active_kb, 31
 - cmd_active_mouse, 32
 - cmd_idle_kb, 32
 - cmd_idle_mouse, 32
 - cmd_pollrate, 33
 - DEV_MAX, 30
 - devmutex, 37
 - dmutex, 31
 - IN_CORSAIR, 31
 - IN_HID, 31
 - IS_CONNECTED, 31
 - imutex, 31
 - inputmutex, 37
 - keyboard, 37
 - setactive, 31
 - setactive_kb, 33
 - setactive_mouse, 35
 - setmodeindex_nrgb, 36
 - start_dev, 36
 - start_kb_nrgb, 37
- device_keyboard.c
 - cmd_active_kb, 38
 - cmd_idle_kb, 38
 - setactive_kb, 39
 - setmodeindex_nrgb, 41
 - start_kb_nrgb, 41
- device_mouse.c
 - cmd_active_mouse, 42
 - cmd_idle_mouse, 42
 - cmd_pollrate, 43
 - setactive_mouse, 43
- device_vtable.c
 - cmd_io_none, 45
 - cmd_macro_none, 45
 - cmd_none, 45
 - int1_int_none, 46
 - int1_void_none, 46
 - loadprofile_none, 46
 - vtable_keyboard, 46
 - vtable_keyboard_nonrgb, 46
 - vtable_mouse, 46
- devlistmutex
 - device.c, 28
- devmain
 - usb.c, 269
- devmutex
 - device.c, 28
 - device.h, 37
- devnode.c
 - _mkdevpath, 49
 - _mknotifynode, 51
 - _rmnotifynode, 52
 - _updateconnected, 52
 - devpath, 60
 - gid, 60
 - MAX_BUFFER, 48
 - mkdevpath, 53
 - mkfwnode, 54
 - mknotifynode, 55
 - readlines, 55
 - readlines_ctx_free, 56
 - readlines_ctx_init, 57
 - rm_recursive, 57
 - rmdevpath, 57
 - rmnotifynode, 58
 - S_GID_READ, 49
 - updateconnected, 59
- devnode.h
 - devpath, 68

- gid, 68
- mkdevpath, 62
- mkfwnode, 62
- mknotifynode, 63
- readlines, 64
- readlines_ctx, 62
- readlines_ctx_free, 65
- readlines_ctx_init, 65
- rmdevpath, 65
- rmnotifynode, 66
- S_CUSTOM, 61
- S_CUSTOM_R, 61
- S_READ, 61
- S_READDIR, 61
- S_READWRITE, 61
- updateconnected, 67
- devpath
 - devnode.c, 60
 - devnode.h, 68
- dmutex
 - device.h, 31
- dpi
 - devcmd.__unnamed__, 8
- dpi.c
 - cmd_dpi, 68
 - cmd_dpisel, 69
 - cmd_lift, 69
 - cmd_snap, 69
 - loaddpi, 70
 - printdpi, 71
 - savedpi, 71
 - updatedpi, 72
- dpi.h
 - cmd_dpi, 73
 - cmd_dpisel, 74
 - cmd_lift, 74
 - cmd_snap, 74
 - loaddpi, 75
 - printdpi, 76
 - savedpi, 76
 - updatedpi, 77
- dpisel
 - devcmd.__unnamed__, 8
- dpiset, 246
- ERASE
 - command.h, 19
- ERASEPROFILE
 - command.h, 19
- erase
 - devcmd.__unnamed__, 8
- eraseprofile
 - devcmd.__unnamed__, 8
- euid_guard_start
 - os.h, 203
- euid_guard_stop
 - os.h, 203
- FPS
 - command.h, 19
- FWUPDATE
 - command.h, 19
- FEAT_ADJRATE
 - structures.h, 254
- FEAT_ANSI
 - structures.h, 254
- FEAT_BIND
 - structures.h, 254
- FEAT_COMMON
 - structures.h, 254
- FEAT_FWUPDATE
 - structures.h, 254
- FEAT_FWVERSION
 - structures.h, 255
- FEAT_HWLOAD
 - structures.h, 255
- FEAT_ISO
 - structures.h, 255
- FEAT_LMASK
 - structures.h, 255
- FEAT_MONOCHROME
 - structures.h, 255
- FEAT_MOUSEACCEL
 - structures.h, 255
- FEAT_NOTIFY
 - structures.h, 255
- FEAT_POLLRATE
 - structures.h, 255
- FEAT_RGB
 - structures.h, 255
- FEAT_STD_NRGB
 - structures.h, 255
- FEAT_STD_RGB
 - structures.h, 256
- FW_MAXSIZE
 - firmware.c, 79
- FW_NOFILE
 - firmware.c, 79
- FW_OK
 - firmware.c, 79
- FW_USBFAIL
 - firmware.c, 79
- FW_WRONGDEV
 - firmware.c, 79
- features_mask
 - main.c, 189
 - usb.c, 276
- firmware.c
 - cmd_fwupdate, 79
 - FW_MAXSIZE, 79
 - FW_NOFILE, 79
 - FW_OK, 79
 - FW_USBFAIL, 79
 - FW_WRONGDEV, 79
 - fwupdate, 80
 - getfwversion, 82
- firmware.h

- cmd_fwupdate, [83](#)
- getfwversion, [84](#)
- freebind
 - input.c, [93](#)
 - input.h, [101](#)
- freemode
 - profile.c, [209](#)
- freeprofile
 - devcmd.__unnamed__, [8](#)
 - profile.c, [210](#)
