

ckb-next

beta-v0.2.8+testing at branch macrotime.0.2

Generated by Doxygen 1.8.6

Mon Jun 5 2017 23:09:31

Contents

1	ckb-next: RGB Driver for Linux and OS X	1
2	Building ckb	9
3	cbk Improvements Roadmap	11
4	DAEMON	13
5	Todo List	21
6	Data Structure Index	23
6.1	Data Structures	23
7	File Index	25
7.1	File List	25
8	Data Structure Documentation	27
8.1	devcmd.__unnamed__ Struct Reference	27
8.1.1	Detailed Description	28
8.1.2	Field Documentation	28
8.1.2.1	active	28
8.1.2.2	allocprofile	28
8.1.2.3	bind	28
8.1.2.4	dpi	28
8.1.2.5	dpisel	28
8.1.2.6	erase	28
8.1.2.7	eraseprofile	28
8.1.2.8	freeprofile	28
8.1.2.9	fwupdate	28
8.1.2.10	get	28
8.1.2.11	hwload	28
8.1.2.12	hwsave	29
8.1.2.13	iauto	29
8.1.2.14	id	29

8.1.2.15	idle	29
8.1.2.16	inotify	29
8.1.2.17	ioff	29
8.1.2.18	ion	29
8.1.2.19	lift	29
8.1.2.20	loadprofile	29
8.1.2.21	macro	29
8.1.2.22	name	29
8.1.2.23	notify	29
8.1.2.24	pollrate	29
8.1.2.25	profileid	29
8.1.2.26	profilename	29
8.1.2.27	rebind	29
8.1.2.28	restart	29
8.1.2.29	rgb	29
8.1.2.30	setmodeindex	29
8.1.2.31	snap	29
8.1.2.32	start	29
8.1.2.33	unbind	29
8.1.2.34	updatedpi	29
8.1.2.35	updateindicators	29
8.1.2.36	updatergb	29
9	File Documentation	31
9.1	BUILD.md File Reference	31
9.2	DAEMON.md File Reference	31
9.3	README.md File Reference	31
9.4	ROADMAP.md File Reference	31
9.5	src/ckb-daemon/command.c File Reference	31
9.5.1	Macro Definition Documentation	32
9.5.1.1	TRY_WITH_RESET	32
9.5.2	Function Documentation	32
9.5.2.1	readcmd	32
9.5.3	Variable Documentation	36
9.5.3.1	cmd_strings	36
9.6	src/ckb-daemon/command.h File Reference	36
9.6.1	Data Structure Documentation	38
9.6.1.1	union devcmd	38
9.6.2	Macro Definition Documentation	38
9.6.2.1	CMD_COUNT	38

9.6.2.2	CMD_DEV_COUNT	39
9.6.3	Typedef Documentation	39
9.6.3.1	cmdhandler	39
9.6.3.2	cmdhandler_io	39
9.6.3.3	cmdhandler_mac	39
9.6.3.4	devcmd	39
9.6.4	Enumeration Type Documentation	39
9.6.4.1	cmd	39
9.6.5	Function Documentation	41
9.6.5.1	readcmd	41
9.7	src/ckb-daemon/device.c File Reference	45
9.7.1	Function Documentation	46
9.7.1.1	_start_dev	46
9.7.1.2	start_dev	47
9.7.2	Variable Documentation	48
9.7.2.1	devlistmutex	48
9.7.2.2	devmutex	48
9.7.2.3	hwload_mode	48
9.7.2.4	inputmutex	48
9.7.2.5	keyboard	48
9.7.2.6	macromutex	48
9.7.2.7	macromutex2	49
9.7.2.8	macrovar	49
9.8	src/ckb-daemon/device.h File Reference	49
9.8.1	Macro Definition Documentation	50
9.8.1.1	ACT_LIGHT	50
9.8.1.2	ACT_LOCK	50
9.8.1.3	ACT_M1	50
9.8.1.4	ACT_M2	51
9.8.1.5	ACT_M3	51
9.8.1.6	ACT_MR_RING	51
9.8.1.7	ACT_NEXT	51
9.8.1.8	ACT_NEXT_NOWRAP	51
9.8.1.9	DEV_MAX	51
9.8.1.10	dmutex	51
9.8.1.11	imutex	51
9.8.1.12	IN_CORSAIR	51
9.8.1.13	IN_HID	51
9.8.1.14	IS_CONNECTED	52
9.8.1.15	mmutex	52

9.8.1.16	mmutex2	52
9.8.1.17	mvar	52
9.8.1.18	setactive	52
9.8.2	Function Documentation	52
9.8.2.1	cmd_active_kb	52
9.8.2.2	cmd_active_mouse	53
9.8.2.3	cmd_idle_kb	53
9.8.2.4	cmd_idle_mouse	53
9.8.2.5	cmd_pollrate	54
9.8.2.6	setactive_kb	54
9.8.2.7	setactive_mouse	56
9.8.2.8	setmodeindex_nrgb	57
9.8.2.9	start_dev	58
9.8.2.10	start_kb_nrgb	58
9.8.3	Variable Documentation	58
9.8.3.1	devmutex	58
9.8.3.2	inputmutex	58
9.8.3.3	keyboard	58
9.8.3.4	macromutex	59
9.8.3.5	macromutex2	59
9.8.3.6	macrovar	59
9.9	src/ckb-daemon/device_keyboard.c File Reference	59
9.9.1	Function Documentation	59
9.9.1.1	cmd_active_kb	59
9.9.1.2	cmd_idle_kb	60
9.9.1.3	setactive_kb	60
9.9.1.4	setmodeindex_nrgb	62
9.9.1.5	start_kb_nrgb	62
9.10	src/ckb-daemon/device_mouse.c File Reference	63
9.10.1	Function Documentation	63
9.10.1.1	cmd_active_mouse	63
9.10.1.2	cmd_idle_mouse	64
9.10.1.3	cmd_pollrate	64
9.10.1.4	setactive_mouse	64
9.11	src/ckb-daemon/device_vtable.c File Reference	66
9.11.1	Function Documentation	67
9.11.1.1	cmd_io_none	67
9.11.1.2	cmd_macro_none	67
9.11.1.3	cmd_none	67
9.11.1.4	int1_int_none	67

9.11.1.5	int1_void_none	67
9.11.1.6	loadprofile_none	67
9.11.2	Variable Documentation	68
9.11.2.1	vtable_keyboard	68
9.11.2.2	vtable_keyboard_nonrgb	68
9.11.2.3	vtable_mouse	68
9.12	src/ckb-daemon/devnode.c File Reference	68
9.12.1	Data Structure Documentation	69
9.12.1.1	struct_readlines_ctx	69
9.12.2	Macro Definition Documentation	70
9.12.2.1	MAX_BUFFER	70
9.12.2.2	S_GID_READ	70
9.12.3	Function Documentation	70
9.12.3.1	_mkdevpath	70
9.12.3.2	_mknotifynode	73
9.12.3.3	_rmnotifynode	74
9.12.3.4	_updateconnected	74
9.12.3.5	mkdevpath	75
9.12.3.6	mkfwnode	76
9.12.3.7	mknotifynode	77
9.12.3.8	readlines	77
9.12.3.9	readlines_ctx_free	78
9.12.3.10	readlines_ctx_init	79
9.12.3.11	rm_recursive	79
9.12.3.12	rmdevpath	80
9.12.3.13	rmnotifynode	80
9.12.3.14	updateconnected	81
9.12.4	Variable Documentation	82
9.12.4.1	devpath	82
9.12.4.2	gid	82
9.13	src/ckb-daemon/devnode.h File Reference	82
9.13.1	Macro Definition Documentation	83
9.13.1.1	S_CUSTOM	83
9.13.1.2	S_CUSTOM_R	83
9.13.1.3	S_READ	83
9.13.1.4	S_READDIR	83
9.13.1.5	S_READWRITE	83
9.13.2	Typedef Documentation	84
9.13.2.1	readlines_ctx	84
9.13.3	Function Documentation	84