 - profile.h, [229](#)
- fwupdate
 - devcmd.__unnamed__, [8](#)
 - firmware.c, [80](#)
- GET
 - command.h, [20](#)
- get
 - devcmd.__unnamed__, [8](#)
- get_vtable
 - usb.c, [271](#)
- getfwversion
 - firmware.c, [82](#)
 - firmware.h, [84](#)
- gethwmodename
 - profile.c, [210](#)
 - profile.h, [229](#)
- gethwprofilename
 - profile.c, [211](#)
 - profile.h, [230](#)
- getid
 - profile.c, [212](#)
 - profile.h, [231](#)
- getmodename
 - profile.c, [212](#)
 - profile.h, [231](#)
- getprofilename
 - profile.c, [213](#)
 - profile.h, [232](#)
- gid
 - devnode.c, [60](#)
 - devnode.h, [68](#)
- HWLOAD
 - command.h, [19](#)
- HWSAVE
 - command.h, [19](#)
- HAS_ANY_FEATURE
 - structures.h, [256](#)
- HAS_FEATURES
 - structures.h, [256](#)
- HW_STANDARD
 - notify.c, [190](#)
- HWMODE_K70
 - structures.h, [256](#)
- HWMODE_K95
 - structures.h, [256](#)
- HWMODE_MAX
 - structures.h, [256](#)
- HWMODE_OR_RETURN
 - notify.c, [190](#)
- has_key
 - led.c, [131](#)
- hid_kb_translate
 - keymap.c, [116](#)
 - keymap.h, [124](#)
- hid_mouse_translate
 - keymap.c, [118](#)
 - keymap.h, [126](#)
- hwload
 - devcmd.__unnamed__, [8](#)
- hwload_mode
 - device.c, [28](#)
 - main.c, [189](#)
 - usb.c, [276](#)
- hwloadmode
 - profile_keyboard.c, [237](#)
- hwloadprofile
 - profile.h, [221](#)
- hwprofile, [250](#)
- hwsave
 - devcmd.__unnamed__, [8](#)
- hwtonative
 - profile.c, [214](#)
 - profile.h, [233](#)
- IAUTO
 - command.h, [20](#)
- ID
 - command.h, [19](#)
- IDLE
 - command.h, [19](#)
- INOTIFY
 - command.h, [20](#)
- IOFF
 - command.h, [19](#)
- ION
 - command.h, [19](#)
- I_CAPS
 - structures.h, [256](#)
- I_NUM
 - structures.h, [256](#)
- I_SCROLL
 - structures.h, [256](#)
- IFACE_MAX
 - structures.h, [256](#)
- IN_CORSAIR
 - device.h, [31](#)
- IN_HID
 - device.h, [31](#)
- INDEX_OF
 - includes.h, [88](#)
- IS_CONNECTED
 - device.h, [31](#)
- IS_FULLRANGE
 - usb.h, [281](#)
- IS_K65
 - usb.h, [281](#)

- IS_K70
 - usb.h, [282](#)
- IS_K95
 - usb.h, [282](#)
- IS_M65
 - usb.h, [282](#)
- IS_MOD
 - input.h, [99](#)
- IS_MONOCHROME
 - usb.h, [282](#)
- IS_MONOCHROME_DEV
 - usb.h, [282](#)
- IS_MOUSE
 - usb.h, [282](#)
- IS_MOUSE_DEV
 - usb.h, [282](#)
- IS_RGB
 - usb.h, [282](#)
- IS_RGB_DEV
 - usb.h, [283](#)
- IS_SABRE
 - usb.h, [283](#)
- IS_SCIMITAR
 - usb.h, [283](#)
- IS_STRAFE
 - usb.h, [283](#)
- IS_WHEEL
 - input.c, [90](#)
- iauto
 - devcmd.__unnamed__, [9](#)
- id
 - devcmd.__unnamed__, [9](#)
- idle
 - devcmd.__unnamed__, [9](#)
- imutex
 - device.h, [31](#)
- includes.h
 - __FILE_NOPATH__, [87](#)
 - ckb_err, [87](#)
 - ckb_err_fn, [87](#)
 - ckb_err_nofile, [87](#)
 - ckb_fatal, [87](#)
 - ckb_fatal_fn, [87](#)
 - ckb_fatal_nofile, [87](#)
 - ckb_info, [87](#)
 - ckb_info_fn, [87](#)
 - ckb_info_nofile, [88](#)
 - ckb_s_err, [88](#)
 - ckb_s_out, [88](#)
 - ckb_warn, [88](#)
 - ckb_warn_fn, [88](#)
 - ckb_warn_nofile, [88](#)
 - INDEX_OF, [88](#)
 - timespec_add, [89](#)
 - timespec_eq, [88](#)
 - timespec_ge, [88](#)
 - timespec_gt, [89](#)
 - timespec_le, [89](#)
 - timespec_lt, [89](#)
 - uchar, [89](#)
 - ushort, [89](#)
- initbind
 - input.c, [94](#)
 - input.h, [101](#)
- initmode
 - profile.c, [214](#)
- inotify
 - devcmd.__unnamed__, [9](#)
- input.c
 - _cmd_macro, [90](#)
 - cmd_bind, [92](#)
 - cmd_macro, [92](#)
 - cmd_rebind, [93](#)
 - cmd_unbind, [93](#)
 - freebind, [93](#)
 - IS_WHEEL, [90](#)
 - initbind, [94](#)
 - inputupdate, [94](#)
 - inputupdate_keys, [95](#)
 - macromask, [97](#)
 - updateindicators_kb, [98](#)
- input.h
 - cmd_bind, [100](#)
 - cmd_macro, [100](#)
 - cmd_rebind, [100](#)
 - cmd_unbind, [101](#)
 - freebind, [101](#)
 - IS_MOD, [99](#)
 - initbind, [101](#)
 - inputupdate, [102](#)
 - os_inputclose, [103](#)
 - os_inputopen, [103](#)
 - os_keypress, [104](#)
 - os_mousemove, [105](#)
 - os_setupindicators, [106](#)
 - updateindicators_kb, [107](#)
- input_linux.c
 - _ledthread, [108](#)
 - isync, [109](#)
 - os_inputclose, [109](#)
 - os_inputopen, [110](#)
 - os_keypress, [111](#)
 - os_mousemove, [111](#)
 - os_setupindicators, [112](#)
 - uinputopen, [113](#)
- inputmutex
 - device.c, [28](#)
 - device.h, [37](#)
- inputupdate
 - input.c, [94](#)
 - input.h, [102](#)
- inputupdate_keys
 - input.c, [95](#)
- int1_int_none
 - device_vtable.c, [46](#)
- int1_void_none

- device_vtable.c, 46
- ioff
 - devcmd.__unnamed__, 9
- ion
 - devcmd.__unnamed__, 9
- isblack
 - led_mouse.c, 177
- iselect
 - led.c, 131
- isync
 - input_linux.c, 109
- KB_NAME_LEN
 - structures.h, 257
- KEY_BACKSLASH_ISO
 - keymap.h, 121
- KEY_CORSAIR
 - keymap.h, 121
- KEY_NONE
 - keymap.h, 121
- KEY_UNBOUND
 - keymap.h, 122
- kbsyspath
 - usb_linux.c, 339
- key, 120
- keyboard
 - device.c, 28
 - device.h, 37
- keymacro, 244
- keymap
 - keymap.c, 119
 - keymap.h, 127
- keymap.c
 - BUTTON_HID_COUNT, 115
 - corsair_kbcopy, 115
 - corsair_mousecopy, 116
 - hid_kb_translate, 116
 - hid_mouse_translate, 118
 - keymap, 119
- keymap.h
 - BTN_WHEELDOWN, 121
 - BTN_WHEELUP, 121
 - corsair_kbcopy, 124
 - corsair_mousecopy, 124
 - hid_kb_translate, 124
 - hid_mouse_translate, 126
 - KEY_BACKSLASH_ISO, 121
 - KEY_CORSAIR, 121
 - KEY_NONE, 121
 - KEY_UNBOUND, 122
 - keymap, 127
 - LED_DPI, 122
 - LED_MOUSE, 122
 - MOUSE_BUTTON_FIRST, 122
 - MOUSE_EXTRA_FIRST, 122
 - N_BUTTONS_EXTENDED, 122
 - N_BUTTONS_HW, 122
 - N_KEY_ZONES, 122
 - N_KEYBYTES_EXTENDED, 122
 - N_KEYBYTES_HW, 122
 - N_KEYBYTES_INPUT, 123
 - N_KEYS_EXTENDED, 123
 - N_KEYS_EXTRA, 123
 - N_KEYS_HW, 123
 - N_KEYS_INPUT, 123
 - N_MOUSE_ZONES, 123
 - SCAN_KBD, 123
 - SCAN_MOUSE, 123
 - SCAN_SILENT, 123
- LAYOUT
 - command.h, 19
- LIFT
 - command.h, 20
- LED_DPI
 - keymap.h, 122
- LED_MOUSE
 - keymap.h, 122
- LIFT_MAX
 - structures.h, 257
- LIFT_MIN
 - structures.h, 257
- led.c
 - cmd_iauto, 129
 - cmd_inotify, 129
 - cmd_ioff, 129
 - cmd_ion, 130
 - cmd_rgb, 130
 - has_key, 131
 - iselect, 131
 - printrgb, 132
- led.h
 - cmd_iauto, 134
 - cmd_inotify, 135
 - cmd_ioff, 135
 - cmd_ion, 136
 - cmd_rgb, 136
 - loadrgb_kb, 136
 - loadrgb_mouse, 138
 - printrgb, 139
 - savergb_kb, 140
 - savergb_mouse, 142
 - updatergb_kb, 142
 - updatergb_mouse, 144
- led_keyboard.c
 - BR1, 168
 - BR2, 168
 - BR4, 168
 - bit_reverse_table, 175
 - loadrgb_kb, 169
 - makergb_512, 170
 - makergb_full, 171
 - O0, 168
 - O1, 168
 - O2, 168
 - O3, 168
 - O4, 168
 - O5, 168

- O6, [168](#)
- O7, [168](#)
- O8, [168](#)
- ordered8to3, [172](#)
- quantize8to3, [172](#)
- rgbcmp, [172](#)
- savergb_kb, [173](#)
- updatergb_kb, [174](#)
- led_mouse.c
 - isblack, [177](#)
 - loadrgb_mouse, [178](#)
 - rgbcmp, [178](#)
 - savergb_mouse, [179](#)
 - updatergb_mouse, [179](#)
- lift
 - devcmd.__unnamed__, [9](#)
- lighting, [247](#)
- loaddpi
 - dpi.c, [70](#)
 - dpi.h, [75](#)
- loadprofile
 - devcmd.__unnamed__, [9](#)