9.13.3.1	mkdevpath	84
9.13.3.2	mkfwnode	85
9.13.3.3	mknotifynode	85
9.13.3.4	readlines	86
9.13.3.5	readlines_ctx_free	87
9.13.3.6	readlines_ctx_init	87
9.13.3.7	rmdevpath	88
9.13.3.8	rmnotifynode	88
9.13.3.9	updateconnected	89
9.13.4	Variable Documentation	90
9.13.4.1	devpath	90
9.13.4.2	gid	90
9.14	src/ckb-daemon/dpi.c File Reference	90
9.14.1	Function Documentation	90
9.14.1.1	cmd_dpi	90
9.14.1.2	cmd_dpisel	91
9.14.1.3	cmd_lift	91
9.14.1.4	cmd_snap	91
9.14.1.5	loaddpi	92
9.14.1.6	printdpi	93
9.14.1.7	savedpi	93
9.14.1.8	updatedpi	94
9.15	src/ckb-daemon/dpi.h File Reference	95
9.15.1	Function Documentation	95
9.15.1.1	cmd_dpi	95
9.15.1.2	cmd_dpisel	96
9.15.1.3	cmd_lift	96
9.15.1.4	cmd_snap	97
9.15.1.5	loaddpi	97
9.15.1.6	printdpi	98
9.15.1.7	savedpi	98
9.15.1.8	updatedpi	99
9.16	src/ckb-daemon/extra_mac.c File Reference	100
9.17	src/ckb-daemon/firmware.c File Reference	100
9.17.1	Macro Definition Documentation	101
9.17.1.1	FW_MAXSIZE	101
9.17.1.2	FW_NOFILE	101
9.17.1.3	FW_OK	101
9.17.1.4	FW_USBFAIL	101
9.17.1.5	FW_WRONGDEV	101

9.17.2	Function Documentation	101
9.17.2.1	cmd_fwupdate	101
9.17.2.2	fwupdate	102
9.17.2.3	getfwversion	104
9.18	src/ckb-daemon/firmware.h File Reference	105
9.18.1	Function Documentation	105
9.18.1.1	cmd_fwupdate	106
9.18.1.2	getfwversion	106
9.19	src/ckb-daemon/includes.h File Reference	107
9.19.1	Macro Definition Documentation	109
9.19.1.1	__FILE_NOPATH__	109
9.19.1.2	ckb_err	109
9.19.1.3	ckb_err_fn	109
9.19.1.4	ckb_err_nofile	109
9.19.1.5	ckb_fatal	109
9.19.1.6	ckb_fatal_fn	109
9.19.1.7	ckb_fatal_nofile	109
9.19.1.8	ckb_info	109
9.19.1.9	ckb_info_fn	110
9.19.1.10	ckb_info_nofile	110
9.19.1.11	ckb_s_err	110
9.19.1.12	ckb_s_out	110
9.19.1.13	ckb_warn	110
9.19.1.14	ckb_warn_fn	110
9.19.1.15	ckb_warn_nofile	110
9.19.1.16	INDEX_OF	110
9.19.1.17	timespec_eq	110
9.19.1.18	timespec_ge	111
9.19.1.19	timespec_gt	111
9.19.1.20	timespec_le	111
9.19.1.21	timespec_lt	111
9.19.2	Typedef Documentation	111
9.19.2.1	uchar	111
9.19.2.2	ushort	111
9.19.3	Function Documentation	111
9.19.3.1	timespec_add	111
9.20	src/ckb-daemon/input.c File Reference	111
9.20.1	Macro Definition Documentation	113
9.20.1.1	IS_WHEEL	113
9.20.2	Function Documentation	113

9.20.2.1	_cmd_macro	113
9.20.2.2	cmd_bind	115
9.20.2.3	cmd_macro	115
9.20.2.4	cmd_rebind	115
9.20.2.5	cmd_unbind	116
9.20.2.6	freebind	116
9.20.2.7	initbind	116
9.20.2.8	inputupdate	117
9.20.2.9	inputupdate_keys	118
9.20.2.10	macro_pt_dequeue	120
9.20.2.11	macro_pt_enqueue	121
9.20.2.12	macro_pt_first	121
9.20.2.13	macromask	122
9.20.2.14	play_macro	122
9.20.2.15	updateindicators_kb	124
9.20.3	Variable Documentation	125
9.20.3.1	pt_head	125
9.20.3.2	pt_tail	125
9.21	src/ckb-daemon/input.h File Reference	125
9.21.1	Data Structure Documentation	127
9.21.1.1	struct parameter	127
9.21.1.2	struct ptlist	128
9.21.2	Macro Definition Documentation	128
9.21.2.1	IS_MOD	128
9.21.3	Typedef Documentation	128
9.21.3.1	parameter_t	128
9.21.3.2	ptlist_t	128
9.21.4	Function Documentation	128
9.21.4.1	cmd_bind	129
9.21.4.2	cmd_macro	129
9.21.4.3	cmd_rebind	129
9.21.4.4	cmd_unbind	130
9.21.4.5	freebind	130
9.21.4.6	initbind	130
9.21.4.7	inputupdate	131
9.21.4.8	os_inputclose	132
9.21.4.9	os_inputopen	132
9.21.4.10	os_keypress	133
9.21.4.11	os_mousemove	134
9.21.4.12	os_setupindicators	135

9.21.4.13	updateindicators_kb	136
9.22	src/ckb-daemon/input_linux.c File Reference	136
9.22.1	Function Documentation	137
9.22.1.1	_ledthread	137
9.22.1.2	isync	138
9.22.1.3	os_inputclose	138
9.22.1.4	os_inputopen	139
9.22.1.5	os_keypress	140
9.22.1.6	os_mousemove	141
9.22.1.7	os_setupindicators	141
9.22.1.8	uinputopen	142
9.23	src/ckb-daemon/input_mac.c File Reference	143
9.24	src/ckb-daemon/input_mac_mouse.c File Reference	143
9.25	src/ckb-daemon/keymap.c File Reference	143
9.25.1	Macro Definition Documentation	144
9.25.1.1	BUTTON_HID_COUNT	144
9.25.2	Function Documentation	144
9.25.2.1	corsair_kbcopy	144
9.25.2.2	corsair_mousecopy	145
9.25.2.3	hid_kb_translate	145
9.25.2.4	hid_mouse_translate	147
9.25.3	Variable Documentation	148
9.25.3.1	keymap	148
9.26	src/ckb-daemon/keymap.h File Reference	148
9.26.1	Data Structure Documentation	149
9.26.1.1	struct key	149
9.26.2	Macro Definition Documentation	150
9.26.2.1	BTN_WHEELDOWN	150
9.26.2.2	BTN_WHEELUP	150
9.26.2.3	KEY_BACKSLASH_ISO	150
9.26.2.4	KEY_CORSAIR	150
9.26.2.5	KEY_NONE	151
9.26.2.6	KEY_UNBOUND	151
9.26.2.7	LED_DPI	151
9.26.2.8	LED_MOUSE	151
9.26.2.9	MOUSE_BUTTON_FIRST	151
9.26.2.10	MOUSE_EXTRA_FIRST	151
9.26.2.11	N_BUTTONS_EXTENDED	151
9.26.2.12	N_BUTTONS_HW	151
9.26.2.13	N_KEY_ZONES	151

9.26.2.14	N_KEYBYTES_EXTENDED	151
9.26.2.15	N_KEYBYTES_HW	152
9.26.2.16	N_KEYBYTES_INPUT	152
9.26.2.17	N_KEYS_EXTENDED	152
9.26.2.18	N_KEYS_EXTRA	152
9.26.2.19	N_KEYS_HW	152
9.26.2.20	N_KEYS_INPUT	152
9.26.2.21	N_MOUSE_ZONES	152
9.26.2.22	N_MOUSE_ZONES_EXTENDED	152
9.26.2.23	SCAN_KBD	152
9.26.2.24	SCAN_MOUSE	152
9.26.2.25	SCAN_SILENT	153
9.26.3	Function Documentation	153
9.26.3.1	corsair_kbcopy	153
9.26.3.2	corsair_mousecopy	153
9.26.3.3	hid_kb_translate	154
9.26.3.4	hid_mouse_translate	155
9.26.4	Variable Documentation	156
9.26.4.1	keymap	156
9.27	src/ckb-daemon/keymap_mac.h File Reference	156
9.28	src/ckb-daemon/led.c File Reference	157
9.28.1	Function Documentation	158
9.28.1.1	cmd_iauto	158
9.28.1.2	cmd_inotify	158
9.28.1.3	cmd_ioff	159
9.28.1.4	cmd_ion	159
9.28.1.5	cmd_rgb	159
9.28.1.6	has_key	160
9.28.1.7	iselect	160
9.28.1.8	printrgb	161
9.29	src/ckb-daemon/led.h File Reference	163
9.29.1	Function Documentation	163
9.29.1.1	cmd_iauto	163
9.29.1.2	cmd_inotify	164
9.29.1.3	cmd_ioff	164
9.29.1.4	cmd_ion	165
9.29.1.5	cmd_rgb	165
9.29.1.6	loadrgb_kb	166
9.29.1.7	loadrgb_mouse	168
9.29.1.8	printrgb	168