 - profile.c, [215](#)
 - profile.h, [233](#)
- loadprofile_none
 - device_vtable.c, [46](#)
- loadrgb_kb
 - led.h, [136](#)
 - led_keyboard.c, [169](#)
- loadrgb_mouse
 - led.h, [138](#)
 - led_mouse.c, [178](#)
- localecase
 - main.c, [181](#)
- MACRO
 - command.h, [20](#)
- MODE
 - command.h, [19](#)
- MACRO_MAX
 - structures.h, [257](#)
- MAX_BUFFER
 - devnode.c, [48](#)
- MD_NAME_LEN
 - structures.h, [257](#)
- MODE_COUNT
 - structures.h, [257](#)
- MOUSE_BUTTON_FIRST
 - keymap.h, [122](#)
- MOUSE_EXTRA_FIRST
 - keymap.h, [122](#)
- MSG_SIZE
 - structures.h, [257](#)
- macro
 - devcmd.__unnamed__, [9](#)
- macroaction, [243](#)
- macromask
 - input.c, [97](#)
- main
 - main.c, [182](#)
- main.c
 - features_mask, [189](#)
 - hwload_mode, [189](#)
 - localecase, [181](#)
 - main, [182](#)
 - main_ac, [189](#)
 - main_av, [189](#)
 - quit, [184](#)
 - quitWithLock, [185](#)
 - reset_stop, [189](#)
 - restart, [186](#)
 - sighandler, [187](#)
 - sighandler2, [188](#)
 - timespec_add, [188](#)
- main_ac
 - main.c, [189](#)
- main_av
 - main.c, [189](#)
- makergb_512
 - led_keyboard.c, [170](#)
- makergb_full
 - led_keyboard.c, [171](#)
- mkdevpath
 - devnode.c, [53](#)
 - devnode.h, [62](#)
- mkfwnode
 - devnode.c, [54](#)
 - devnode.h, [62](#)
- mknotifynode
 - devnode.c, [55](#)
 - devnode.h, [63](#)
- models
 - usb_linux.c, [339](#)
- NAME
 - command.h, [19](#)
- NONE
 - command.h, [19](#)
- NOTIFY
 - command.h, [20](#)
- NOTIFYOFF
 - command.h, [19](#)
- NOTIFYON
 - command.h, [19](#)
- N_BUTTONS_EXTENDED
 - keymap.h, [122](#)
- N_BUTTONS_HW
 - keymap.h, [122](#)
- N_KEY_ZONES
 - keymap.h, [122](#)
- N_KEYBYTES_EXTENDED
 - keymap.h, [122](#)
- N_KEYBYTES_HW
 - keymap.h, [122](#)
- N_KEYBYTES_INPUT
 - keymap.h, [123](#)
- N_KEYS_EXTENDED
 - keymap.h, [123](#)

- N_KEYS_EXTRA
 - keymap.h, [123](#)
- N_KEYS_HW
 - keymap.h, [123](#)
- N_KEYS_INPUT
 - keymap.h, [123](#)
- N_MODELS
 - usb_linux.c, [318](#)
- N_MOUSE_ZONES
 - keymap.h, [123](#)
- NEEDS_FW_UPDATE
 - structures.h, [257](#)
- NK95_HWOFF
 - usb.h, [283](#)
- NK95_HWON
 - usb.h, [283](#)
- NK95_M1
 - usb.h, [283](#)
- NK95_M2
 - usb.h, [283](#)
- NK95_M3
 - usb.h, [283](#)
- name
 - devcmd.__unnamed__, [9](#)
- nativetohw
 - profile.c, [215](#)
 - profile.h, [234](#)
- nk95cmd
 - usb.h, [284](#)
- notify
 - devcmd.__unnamed__, [9](#)
- notify.c
 - _cmd_get, [191](#)
 - cmd_get, [193](#)
 - cmd_notify, [194](#)
 - cmd_restart, [194](#)
 - HW_STANDARD, [190](#)
 - HWMODE_OR_RETURN, [190](#)
 - nprintf, [195](#)
 - nprintind, [195](#)
 - nprintkey, [196](#)
 - restart, [197](#)
- notify.h
 - cmd_get, [199](#)
 - cmd_notify, [199](#)
 - cmd_restart, [200](#)
 - nprintf, [200](#)
 - nprintind, [201](#)
 - nprintkey, [202](#)
- nprintf
 - notify.c, [195](#)
 - notify.h, [200](#)
- nprintind
 - notify.c, [195](#)
 - notify.h, [201](#)
- nprintkey
 - notify.c, [196](#)
 - notify.h, [202](#)
- O0
 - led_keyboard.c, [168](#)
- O1
 - led_keyboard.c, [168](#)
- O2
 - led_keyboard.c, [168](#)
- O3
 - led_keyboard.c, [168](#)
- O4
 - led_keyboard.c, [168](#)
- O5
 - led_keyboard.c, [168](#)
- O6
 - led_keyboard.c, [168](#)
- O7
 - led_keyboard.c, [168](#)
- O8
 - led_keyboard.c, [168](#)
- OUTFIFO_MAX
 - structures.h, [257](#)
- ordered8to3
 - led_keyboard.c, [172](#)
- os.h
 - _DEFAULT_SOURCE, [203](#)
 - _GNU_SOURCE, [203](#)
 - euid_guard_start, [203](#)
 - euid_guard_stop, [203](#)
 - UINPUT_VERSION, [203](#)
- os_closeusb
 - usb.h, [296](#)
 - usb_linux.c, [320](#)
- os_inputclose
 - input.h, [103](#)
 - input_linux.c, [109](#)
- os_inputmain
 - usb.h, [297](#)
 - usb_linux.c, [321](#)
- os_inputopen
 - input.h, [103](#)
 - input_linux.c, [110](#)
- os_keypress
 - input.h, [104](#)
 - input_linux.c, [111](#)
- os_mousemove
 - input.h, [105](#)
 - input_linux.c, [111](#)
- os_resetusb
 - usb.h, [301](#)
 - usb_linux.c, [324](#)
- os_sendindicators
 - usb.h, [302](#)
 - usb_linux.c, [325](#)
- os_setupindicators
 - input.h, [106](#)
 - input_linux.c, [112](#)
- os_setupusb
 - usb.h, [302](#)
 - usb_linux.c, [326](#)

- os_usbrecv
 - usb.h, [304](#)
 - usb_linux.c, [327](#)
- os_usbsend
 - usb.h, [305](#)
 - usb_linux.c, [328](#)
- POLLRATE
 - command.h, [19](#)
- PROFILEID
 - command.h, [19](#)
- PROFILENAME
 - command.h, [19](#)
- P_HARPOON
 - usb.h, [284](#)
- P_HARPOON_STR
 - usb.h, [284](#)
- P_K65
 - usb.h, [284](#)
- P_K65_LUX
 - usb.h, [284](#)
- P_K65_LUX_STR
 - usb.h, [284](#)
- P_K65_NRGB
 - usb.h, [284](#)
- P_K65_NRGB_STR
 - usb.h, [284](#)
- P_K65_RFIRE
 - usb.h, [284](#)
- P_K65_RFIRE_STR
 - usb.h, [284](#)
- P_K65_STR
 - usb.h, [285](#)
- P_K70
 - usb.h, [285](#)
- P_K70_LUX
 - usb.h, [285](#)
- P_K70_LUX_NRGB
 - usb.h, [285](#)
- P_K70_LUX_NRGB_STR
 - usb.h, [285](#)
- P_K70_LUX_STR
 - usb.h, [285](#)
- P_K70_NRGB
 - usb.h, [285](#)
- P_K70_NRGB_STR
 - usb.h, [285](#)
- P_K70_RFIRE
 - usb.h, [285](#)
- P_K70_RFIRE_NRGB
 - usb.h, [285](#)
- P_K70_RFIRE_NRGB_STR
 - usb.h, [286](#)
- P_K70_RFIRE_STR
 - usb.h, [286](#)
- P_K70_STR
 - usb.h, [286](#)
- P_K95
 - usb.h, [286](#)
- P_K95_NRGB
 - usb.h, [286](#)
- P_K95_NRGB_STR
 - usb.h, [286](#)
- P_K95_PLATINUM
 - usb.h, [286](#)
- P_K95_PLATINUM_STR
 - usb.h, [286](#)
- P_K95_STR
 - usb.h, [286](#)
- P_M65
 - usb.h, [286](#)
- P_M65_PRO
 - usb.h, [286](#)
- P_M65_PRO_STR
 - usb.h, [287](#)
- P_M65_STR
 - usb.h, [287](#)
- P_SABRE_L
 - usb.h, [287](#)
- P_SABRE_L_STR
 - usb.h, [287](#)
- P_SABRE_N
 - usb.h, [287](#)
- P_SABRE_N_STR
 - usb.h, [287](#)
- P_SABRE_O
 - usb.h, [287](#)
- P_SABRE_O2
 - usb.h, [287](#)
- P_SABRE_O2_STR
 - usb.h, [287](#)
- P_SABRE_O_STR
 - usb.h, [287](#)
- P_SCIMITAR
 - usb.h, [287](#)
- P_SCIMITAR_PRO
 - usb.h, [288](#)
- P_SCIMITAR_PRO_STR
 - usb.h, [288](#)
- P_SCIMITAR_STR
 - usb.h, [288](#)
- P_STRAFE
 - usb.h, [288](#)
- P_STRAFE_NRGB
 - usb.h, [288](#)
- P_STRAFE_NRGB_STR
 - usb.h, [288](#)
- P_STRAFE_STR
 - usb.h, [288](#)
- PR_NAME_LEN
 - structures.h, [258](#)
- pollrate
 - devcmd.__unnamed__, [9](#)
- printdpi
 - dpi.c, [71](#)
 - dpi.h, [76](#)
- printname

- profile.c, 216
- printrgb
 - led.c, 132
 - led.h, 139
- product_str
 - usb.c, 272
 - usb.h, 308
- profile.c
 - _freeprofile, 205
 - allocprofile, 205
 - cmd_erase, 206
 - cmd_eraseprofile, 207
 - cmd_id, 207
 - cmd_name, 208
 - cmd_profileid, 208
 - cmd_profilename, 209
 - freemode, 209
 - freeprofile, 210
 - gethwmodename, 210
 - gethwprofilename, 211
 - getid, 212
 - getmodename, 212
 - getprofilename, 213
 - hwtonative, 214
 - initmode, 214
 - loadprofile, 215
 - nativetohw, 215
 - printname, 216
 - setid, 217
 - u16dec, 217
 - u16enc, 218
 - urldecode2, 218
 - urlencode2, 219
 - utf16to8, 220
 - utf8to16, 220
- profile.h
 - allocprofile, 221
 - cmd_erase, 222
 - cmd_eraseprofile, 223
 - cmd_hwload_kb, 223
 - cmd_hwload_mouse, 224
 - cmd_hwsave_kb, 225