9.29.1.9	savergb_kb	170
9.29.1.10	savergb_mouse	171
9.29.1.11	updatergb_kb	171
9.29.1.12	updatergb_mouse	173
9.30	src/ckb-daemon/led_keyboard.c File Reference	174
9.30.1	Macro Definition Documentation	197
9.30.1.1	BR1	197
9.30.1.2	BR2	197
9.30.1.3	BR4	197
9.30.1.4	O0	197
9.30.1.5	O1	197
9.30.1.6	O2	197
9.30.1.7	O3	197
9.30.1.8	O4	197
9.30.1.9	O5	197
9.30.1.10	O6	197
9.30.1.11	O7	197
9.30.1.12	O8	197
9.30.2	Function Documentation	198
9.30.2.1	loadrgb_kb	198
9.30.2.2	makergb_512	200
9.30.2.3	makergb_full	200
9.30.2.4	ordered8to3	201
9.30.2.5	quantize8to3	201
9.30.2.6	rgbcmp	202
9.30.2.7	savergb_kb	202
9.30.2.8	updatergb_kb	204
9.30.3	Variable Documentation	205
9.30.3.1	bit_reverse_table	206
9.31	src/ckb-daemon/led_mouse.c File Reference	207
9.31.1	Function Documentation	207
9.31.1.1	isblack	207
9.31.1.2	loadrgb_mouse	208
9.31.1.3	rgbcmp	208
9.31.1.4	savergb_mouse	209
9.31.1.5	updatergb_mouse	209
9.32	src/ckb-daemon/main.c File Reference	210
9.32.1	Function Documentation	211
9.32.1.1	localecase	211
9.32.1.2	main	212

9.32.1.3	quit	214
9.32.1.4	quitWithLock	215
9.32.1.5	restart	216
9.32.1.6	sighandler	217
9.32.1.7	sighandler2	218
9.32.1.8	timespec_add	218
9.32.2	Variable Documentation	219
9.32.2.1	features_mask	219
9.32.2.2	hwload_mode	219
9.32.2.3	main_ac	219
9.32.2.4	main_av	219
9.32.2.5	reset_stop	219
9.33	src/ckb-daemon/notify.c File Reference	219
9.33.1	Macro Definition Documentation	220
9.33.1.1	HW_STANDARD	220
9.33.1.2	HWMODE_OR_RETURN	220
9.33.2	Function Documentation	221
9.33.2.1	_cmd_get	221
9.33.2.2	cmd_get	223
9.33.2.3	cmd_notify	224
9.33.2.4	cmd_restart	224
9.33.2.5	nprintf	225
9.33.2.6	nprintind	226
9.33.2.7	nprintkey	226
9.33.2.8	restart	227
9.34	src/ckb-daemon/notify.h File Reference	228
9.34.1	Function Documentation	229
9.34.1.1	cmd_get	229
9.34.1.2	cmd_notify	230
9.34.1.3	cmd_restart	230
9.34.1.4	nprintf	230
9.34.1.5	nprintind	231
9.34.1.6	nprintkey	232
9.35	src/ckb-daemon/os.h File Reference	232
9.35.1	Macro Definition Documentation	233
9.35.1.1	_DEFAULT_SOURCE	233
9.35.1.2	_GNU_SOURCE	233
9.35.1.3	euid_guard_start	233
9.35.1.4	euid_guard_stop	233
9.35.1.5	UINPUT_VERSION	234

9.36	src/ckb-daemon/profile.c File Reference	234
9.36.1	Function Documentation	235
9.36.1.1	_freeprofile	235
9.36.1.2	allocprofile	235
9.36.1.3	cmd_erase	236
9.36.1.4	cmd_eraseprofile	237
9.36.1.5	cmd_id	237
9.36.1.6	cmd_name	238
9.36.1.7	cmd_profileid	238
9.36.1.8	cmd_profilename	239
9.36.1.9	freemode	239
9.36.1.10	freeprofile	240
9.36.1.11	gethwmodename	240
9.36.1.12	gethwprofilename	241
9.36.1.13	getid	242
9.36.1.14	getmodename	242
9.36.1.15	getprofilename	243
9.36.1.16	hwtonative	244
9.36.1.17	initmode	244
9.36.1.18	loadprofile	245
9.36.1.19	nativetohw	245
9.36.1.20	printname	246
9.36.1.21	setid	247
9.36.1.22	u16dec	247
9.36.1.23	u16enc	248
9.36.1.24	urldecode2	249
9.36.1.25	urlencode2	249
9.36.2	Variable Documentation	250
9.36.2.1	utf16to8	250
9.36.2.2	utf8to16	250
9.37	src/ckb-daemon/profile.h File Reference	250
9.37.1	Macro Definition Documentation	251
9.37.1.1	hwloadprofile	251
9.37.2	Function Documentation	251
9.37.2.1	allocprofile	251
9.37.2.2	cmd_erase	252
9.37.2.3	cmd_eraseprofile	253
9.37.2.4	cmd_hwload_kb	253
9.37.2.5	cmd_hwload_mouse	254
9.37.2.6	cmd_hwsave_kb	255

9.37.2.7	cmd_hwsave_mouse	256
9.37.2.8	cmd_id	257
9.37.2.9	cmd_name	257
9.37.2.10	cmd_profileid	258
9.37.2.11	cmd_profilename	258
9.37.2.12	freeprofile	259
9.37.2.13	gethwmodename	259
9.37.2.14	gethwprofilename	260
9.37.2.15	getid	261
9.37.2.16	getmodename	261
9.37.2.17	getprofilename	262
9.37.2.18	hwtonative	263
9.37.2.19	loadprofile	263
9.37.2.20	nativetohw	264
9.37.2.21	setid	264
9.38	src/ckb-daemon/profile_keyboard.c File Reference	265
9.38.1	Function Documentation	265
9.38.1.1	cmd_hwload_kb	265
9.38.1.2	cmd_hwsave_kb	266
9.38.1.3	hwloadmode	267
9.39	src/ckb-daemon/profile_mouse.c File Reference	268
9.39.1	Function Documentation	268
9.39.1.1	cmd_hwload_mouse	269
9.39.1.2	cmd_hwsave_mouse	270
9.40	src/ckb-daemon/structures.h File Reference	270
9.40.1	Data Structure Documentation	272
9.40.1.1	struct usbid	272
9.40.1.2	struct macroaction	273
9.40.1.3	struct keymacro	274
9.40.1.4	struct binding	275
9.40.1.5	struct dpiset	276
9.40.1.6	struct lighting	277
9.40.1.7	struct usbmode	278
9.40.1.8	struct usbprofile	279
9.40.1.9	struct hwprofile	280
9.40.1.10	struct usbinput	281
9.40.1.11	struct usbdevice	282
9.40.2	Macro Definition Documentation	284
9.40.2.1	CLEAR_KEYBIT	284
9.40.2.2	DPI_COUNT	284

9.40.2.3	FEAT_ADJRATE	284
9.40.2.4	FEAT_ANSI	284
9.40.2.5	FEAT_BIND	284
9.40.2.6	FEAT_COMMON	284
9.40.2.7	FEAT_FWUPDATE	284
9.40.2.8	FEAT_FWVERSION	285
9.40.2.9	FEAT_HWLOAD	285
9.40.2.10	FEAT_ISO	285
9.40.2.11	FEAT_LMASK	285
9.40.2.12	FEAT_MONOCHROME	285
9.40.2.13	FEAT_MOUSEACCEL	285
9.40.2.14	FEAT_NOTIFY	285
9.40.2.15	FEAT_POLLRATE	285
9.40.2.16	FEAT_RGB	285
9.40.2.17	FEAT_STD_NRGB	286
9.40.2.18	FEAT_STD_RGB	286
9.40.2.19	HAS_ANY_FEATURE	286
9.40.2.20	HAS_FEATURES	286
9.40.2.21	HWMODE_K70	286
9.40.2.22	HWMODE_K95	286
9.40.2.23	HWMODE_MAX	286
9.40.2.24	I_CAPS	286
9.40.2.25	I_NUM	286
9.40.2.26	I_SCROLL	286
9.40.2.27	IFACE_MAX	287
9.40.2.28	KB_NAME_LEN	287
9.40.2.29	LIFT_MAX	287
9.40.2.30	LIFT_MIN	287
9.40.2.31	MACRO_MAX	287
9.40.2.32	MD_NAME_LEN	287
9.40.2.33	MODE_COUNT	287
9.40.2.34	MSG_SIZE	287
9.40.2.35	NEEDS_FW_UPDATE	287
9.40.2.36	OUTFIFO_MAX	288
9.40.2.37	PR_NAME_LEN	288
9.40.2.38	SCROLL_ACCELERATED	288
9.40.2.39	SCROLL_MAX	288
9.40.2.40	SCROLL_MIN	288
9.40.2.41	SERIAL_LEN	288
9.40.2.42	SET_KEYBIT	288