 - cmd_hwsave_mouse, 226
 - cmd_id, 227
 - cmd_name, 227
 - cmd_profileid, 228
 - cmd_profilename, 228
 - freeprofile, 229
 - gethwmodename, 229
 - gethwprofilename, 230
 - getid, 231
 - getmodename, 231
 - getprofilename, 232
 - hwloadprofile, 221
 - hwtonative, 233
 - loadprofile, 233
 - nativetohw, 234
 - setid, 234
- profile_keyboard.c
 - cmd_hwload_kb, 235
 - cmd_hwsave_kb, 236
 - hwloadmode, 237
- profile_mouse.c
 - cmd_hwload_mouse, 238
 - cmd_hwsave_mouse, 239
- profileid
 - devcmd.__unnamed__, 9
- profilename
 - devcmd.__unnamed__, 9
- quantize8to3
 - led_keyboard.c, 172
- quit
 - main.c, 184
- quitWithLock
 - main.c, 185
- REBIND
 - command.h, 20
- RESTART
 - command.h, 20
- RGB
 - command.h, 19
- readcmd
 - command.c, 12
 - command.h, 21
- readlines
 - devnode.c, 55
 - devnode.h, 64
- readlines_ctx
 - devnode.h, 62
- readlines_ctx_free
 - devnode.c, 56
 - devnode.h, 65
- readlines_ctx_init
 - devnode.c, 57
 - devnode.h, 65
- rebind
 - devcmd.__unnamed__, 9
- reset_stop
 - main.c, 189
 - usb.c, 276
- resetusb
 - usb.h, 288
- restart
 - devcmd.__unnamed__, 9
 - main.c, 186
 - notify.c, 197
- revertusb
 - usb.c, 273
 - usb.h, 309
- rgb
 - devcmd.__unnamed__, 9
- rgbcmp
 - led_keyboard.c, 172
 - led_mouse.c, 178
- rm_recursive

- devnode.c, 57
- rmdevpath
 - devnode.c, 57
 - devnode.h, 65
- rmnotifynode
 - devnode.c, 58
 - devnode.h, 66
- SCROLLSPEED
 - command.h, 19
- SNAP
 - command.h, 20
- SWITCH
 - command.h, 19
- S_CUSTOM
 - devnode.h, 61
- S_CUSTOM_R
 - devnode.h, 61
- S_GID_READ
 - devnode.c, 49
- S_READ
 - devnode.h, 61
- S_READDIR
 - devnode.h, 61
- S_READWRITE
 - devnode.h, 61
- SCAN_KBD
 - keymap.h, 123
- SCAN_MOUSE
 - keymap.h, 123
- SCAN_SILENT
 - keymap.h, 123
- SCROLL_ACCELERATED
 - structures.h, 258
- SCROLL_MAX
 - structures.h, 258
- SCROLL_MIN
 - structures.h, 258
- SERIAL_LEN
 - structures.h, 258
- SET_KEYBIT
 - structures.h, 258
- savedpi
 - dpi.c, 71
 - dpi.h, 76
- savergb_kb
 - led.h, 140
 - led_keyboard.c, 173
- savergb_mouse
 - led.h, 142
 - led_mouse.c, 179
- setactive
 - device.h, 31
- setactive_kb
 - device.h, 33
 - device_keyboard.c, 39
- setactive_mouse
 - device.h, 35
 - device_mouse.c, 43
- setid
 - profile.c, 217
 - profile.h, 234
- setmodeindex
 - devcmd.__unnamed__, 9
- setmodeindex_nrgb
 - device.h, 36
 - device_keyboard.c, 41
- setupusb
 - usb.c, 274
 - usb.h, 311
- sighandler
 - main.c, 187
- sighandler2
 - main.c, 188
- snap
 - devcmd.__unnamed__, 9
- src/ckb-daemon/command.c, 11
- src/ckb-daemon/command.h, 16
- src/ckb-daemon/device.c, 25
- src/ckb-daemon/device.h, 28
- src/ckb-daemon/device_keyboard.c, 38
- src/ckb-daemon/device_mouse.c, 41
- src/ckb-daemon/device_vtable.c, 44
- src/ckb-daemon/devnode.c, 47
- src/ckb-daemon/devnode.h, 60
- src/ckb-daemon/dpi.c, 68
- src/ckb-daemon/dpi.h, 73
- src/ckb-daemon/extra_mac.c, 78
- src/ckb-daemon/firmware.c, 78
- src/ckb-daemon/firmware.h, 83
- src/ckb-daemon/includes.h, 85
- src/ckb-daemon/input.c, 89
- src/ckb-daemon/input.h, 98
- src/ckb-daemon/input_linux.c, 107
- src/ckb-daemon/input_mac.c, 114
- src/ckb-daemon/input_mac_mouse.c, 114
- src/ckb-daemon/keymap.c, 114
- src/ckb-daemon/keymap.h, 119
- src/ckb-daemon/keymap_mac.h, 127
- src/ckb-daemon/led.c, 128
- src/ckb-daemon/led.h, 134
- src/ckb-daemon/led_keyboard.c, 145
- src/ckb-daemon/led_mouse.c, 177
- src/ckb-daemon/main.c, 180
- src/ckb-daemon/notify.c, 189