9.40.3	Variable Documentation	288
9.40.3.1	vtable_keyboard	288
9.40.3.2	vtable_keyboard_nonrgb	288
9.40.3.3	vtable_mouse	289
9.41	src/ckb-daemon/usb.c File Reference	289
9.41.1	Function Documentation	290
9.41.1.1	_resetusb	290
9.41.1.2	_setupusb	290
9.41.1.3	_usbrecv	295
9.41.1.4	_usbsend	296
9.41.1.5	closeusb	297
9.41.1.6	devmain	299
9.41.1.7	get_vtable	301
9.41.1.8	product_str	302
9.41.1.9	revertusb	303
9.41.1.10	setupusb	304
9.41.1.11	usb_tryreset	305
9.41.1.12	vendor_str	306
9.41.2	Variable Documentation	306
9.41.2.1	features_mask	306
9.41.2.2	hwload_mode	306
9.41.2.3	reset_stop	307
9.41.2.4	usbmutex	307
9.42	src/ckb-daemon/usb.h File Reference	307
9.42.1	Detailed Description	310
9.42.2	Macro Definition Documentation	311
9.42.2.1	DELAY_LONG	311
9.42.2.2	DELAY_MEDIUM	311
9.42.2.3	DELAY_SHORT	311
9.42.2.4	IS_FULLRANGE	311
9.42.2.5	IS_K65	311
9.42.2.6	IS_K70	311
9.42.2.7	IS_K95	312
9.42.2.8	IS_M65	312
9.42.2.9	IS_MONOCHROME	312
9.42.2.10	IS_MONOCHROME_DEV	312
9.42.2.11	IS_MOUSE	312
9.42.2.12	IS_MOUSE_DEV	312
9.42.2.13	IS_RGB	312
9.42.2.14	IS_RGB_DEV	312

9.42.2.15 IS_SABRE	312
9.42.2.16 IS_SCIMITAR	313
9.42.2.17 IS_STRAFE	313
9.42.2.18 NK95_HWOFF	313
9.42.2.19 NK95_HWON	313
9.42.2.20 NK95_M1	313
9.42.2.21 NK95_M2	313
9.42.2.22 NK95_M3	313
9.42.2.23 nk95cmd	313
9.42.2.24 P_K65	314
9.42.2.25 P_K65_LUX	314
9.42.2.26 P_K65_LUX_STR	314
9.42.2.27 P_K65_NRGB	314
9.42.2.28 P_K65_NRGB_STR	314
9.42.2.29 P_K65_RFIRE	314
9.42.2.30 P_K65_RFIRE_STR	314
9.42.2.31 P_K65_STR	314
9.42.2.32 P_K70	314
9.42.2.33 P_K70_LUX	314
9.42.2.34 P_K70_LUX_NRGB	315
9.42.2.35 P_K70_LUX_NRGB_STR	315
9.42.2.36 P_K70_LUX_STR	315
9.42.2.37 P_K70_NRGB	315
9.42.2.38 P_K70_NRGB_STR	315
9.42.2.39 P_K70_RFIRE	315
9.42.2.40 P_K70_RFIRE_NRGB	315
9.42.2.41 P_K70_RFIRE_NRGB_STR	315
9.42.2.42 P_K70_RFIRE_STR	315
9.42.2.43 P_K70_STR	315
9.42.2.44 P_K95	315
9.42.2.45 P_K95_NRGB	316
9.42.2.46 P_K95_NRGB_STR	316
9.42.2.47 P_K95_PLATINUM	316
9.42.2.48 P_K95_PLATINUM_STR	316
9.42.2.49 P_K95_STR	316
9.42.2.50 P_M65	316
9.42.2.51 P_M65_PRO	316
9.42.2.52 P_M65_PRO_STR	316
9.42.2.53 P_M65_STR	316
9.42.2.54 P_SABRE_L	316

9.42.2.55	P_SABRE_L_STR	316
9.42.2.56	P_SABRE_N	317
9.42.2.57	P_SABRE_N_STR	317
9.42.2.58	P_SABRE_O	317
9.42.2.59	P_SABRE_O2	317
9.42.2.60	P_SABRE_O2_STR	317
9.42.2.61	P_SABRE_O_STR	317
9.42.2.62	P_SCIMITAR	317
9.42.2.63	P_SCIMITAR_PRO	317
9.42.2.64	P_SCIMITAR_PRO_STR	317
9.42.2.65	P_SCIMITAR_STR	317
9.42.2.66	P_STRAFE	318
9.42.2.67	P_STRAFE_NRGB	318
9.42.2.68	P_STRAFE_NRGB_STR	318
9.42.2.69	P_STRAFE_STR	318
9.42.2.70	resetusb	318
9.42.2.71	USB_DELAY_DEFAULT	318
9.42.2.72	usbrecv	318
9.42.2.73	usbsend	318
9.42.2.74	V_CORSAIR	319
9.42.2.75	V_CORSAIR_STR	319
9.42.3	Function Documentation	319
9.42.3.1	_nk95cmd	319
9.42.3.2	_resetusb	320
9.42.3.3	_usbrecv	321
9.42.3.4	_usbsend	323
9.42.3.5	closeusb	324
9.42.3.6	os_closeusb	326
9.42.3.7	os_inputmain	327
9.42.3.8	os_resetusb	330
9.42.3.9	os_sendindicators	332
9.42.3.10	os_setupusb	334
9.42.3.11	os_usbrecv	335
9.42.3.12	os_usbsend	337
9.42.3.13	product_str	339
9.42.3.14	revertusb	340
9.42.3.15	setupusb	341
9.42.3.16	usb_tryreset	342
9.42.3.17	usbkill	343
9.42.3.18	usbmain	344

9.42.3.19 vendor_str	345
9.43 src/ckb-daemon/usb_linux.c File Reference	346
9.43.1 Data Structure Documentation	348
9.43.1.1 struct_model	348
9.43.2 Macro Definition Documentation	348
9.43.2.1 DEBUG	348
9.43.2.2 DEBUG	348
9.43.2.3 N_MODELS	348
9.43.2.4 TEST_RESET	348
9.43.3 Function Documentation	349
9.43.3.1 _nk95cmd	349
9.43.3.2 os_closeusb	350
9.43.3.3 os_inputmain	351
9.43.3.4 os_resetusb	354
9.43.3.5 os_sendindicators	355
9.43.3.6 os_setupusb	356
9.43.3.7 os_usbrecv	358
9.43.3.8 os_usbsend	359
9.43.3.9 strtrim	361
9.43.3.10 udev_enum	361
9.43.3.11 usb_add_device	363
9.43.3.12 usb_rm_device	365
9.43.3.13 usbadd	366
9.43.3.14 usbclaim	367
9.43.3.15 usbkill	368
9.43.3.16 usbmain	368
9.43.3.17 usbunclaim	370
9.43.4 Variable Documentation	371
9.43.4.1 kbsyspath	371
9.43.4.2 models	371
9.43.4.3 udev	372
9.43.4.4 udevthread	372
9.43.4.5 usbthread	372
9.44 src/ckb-daemon/usb_mac.c File Reference	372

Chapter 1

ckb-next: RGB Driver for Linux and OS X

ckb-next is an open-source driver for Corsair keyboards and mice. It aims to bring the features of their proprietary CUE software to the Linux and Mac operating systems. This project is currently a work in progress, but it already supports much of the same functionality, including full RGB animations. More features are coming soon. Testing and bug reports are appreciated!