- src/ckb-daemon/notify.h, 198
- src/ckb-daemon/os.h, 202
- src/ckb-daemon/profile.c, 204
- src/ckb-daemon/profile.h, 220
- src/ckb-daemon/profile_keyboard.c, 235
- src/ckb-daemon/profile_mouse.c, 238
- src/ckb-daemon/structures.h, 240
- src/ckb-daemon/usb.c, 259
- src/ckb-daemon/usb.h, 277
- src/ckb-daemon/usb_linux.c, 316
- src/ckb-daemon/usb_mac.c, 340
- start

- devcmd.__unnamed__, 9
- start_dev
 - device.c, 27
 - device.h, 36
- start_kb_nrgb
 - device.h, 37
 - device_keyboard.c, 41
- strtrim
 - usb_linux.c, 330
- structures.h
 - CLEAR_KEYBIT, 254
 - DPI_COUNT, 254
 - FEAT_ADJRATE, 254
 - FEAT_ANSI, 254
 - FEAT_BIND, 254
 - FEAT_COMMON, 254
 - FEAT_FWUPDATE, 254
 - FEAT_FWVERSION, 255
 - FEAT_HWLOAD, 255
 - FEAT_ISO, 255
 - FEAT_LMASK, 255
 - FEAT_MONOCHROME, 255
 - FEAT_MOUSEACCEL, 255
 - FEAT_NOTIFY, 255
 - FEAT_POLLRATE, 255
 - FEAT_RGB, 255
 - FEAT_STD_NRGB, 255
 - FEAT_STD_RGB, 256
 - HAS_ANY_FEATURE, 256
 - HAS_FEATURES, 256
 - HWMODE_K70, 256
 - HWMODE_K95, 256
 - HWMODE_MAX, 256
 - I_CAPS, 256
 - I_NUM, 256
 - I_SCROLL, 256
 - IFACE_MAX, 256
 - KB_NAME_LEN, 257
 - LIFT_MAX, 257
 - LIFT_MIN, 257
 - MACRO_MAX, 257
 - MD_NAME_LEN, 257
 - MODE_COUNT, 257
 - MSG_SIZE, 257
 - NEEDS_FW_UPDATE, 257
 - OUTFIFO_MAX, 257
 - PR_NAME_LEN, 258
 - SCROLL_ACCELERATED, 258
 - SCROLL_MAX, 258
 - SCROLL_MIN, 258
 - SERIAL_LEN, 258
 - SET_KEYBIT, 258
 - vtable_keyboard, 258
 - vtable_keyboard_nonrgb, 258
 - vtable_mouse, 258
- TEST_RESET
 - usb_linux.c, 318
- TRY_WITH_RESET
 - command.c, 11
- timespec_add
 - includes.h, 89
 - main.c, 188
- timespec_eq
 - includes.h, 88
- timespec_ge
 - includes.h, 88
- timespec_gt
 - includes.h, 89
- timespec_le
 - includes.h, 89
- timespec_lt
 - includes.h, 89
- u16dec
 - profile.c, 217
- u16enc
 - profile.c, 218
- UNBIND
 - command.h, 20
- UINPUT_VERSION
 - os.h, 203
- USB_DELAY_DEFAULT
 - usb.h, 288
- uchar
 - includes.h, 89
- udev
 - usb_linux.c, 340
- udev_enum
 - usb_linux.c, 331
- udevthread
 - usb_linux.c, 340
- uinputopen
 - input_linux.c, 113
- unbind
 - devcmd.__unnamed__, 9
- updateconnected
 - devnode.c, 59
 - devnode.h, 67
- updatedpi
 - devcmd.__unnamed__, 9
 - dpi.c, 72
 - dpi.h, 77
- updateindicators
 - devcmd.__unnamed__, 9
- updateindicators_kb
 - input.c, 98
 - input.h, 107
- updatergb
 - devcmd.__unnamed__, 9
- updatergb_kb
 - led.h, 142
 - led_keyboard.c, 174
- updatergb_mouse
 - led.h, 144
 - led_mouse.c, 179
- urldecode2
 - profile.c, 218

- urlencode2
 - profile.c, [219](#)
- usb.c
 - _resetusb, [260](#)
 - _setupusb, [260](#)
 - _usbrecv, [264](#)
 - _usbsend, [266](#)
 - closeusb, [267](#)
 - devmain, [269](#)
 - features_mask, [276](#)
 - get_vtable, [271](#)
 - hwload_mode, [276](#)
 - product_str, [272](#)
 - reset_stop, [276](#)
 - revertusb, [273](#)
 - setupusb, [274](#)
 - usb_tryreset, [275](#)
 - usbmutex, [277](#)
 - vendor_str, [276](#)
- usb.h
 - _nk95cmd, [289](#)
 - _resetusb, [291](#)
 - _usbrecv, [291](#)
 - _usbsend, [293](#)
 - closeusb, [295](#)
 - DELAY_LONG, [281](#)
 - DELAY_MEDIUM, [281](#)
 - DELAY_SHORT, [281](#)
 - IS_FULLRANGE, [281](#)
 - IS_K65, [281](#)
 - IS_K70, [282](#)
 - IS_K95, [282](#)
 - IS_M65, [282](#)
 - IS_MONOCHROME, [282](#)
 - IS_MONOCHROME_DEV, [282](#)
 - IS_MOUSE, [282](#)
 - IS_MOUSE_DEV, [282](#)
 - IS_RGB, [282](#)
 - IS_RGB_DEV, [283](#)
 - IS_SABRE, [283](#)
 - IS_SCIMITAR, [283](#)
 - IS_STRAFE, [283](#)
 - NK95_HWOFF, [283](#)