Disclaimer: ckb-next is not an official Corsair product. It is licensed under the GNU General Public License (version 2) in the hope that it will be useful, but with NO WARRANTY of any kind.

What happened to the original `ckb`?

We don't know. The author of **ckb** `ccMSC` suddenly disappeared and hasn't showed up since July 2016. So the community around ckb decided to take the project over and continue its development. That's how **ckb-next** was created. Currently it's not rock solid and very easy to set up on newer systems but we are actively working on this. Nevertheless the project already incorporates a notable amount of fixes and patches in comparison to the original ckb.

Contents

- [Current Status](#)
- [Device Support](#)
- [Linux Installation](#)
- [OS X/macOS Installation](#)
- [Usage](#)
- [Troubleshooting](#)
- [Known Issues](#)
- [Contributing](#)

See also:

- <https://github.com/mattanger/ckb-next/blob/master/DAEMON.md> "Manual for the driver daemon"
- [ckb testing repository](#) (updated more frequently, but may be unstable)

Device Support

Keyboards:

- K65 RGB
- K70
- K70 RGB
- K70 LUX RGB
- K95*
- K95 RGB
- Strafe
- Strafe RGB
- = hardware playback not supported. Settings will be saved to software only.

Mice:

- M65 RGB
- M65 PRO RGB
- Sabre RGB
- Scimitar RGB

Linux Installation

Pre-made packages:

ckb-next packages:

- Fedora 24/25, CentOS/RHEL 7 (maintained by):
 - `'johanh/ckb'` - based on `master` branch

ckb packages (deprecated):

- Arch Linux (maintained by):
 - `'aur/ckb-git'` - based on `master` branch (more stable)
 - `'aur/ckb-git-latest'` - based on `testing` branch (less stable but fresher)
- Gentoo (maintained by): `emerge -av app-misc/ckb`

If you are a package maintainer or want to discuss something with package maintainers let us know in the issues, so we can have an accountable and centralized communication about this. *If you would like to maintain a package for your favorite distro/OS, please let us know as well.*

Preparation:

ckb-next requires Qt5 (Qt 5.6 recommended for OS X), libudev, zlib, gcc, g++, and glibc.

- **Ubuntu:** `sudo apt-get install build-essential libudev-dev qt5-default zlib1g-dev libappindicator-dev`
- **Fedora:** `sudo dnf install zlib-devel qt5-qtbase-devel libgudev-devel libappindicator-devel systemd-devel gcc-c++`
- **Arch:** `sudo pacman -S base-devel qt5-base zlib`
- **Other distros:** Look for `qt5` or `libqt5*-devel`

Note: If you build your own kernels, ckb-next requires the `uinput` flag to be enabled. It is located in `Device Drivers -> Input Device Support -> Miscellaneous devices -> User level driver support`. If you don't know what this means, you can ignore this.

Installing:

You can download ckb-next using the "Download zip" option on the right. Extract it and open the `ckb-master` directory. The easiest way to install ckb is to double-click the `quickinstall` script and run it in a Terminal. It will attempt to build ckb and then ask if you'd like to install/run the application. If the build doesn't succeed, or if you'd like to compile ckb manually, see <https://github.com/ccMSC/ckb/blob/master/BUILD.md> "BUILD.md" for instructions.

Upgrading:

To install a new version of ckb, or to reinstall the same version, first delete the `ckb-master` directory and the zip file from your previous download. Then download the source code again and re-run `quickinstall`. The script will automatically replace the previous installation. You may need to reboot afterward.

Uninstalling:

First, stop the `ckb-daemon` service and remove the service file.

- If you have `systemd` (Ubuntu versions starting with 15.04): `sudo systemctl stop ckb-daemon sudo rm -f /usr/lib/systemd/system/ckb-daemon.service`
- If you have `Upstart` (Ubuntu versions earlier than 15.04): `sudo service ckb-daemon stop sudo rm -f /etc/init/ckb-daemon.conf`
- If you have `OpenRC`: `sudo rc-service ckb-daemon stop sudo rc-update del ckb-daemon default sudo rm -f /etc/init.d/ckb-daemon`
- If you're not sure, re-run the `quickinstall` script and proceed to the service installation. The script will say `System service: Upstart detected` or `System service: systemd detected`. Please be aware that `OpenRC` is currently not detected automatically.

Afterward, remove the applications and related files: `sudo rm -f /usr/bin/ckb /usr/bin/ckb-daemon /usr/share/applications/ckb.desktop /usr/share/icons/hicolor/512x512/apps/ckb.png sudo rm -rf /usr/lib/ckb-animations`

Before <https://github.com/mattanger/ckb-next/commit/f347e60df211c60452f95084b6c46dc4ec5f42> animations were located elsewhere, try removing them as well: `sudo rm -rf /usr/bin/ckb-animations`

OS X/macOS Installation**Binary download:**

The latest OS X/macOS binary can be downloaded here: <https://github.com/mattanger/ckb-next/releases/latest>

Click on `ckb.pkg` under the Downloads section. This is an automated installer which will set up the driver for you. After it's finished, open `ckb.app` (it will be installed to your Applications directory) to get started.

Building from source:

Install the latest version of Xcode from the App Store. Open Xcode, accept the license agreement, and wait for it to install any additional components (if necessary). When you see the "Welcome to Xcode" screen, the setup is finished and you can close the app. Then install Xcode Command Line Tools package issuing `xcode-select --install` in a Terminal app. Afterwards install Qt5 from here: <http://www.qt.io/download-open-source/>

The easiest way to build the driver is with the `quickinstall` script, which is present in the `ckb-master` folder. Double-click on `quickinstall` and it will compile the app for you, then ask if you'd like to install it system-wide. If the build fails for any reason, or if you'd like to compile manually, see <https://github.com/ccMSC/ckb/blob/master/BUILD.md> "BUILD.md".

Upgrading (binary):

Download the latest `ckb.pkg`, run the installer, and reboot. The newly-installed driver will replace the old one.

Upgrading (source):

Remove the existing `ckb-master` directory and zip file. Re-download the source code and run the `quickinstall` script again. The script will automatically replace the previous installation. You may need to reboot afterward.

Uninstalling:

Drag `ckb.app` into the trash. If the system service file isn't cleaned up automatically, you can find it and remove it here: `/Library/LaunchDaemons/com.ckb.daemon.plist`.

Usage

The user interface is still a work in progress.

Major features:

- Control multiple devices independently
- United States and European keyboard layouts
- Customizable key bindings
- Per-key lighting and animation
- Reactive lighting
- Multiple profiles/modes with hardware save function
- Adjustable mouse DPI with ability to change DPI on button press
- Key macros (G-Keys also); Have a look at https://youtu.be/qhrKP03_NrM for a short video tutorial
- Key macro delays: Handle delays between keystrokes when playing a macro

Closing `ckb` will actually minimize it to the system tray. Use the Quit option from the tray icon or the settings screen to exit the application.

Roadmap (roughly in order)

- **v0.3 release:**
 - Ability to store profiles separately from devices, import/export them
 - More functions for the Win Lock key
- **v0.4 release:**

- Ability to import CUE profiles
- Ability to tie profiles to which application has focus
- **v0.5 release:**
 - Key combos
 - Timers?
- **v1.0 release:**
 - OSD? (Not sure if this can actually be done)
 - Extra settings?
 - ????

Troubleshooting

Linux

If you have problems connecting the device to your system (device doesn't respond, ckb-daemon doesn't recognize or can't connect it) and/or you experience long boot times when using the keyboard, try adding the following to your kernel's `cmdline`:

- K65 RGB: `usbhid.quirks=0x1B1C:0x1B17:0x20000408`
- K70: `usbhid.quirks=0x1B1C:0x1B09:0x0x20000408`
- K70 RGB: `usbhid.quirks=0x1B1C:0x1B13:0x20000408`
- K95: `usbhid.quirks=0x1B1C:0x1B08:0x20000408`
- K95 RGB: `usbhid.quirks=0x1B1C:0x1B11:0x20000408`
- Strafe: `usbhid.quirks=0x1B1C:0x1B15:0x20000408`
- Strafe RGB: `usbhid.quirks=0x1B1C:0x1B20:0x20000408`
- M65 RGB: `usbhid.quirks=0x1B1C:0x1B12:0x20000408`
- Sabre RGB Optical: `usbhid.quirks=0x1B1C:0x1B14:0x20000408`
- Sabre RGB Laser: `usbhid.quirks=0x1B1C:0x1B19:0x20000408`
- Scimitar RGB: `usbhid.quirks=0x1B1C:0x1B1E:0x20000408`

For instructions on adding `cmdline` parameters in Ubuntu, see <https://wiki.ubuntu.com/Kernel/-KernelBootParameters>

If you have multiple devices, combine them with commas, starting after the `=`. For instance, for K70 RGB + M65 RGB: `usbhid.quirks=0x1B1C:0x1B13:0x20000408,0x1B1C:0x1B12:0x20000408`

If it still doesn't work, try replacing `0x20000408` with `0x4`. Note that this will cause the kernel driver to ignore the device(s) completely, so you need to ensure `ckb-daemon` is running at boot or else you'll have no input. This will not work if you are using full-disk encryption.