 - NK95_HWON, [283](#)
 - NK95_M1, [283](#)
 - NK95_M2, [283](#)
 - NK95_M3, [283](#)
 - nk95cmd, [284](#)
 - os_closeusb, [296](#)
 - os_inputmain, [297](#)
 - os_resetusb, [301](#)
 - os_sendindicators, [302](#)
 - os_setupusb, [302](#)
 - os_usbrecv, [304](#)
 - os_usbsend, [305](#)
 - P_HARPOON, [284](#)
 - P_HARPOON_STR, [284](#)
 - P_K65, [284](#)
 - P_K65_LUX, [284](#)
 - P_K65_LUX_STR, [284](#)
 - P_K65_NRGB, [284](#)
 - P_K65_NRGB_STR, [284](#)
 - P_K65_RFIRE, [284](#)
 - P_K65_RFIRE_STR, [284](#)
 - P_K65_STR, [285](#)
 - P_K70, [285](#)
 - P_K70_LUX, [285](#)
 - P_K70_LUX_NRGB, [285](#)
 - P_K70_LUX_NRGB_STR, [285](#)
 - P_K70_LUX_STR, [285](#)
 - P_K70_NRGB, [285](#)
 - P_K70_NRGB_STR, [285](#)
 - P_K70_RFIRE, [285](#)
 - P_K70_RFIRE_NRGB, [285](#)
 - P_K70_RFIRE_STR, [286](#)
 - P_K70_STR, [286](#)
 - P_K95, [286](#)
 - P_K95_NRGB, [286](#)
 - P_K95_NRGB_STR, [286](#)
 - P_K95_PLATINUM, [286](#)
 - P_K95_PLATINUM_STR, [286](#)
 - P_K95_STR, [286](#)
 - P_M65, [286](#)
 - P_M65_PRO, [286](#)
 - P_M65_PRO_STR, [287](#)
 - P_M65_STR, [287](#)
 - P_SABRE_L, [287](#)
 - P_SABRE_L_STR, [287](#)
 - P_SABRE_N, [287](#)
 - P_SABRE_N_STR, [287](#)
 - P_SABRE_O, [287](#)
 - P_SABRE_O2, [287](#)
 - P_SABRE_O2_STR, [287](#)
 - P_SABRE_O_STR, [287](#)
 - P_SCIMITAR, [287](#)
 - P_SCIMITAR_PRO, [288](#)
 - P_SCIMITAR_PRO_STR, [288](#)
 - P_SCIMITAR_STR, [288](#)
 - P_STRAFE, [288](#)
 - P_STRAFE_NRGB, [288](#)
 - P_STRAFE_NRGB_STR, [288](#)
 - P_STRAFE_STR, [288](#)
 - product_str, [308](#)
 - resetusb, [288](#)
 - revertusb, [309](#)
 - setupusb, [311](#)
 - USB_DELAY_DEFAULT, [288](#)
 - usb_tryreset, [312](#)
 - usbkill, [314](#)
 - usbmain, [314](#)
 - usbrecv, [288](#)
 - usbsend, [289](#)
 - V_CORSAIR, [289](#)
 - V_CORSAIR_STR, [289](#)
 - vendor_str, [316](#)
- usb_add_device

- usb_linux.c, [332](#)
- usb_linux.c
 - _nk95cmd, [319](#)
 - kbsyspath, [339](#)
 - models, [339](#)
 - N_MODELS, [318](#)
 - os_closeusb, [320](#)
 - os_inputmain, [321](#)
 - os_resetusb, [324](#)
 - os_sendindicators, [325](#)
 - os_setupusb, [326](#)
 - os_usbrecv, [327](#)
 - os_usbsend, [328](#)
 - strtrim, [330](#)
 - TEST_RESET, [318](#)
 - udev, [340](#)
 - udev_enum, [331](#)
 - udevthread, [340](#)
 - usb_add_device, [332](#)
 - usb_rm_device, [333](#)
 - usbadd, [334](#)
 - usbclaim, [336](#)
 - usbkill, [336](#)
 - usbmain, [337](#)
 - usbthread, [340](#)
 - usbunclaim, [338](#)
- usb_rm_device
 - usb_linux.c, [333](#)
- usb_tryreset
 - usb.c, [275](#)
 - usb.h, [312](#)
- usbadd
 - usb_linux.c, [334](#)
- usbclaim
 - usb_linux.c, [336](#)
- usbdevice, [252](#)
- usbid, [242](#)
- usbinput, [251](#)
- usbkill
 - usb.h, [314](#)
 - usb_linux.c, [336](#)
- usbmain
 - usb.h, [314](#)
 - usb_linux.c, [337](#)
- usbmode, [248](#)
- usbmutex
 - usb.c, [277](#)
- usbprofile, [249](#)
- usbrecv
 - usb.h, [288](#)
- usbend
 - usb.h, [289](#)
- usbthread
 - usb_linux.c, [340](#)
- usbunclaim
 - usb_linux.c, [338](#)
- ushort
 - includes.h, [89](#)
- utf16to8
 - profile.c, [220](#)
- utf8to16
 - profile.c, [220](#)
- V_CORSAIR
 - usb.h, [289](#)
- V_CORSAIR_STR
 - usb.h, [289](#)
- vendor_str
 - usb.c, [276](#)
 - usb.h, [316](#)
- vtable_keyboard
 - device_vtable.c, [46](#)
 - structures.h, [258](#)
- vtable_keyboard_nonrgb
 - device_vtable.c, [46](#)
 - structures.h, [258](#)
- vtable_mouse
 - device_vtable.c, [46](#)
 - structures.h, [258](#)