If you see **GLib** critical errors like 'GLib-GObject-CRITICAL **: g_type_add_interface_static: assertion 'G_TYPE_IS_INSTANTIATABLE (instance_type)' failed' and you are using:

- Qt 5.8 and newer, remove your Qt configuration files and restart the ckb GUI. Also watch out for different style overridings in dotfiles under `~/` generated by Qt automatically, remove them as well.
- Qt 5.7 and lower, install `qt5ct` package on Arch Linux (find a similar one for your distribution). That's all. This is a known Qt bug. It happened because Qt did not ship required GTK files.

If you're using **Unity** and the tray icon doesn't appear correctly, run `sudo apt-get install libappindicator-dev`. Then reinstall ckb.

OS X/macOS

- ****“ckb.pkg” can’t be opened because it is from an unidentified developer**** Open System Preferences > Security & Privacy > General and click Open Anyway.
- **Modifier keys (Shift, Ctrl, etc.) are not rebound correctly** ckb does not recognize modifier keys rebound from System Preferences. You can rebound them again within the application.
- ****~ key prints \$±**** Check your keyboard layout on ckb’s Settings screen. Choose the layout that matches your physical keyboard.
- **Compile problems** Can usually be resolved by rebooting your computer and/or reinstalling Qt. Make sure that Xcode works on its own. If a compile fails, delete the `ckb-master` directory as well as any automatically generated `build-ckb` folders and try again from a new download.
- **Scroll wheel does not scroll** As of #c3474d2 it’s now possible to **disable scroll acceleration** from the GUI. You can access it under "OSX tweaks" in the "More settings" screen. Once disabled, the scroll wheel should behave consistently.

General

Please ensure your keyboard firmware is up to date. If you’ve just bought the keyboard, connect it to a Windows computer first and update the firmware from Corsair’s official utility.

Before reporting an issue, connect your keyboard to a Windows computer and see if the problem still occurs. If it does, contact Corsair. Additionally, please check the Corsair user forums to see if your issue has been reported by other users. If so, try their solutions first.

Common issues:

- **Problem:** ckb says "No devices connected" or "Driver inactive"
- **Solution:** Try rebooting the computer and/or reinstalling ckb. Try removing the keyboard and plugging it back in. If the error doesn’t go away, try the following:
- **Problem:** Keyboard doesn’t work in BIOS, doesn’t work at boot
- **Solution:** Some BIOSes have trouble communicating with the keyboard. They may prevent the keyboard from working correctly in the operating system as well. First, try booting the OS *without* the keyboard attached, and plug the keyboard in after logging in. If the keyboard works after the computer is running but does not work at boot, you may need to use the keyboard’s BIOS mode option.
- BIOS mode can be activated using the poll rate switch at the back of the keyboard. Slide it all the way to the position marked "BIOS". You should see the scroll lock light blinking to indicate that it is on. (Note: Unfortunately, this has its own problems - see Known Issues. You may need to activate BIOS mode when booting the computer and deactivate it after logging in).
- **Problem:** Keyboard isn’t detected when plugged in, even if driver is already running
- **Solution:** Try moving to a different USB port. Be sure to follow [Corsair’s USB connection requirements](#). Note that the keyboard does not work with some USB3 controllers - if you have problems with USB3 ports, try USB2 instead. If you have any USB hubs on hand, try those as well. You may also have success sliding the poll switch back and forth a few times.

Reporting issues

If you have a problem that you can’t solve (and it isn’t mentioned in the Known Issues section below), you can report it on [the GitHub issue tracker](#). Before opening a new issue, please check to see if someone else has reported your problem already - if so, feel free to leave a comment there.

Known issues

- Using the keyboard in BIOS mode prevents the media keys (including mute and volume wheel), as well as the K95's G-keys from working. This is a hardware limitation.
- The tray icon doesn't appear in some desktop environments. This is a known Qt bug. If you can't see the icon, reopen ckb to bring the window back.
- When starting the driver manually, the Terminal window sometimes gets spammed with enter keys. You can stop it by unplugging and replugging the keyboard or by moving the poll rate switch.
- When stopping the driver manually, the keyboard sometimes stops working completely. You can reconnect it by moving the poll rate switch.

Contributing

You can contribute to the project by [opening a pull request](#). It's best if you base your changes off of the `testing` branch as opposed to the `master`, because the pull request will be merged there first. If you'd like to contribute but don't know what you can do, take a look at [the issue tracker](#) and see if any features/problems are still unresolved. Feel free to ask if you'd like some ideas.

Chapter 2

Building ckb

Linux

You can build the project by running `./qmake-auto && make` in a Terminal inside the `ckb-master` directory. The binaries will be placed in a new `bin` directory assuming they compile successfully. If you get a `No suitable qmake found` error, make sure Qt5 is installed and up to date. You may have to invoke `qmake` manually, then run `make` on its own. If you have Qt Creator installed, you can open `ckb.pro` (when asked to configure the project, make sure "Desktop" is checked) and use `Build > Build Project "ckb"` (Ctrl+B) to build the application instead.

Running as a service:

First copy the binary and the service files to their system directories:

- **Upstart (Ubuntu, prior to 15.04):** `sudo cp -R bin/* /usr/bin && sudo cp service/upstart/ckb-daemon.conf /etc/init`
- **Systemd (Ubuntu 15.04 and later):** `sudo cp -R bin/* /usr/bin && sudo cp service/systemd/ckb-daemon.service /usr/lib/systemd/system`
- **OpenRC:** `sudo cp -R bin/* /usr/bin && sudo cp service/openrc/ckb-daemon /etc/init.d/`

To launch the driver and enable it at start-up:

- **Upstart:** `sudo service ckb-daemon start`
- **Systemd:** `sudo systemctl start ckb-daemon && sudo systemctl enable ckb-daemon`
- **OpenRC:** `sudo rc-service ckb-daemon start && sudo rc-update add ckb-daemon default`

Open the `bin` directory and double-click on `ckb` to launch the user interface. If you want to run it at login, add `ckb --background` to your Startup Applications.

Running manually:

Open the `bin` directory in a Terminal and run `sudo ./ckb-daemon` to start the driver. To start the user interface, run `./ckb`. Running the driver manually may be useful for testing/debugging purposes, but you must leave the terminal window open and you'll have to re-run it at every reboot, so installing it as a service is the best long-term solution.

OSX

Open `ckb.pro` in Qt Creator. You should be prompted to configure the project (make sure the "Desktop" configuration is selected and not iOS). Once it's finished loading, press `Cmd+B` or select `Build > Build Project "ckb"` from the menu bar. When it's done, you should see a newly-created `ckb.app` in the project directory. Exit Qt Creator.

Alternatively, open a Terminal in the `ckb-master` directory and run `./qmake-auto && make`. It will detect Qt automatically if you installed it to one of the standard locations. You should see a newly created `ckb.app` if the build is successful.

Running as a service:

Copy `ckb.app` to your Applications folder. Copy the file `'service/launchd/com.ckb.daemon.plist'` to your computer's `/Library/LaunchDaemons` folder (you can get to it by pressing `Cmd+Shift+G` in Finder and typing the location). Then open a Terminal and run the following commands to launch the driver:

```
"" sudo chown root:wheel /Library/LaunchDaemons/com.ckb.daemon.plist sudo chmod 0700 /Library/LaunchDaemons/com.ckb.daemon.plist sudo launchctl load /Library/LaunchDaemons/com.ckb.daemon.plist ""
```

After you're done, open `ckb.app` to launch the user interface.

Running manually:

Open a Terminal in the `ckb` directory and run `sudo ckb.app/Contents/Resources/ckb-daemon` to start the driver. Open `ckb.app` to start the user interface. Note that you must leave the terminal window open and must re-launch the driver at every boot if you choose this; installing as a service is the better long term solution.

Chapter 3

cbk Improvements Roadmap

Short term plan

- merge existing PR submitted to original ckb repo
- Contact other developers interested in collaboration on a new and improved version of ckb
- Figure out the issues relating to MacOS Sierra and other version
- Device support:
 - Determine which will need support other than just USB id additions
- Address existing bugs. Not help requests.

Chapter 4

DAEMON

The daemon provides devices at `/dev/input/ckb*`, where `*` is the device number, starting at 1. Up to 9 devices may be connected at once and controlled independently. The daemon additionally provides `/dev/input/ckb0`, which stores driver information.

Mac note: The devices on OSX are located at `/var/run/ckb*` and not `/dev/input/ckb*`. So wherever you see `/dev/input` in this document, replace it with `/var/run`.

`/dev/input/ckb0` contains the following files:

- `connected`: A list of all connected devices, one per line. Each line contains a device path followed by the device's serial number and its description.
- `pid`: The process identifier of the daemon.
- `version`: The daemon version.

Other `ckb*` devices contain the following:

- `cmd`: Keyboard controller.
- `notify0`: Keyboard notifications.
- `features`: Device features.
- `fwversion`: Device firmware version (not present on all devices).
- `model`: Device description/model.
- `pollrate`: Poll rate in milliseconds (not present on all devices).
- `serial`: Device serial number. `model` and `serial` will match the info found in `ckb0/connected`

Commands

The `/dev/input/ckb*/cmd` nodes accept input in the form of text commands. They are normally accessible to all users on the system (see Security section). Commands should be given in the following format: `[mode <n>] command1 [parameter1] [command2] [parameter2] [command3] [parameter3] ...`

In a terminal shell, you can do this like `echo mycommand > /dev/input/ckb1/cmd`. Programmatically, you can open and write them as regular files. When programming, you must append a newline character and flush the output before your command(s) will actually be read.

The `mode` parameter is used to group settings. Most (but not all) settings are mode-specific; that is, changing mode 1 will not affect mode 2. By default, all commands affect the current mode. Use `mode <n> switch` to change the current mode.

When plugged in, all devices start in hardware-controlled mode (also known as idle mode) and will not respond to commands. Before issuing any other commands, write `active` to the command node, like `echo active > /dev/input/ckbl/cmd`. To put the device back into hardware mode, issue the `idle` command.

Features

The `features` node describes features supported by the device, which may not be present on all devices. The first two words in the `features` node are always `<vendor>` `<model>`, like `corsair k70`. After that, any of the following features may appear:

- `adjrate`: Device supports adjustable poll rate.
- `bind`: Device supports key rebinding.
- `fwupdate`: Device supports firmware updates.
- `fwversion`: Device has a detectable firmware version (stored in the `fwversion` node).
- `notify`: Device supports key notifications.
- `pollrate`: Device has a detectable poll rate (stored in the `pollrate` node).
- `rgb`: Device supports RGB lighting.

Keyboard layout

The driver has no concept of keyboard layouts; all keys are referred to by their English names regardless of the underlying hardware. This means that, for instance, in an AZERTY layout the `q` key in `ckb-daemon` corresponds to `A` on the physical keyboard. Note that on UK/european (ISO) layouts, the backslash key (beside left shift) is called `bslash_iso`, while `bslash` refers to the backslash on the US keyboard. The key next to Enter on the ISO keyboard is known as `hash`. See <src/ckb-daemon/keymap.c> for the full table of supported keys.

For technical reasons, the OSX driver may swap the `bslash_iso` and `grave` keys if the keyboard layout is not set correctly. To compensate for this, write `layout iso` or `layout ansi` to the command node.

Poll rate

A device's current poll rate can be read from its `pollrate` node, assuming it has one. Keyboards have a hardware switch to control poll rate and cannot be adjusted via software. However, mice have a software-controlled poll rate. You can change it by issuing `pollrate <interval>` to the command node, where `interval` is the time in milliseconds. Valid poll rates are 1, 2, 4, and 8.

Profiles and modes

Each mode has its own independent binding and lighting setup. When the daemon starts or a keyboard is plugged in, the profile will be loaded from the hardware. By default, all commands will update the currently selected mode. The `mode <n>` command may be used to change the settings for a different mode. Up to 6 modes are available. Each keyboard has one profile, which may be given a name. Mode 1 may be saved to the device hardware, or modes 1-3 in the case of the K95. Modes 4 through 6 are software-only. Profile management commands are as follows:

- `profilename <name>` sets the profile's name. The name must be written without spaces; to add a space, use `%20`.
- `name <name>` sets the current mode's name. Use `mode <n> name <name>` to set a different mode's name.

- `profileid <guid> [<modification>]` sets a profile's ID. The GUID must be written in registry format, like {12345678-ABCD-EF01-2345-6789ABCDEF01}. The optional modification number must be written with 8 hex digits, like ABCDEF01.
- `id <guid> [<modification>]` sets a mode's ID.
- `mode <n> switch` switches the keyboard to mode N. If the mode does not exist, it will be created with a blank ID, black lighting, and default bindings.
- `hwload` loads the RGB profile from the hardware. Key bindings and non-hardware RGB modes are unaffected.
- `hwsave` saves the RGB profile to the hardware.
- `erase` erases the current mode, resetting its lighting and bindings. Use `mode <n> erase` to erase a different mode.
- `eraseprofile` erases the entire profile, deleting its name, ID, and all of its modes.

Examples:

- `profilename My%20Profile mode 1 name Mode%201 mode 2 name Mode%202 mode 3 name Mode%203` will name the profile "My Profile" and name modes 1-3 "Mode 1", "Mode 2", and "Mode 3".
- `eraseprofile hwload` resets the entire profile to its hardware settings.

LED commands

The backlighting is controlled by the `rgb` commands.

- `rgb <RRGGBB>` sets the entire keyboard to the color specified by the hex constant RRGGBB.
- `rgb <key>:<RRGGBB>` sets the specified key to the specified hex color.

Examples:

- `rgb ffffffff` makes the whole keyboard white.
- `rgb 000000` makes the whole keyboard black.
- `rgb esc:ff0000` sets the Esc key red but leaves the rest of the keyboard unchanged.

Multiple keys may be changed to one color when separated with commas, for instance:

- `rgb w,a,s,d:0000ff` sets the WASD keys to blue.

Additionally, multiple commands may be combined into one, for instance:

- `rgb ffffffff esc:ff0000 w,a,s,d:0000ff` sets the Esc key red, the WASD keys blue, and the rest of the keyboard white (note the lack of a key name before `ffffffff`, implying the whole keyboard is to be set).

By default, the controller runs at 30 FPS, meaning that attempts to animate the LEDs faster than that will be ignored. If you wish to change it, send the command `fps <n>`. The maximum frame rate is 60.

For devices running in 512-color mode, color dithering can be enabled by sending the command `dither 1`. The command `dither 0` disables dithering.

Indicators

The indicator LEDs (Num Lock, Caps Lock, Scroll Lock) are controlled with the `i` commands.

- `ioff <led>` turns an indicator off permanently. Valid LED names are `num`, `caps`, and `scroll`.
- `ion <led>` turns an indicator on permanently.
- `iauto <led>` turns an indicator off or on automatically (default behavior).

Binding keys

Keys may be rebound through use of the `bind` commands. Binding is a 1-to-1 operation that translates one keypress to a different keypress regardless of circumstance.

- `bind <key1>:<key2>` remaps `key1` to `key2`.
- `unbind <key>` unbinds a key, causing it to lose all function.
- `rebind <key>` resets a key, returning it to its default binding.

Examples:

- `bind g1:esc` makes G1 become an alternate Esc key (the actual Esc key is not changed).
- `bind caps:tab tab:caps` switches the functions of the Tab and Caps Lock keys.
- `unbind lwin rwin` disables both Windows keys, even without using the keyboard's Windows Lock function.
- `rebind all` resets the whole keyboard to its default bindings.

Key macros

Macros are a more advanced form of key binding, controlled with the `macro` command.

- `macro <keys>:<command>` binds a key combination to a command, where the command is a series of key presses. To combine keys, separate them with `+`; for instance, `lctrl+a` binds a macro to (left) Ctrl+A. In the command field, enter `+``<key>` to trigger a key down or `-<key>` to trigger a key up. To simulate a key press, use `+``<key>`, `-<key>`.
- `macro <keys>:clear` clears commands associated with a key combination. Only one macro may be assigned per combination; assigning a second one will overwrite the first.
- `macro clear` clears all macros.

Examples:

- `macro g1:+lctrl,+a,-a,-lctrl` triggers a Ctrl+A when G1 is pressed.
- `macro g2+g3:+lalt,+f4,-f4,-lalt` triggers an Alt+F4 when G2 and G3 are pressed simultaneously.

Assigning a macro to a key will cause its binding to be ignored; for instance, `macro a:+b,-b` will cause A to generate a B character regardless of its binding. However, `macro lctrl+a:+b,-b` will cause A to generate a B only when Ctrl is also held down.

Macro playback delay

There are two types of playback delay that can be set with macros; global and local. Setting a *global delay* value introduces a time delay between events during macro execution or playback. *Local delay* allows setting the delay after an individual event, overriding the global delay value for that event. Thus global delay can be used to set the overall playback speed of macros and local delays can be used to tune individual events within a macro.

All delay values are specified in microseconds (us) and are positive values from 0 to `UINT_MAX - 1`. This means delays range from 0 to just over 1 hour (4,294,967,294us, 4,294 seconds, 71 minutes, or 1.19 hours). A value of zero (0) represents no delay between actions.

Global macro delay (default delay)

Global delay allows macro playback speed to be changed. It sets the time between (actually after) each recorded macro event. If global delay is set to 1 microsecond then a 1 ms delay will follow each individual macro event when the macro is triggered.

The *global delay* is set with the `ckb-daemon`'s existing (in testing branch) `delay` command followed by an unsigned integer representing the number of microseconds to wait after each macro action and before the next.

Global delay can also be set to `on` which maintains backwards compatibility with the current development of `ckb-daemon` for long macro playback. That is, setting the global delay to `on` introduces a 30us and a 100us delay based on the macro's length during playback.

NOTE: This setting also introduces a delay after the last macro action. This functionality exists in the current testing branch and was left as-is. It is still to be determined if this is a bug or a feature.

Examples:

- `delay 1000` sets a 1,000us delay between action playback.
- `delay on` sets long macro delay; 30us for actions between 20 and 200, 100us for actions > 200.
- `delay off` sets no delay (same as 0).
- `delay 0` sets no delay (same as off).
- `delay spearmint-potato` is invalid input, sets no delay (same as off).

Local macro delay (keystroke delay)

Local Delay allows each macro action to have a post-action delay associated with it. This allows a macro to vary it's playback speed for each event. If no local delay is specified for a macro action, then the `global delay` (above) is used. All delay values are in microsecons (us) as with the global delay setting.

Examples:

- `macro g5:+d,-d,+e=5000,-e,+l,-l=10000,+a,-a,+y,-y=1000000,+enter,-enter` define a macro for `g5` with a 5,000us delay between the `e` down and `e` up actions. A 1,000us delay between `l` up and `a` down, a delay of one second (1,000,000us) after `y` up and before `enter`, and the global delay for all other actions.
- `macro g5:+d,-d=0` use default delay between `d` down and `d` up and no delay (0us) after `d` up. This removes the noted feature/bug (above) where the last action has a trailing delay associated with it.

DPI and mouse settings

DPI settings are stored in a bank. They are controlled with the `dpi` command.

- `dpi <stage>:<x>,<y>` sets the DPI for a given `stage` to `x` by `y`. Valid stages are 0 through 5. In hardware, 1 is the first (lowest) stage and 5 is the highest. Stage 0 is used for Sniper mode.
- `dpi <stage>:<xy>` sets both X and Y.

- `dpi <stage>:off` disables a DPI stage.
- `dpisel <stage>` sets the current stage selection.

In order to change the mouse's current DPI, first update one of the stages with the value you want, then select that stage. For instance:

- `dpi 1:1000 dpisel 1` sets the current DPI to 1000x1000.

Additional mouse settings:

- `lift <height>` sets the lift height, from 1 (lowest) to 5 (highest)
- `snap <on|off>` enables or disables Angle Snap.

Notifications

The keyboard can be configured to generate user-readable notifications on keypress events. These are controlled with the `notify` commands. In order to see events, read from `/dev/input/ckb*/notify0`. In a terminal, you can do this like `cat /dev/input/ckb1/notify0`. Programmatically, you can open it for reading like a regular file.

Note that the file can only reliably be read by one application: if you try to open it in two different programs, they may both fail to get data. Data will be buffered as long as no programs are reading, so you will receive all unread notifications as soon as you open the file. If you'd like to read notifications from two separate applications, send the command `notifyon <n>` to the keyboard you wish to receive notifications from, where N is a number between 1 and 9. If `/dev/input/ckb*/notify<n>` does not already exist, it will be created, and you can read notifications from there without disrupting any other program. To close a notification node, send `notifyoff <n>`.

`notify0` is always open and will not be affected by `notifyon/notifyoff` commands. By default, all notifications are printed to `notify0`. To print output to a different node, prefix your command with `@<node>`.

Notifications are printed with one notification per line. Commands are as follows:

- `notify <key>:on` or simply `notify <key>` enables notifications for a key. Each key will generate two notifications: `key +<key>` when the key is pressed, and `key -<key>` when it is released.
- `notify <key>:off` turns notifications off for a key.

Examples:

- `notify w a s d` sends notifications whenever W, A, S, or D is pressed.
- `notify g1 g2 g3 g4 g5 g6 g7 g8 g9 g10 g11 g12 g13 g14 g15 g16 g17 g18 mr m1 m2 m3 light lock` prints a notification whenever a non-standard key is pressed.
- `notify all:off` turns all key notifications off.
- `@5 notify esc` prints Esc key notifications to `notify5`.

Note: Key notifications are *not* affected by bindings. For instance, if you run `echo bind a:b notify a >/dev/input/ckb1/cmd` and then press the A key, the notifications will read `key +a key -a`, despite the fact that the character printed on screen will be `b`. Likewise, unbinding a key or assigning a macro to a key does not affect the notifications.

Indicator notifications

You can also choose to receive notifications for the indicator LEDs by using the `inotify` command. For instance, `inotify caps:on` or simply `inotify caps` will print notifications whenever the Caps Lock LED is toggled. The notifications will read `i +caps` when the light is turned on and `i -caps` when it is turned off. It is also possible to toggle all indicators at once using `inotify all` or `inotify all:off`.

Like key notifications, indicator notifications are not affected by bindings, nor by the `ion`, `ioff`, or `iauto` commands. The notifications will reflect the state of the LEDs as seen by the event device.

Getting parameters

Parameters can be retrieved using the `get` command. The data will be sent out as a notification. Generally, the syntax to get the data associated with a command is `get :<command>` (note the colon), and the associated data will be returned in the form of `<command> <data>`. The following data may be gotten:

- `get :mode` returns the current mode in the form of a `switch` command. (Note: Do not use this in a line containing a `mode` command or it will return the mode that you selected, rather than the keyboard's current mode.)
- `get :name` returns the current mode's name in the form of `mode <n> name <name>`. To see the name of another mode, use `mode <n> get :name`. The name is URL-encoded; spaces are written as `%20`. The name may be truncated, so `name <some long string> get :name` may return something shorter than what was entered.
- `get :profilename` returns the profile's name, in the form of `profilename <name>`. As above, it is URL-encoded and may be truncated.
- `get :hwname` and `get :hwprofilename` return the same thing except taken from the current hardware profile instead of the in-memory profile. The output is identical but will read `hwname` instead of `name` and `hwprofilename` instead of `profilename`.
- `get :id` returns the current mode's ID and modification number in the form of `mode <n> id <guid> <modification>`.
- `get :profileid` returns the current profile's ID and modification number in the form of `profileid <guid> <modification>`.
- `get :hwid` and `get :hwprofileid` return the same thing except from the current hardware profile/mode. As before, the output will be the same but with `hwid` and `hwprofileid` instead of `id` and `profileid`.
- `get :rgb` returns an `rgb` command equivalent to the current RGB state.
- `get :hwrngb` does the same thing, but retrieves the colors currently stored in the hardware profile. The output will say `hwrngb` instead of `rgb`.
- `get :dpi` returns a `dpi` command equivalent to the current DPI bank.
- `get :dpisel` returns a `dpisel` command for the currently-selected DPI stage.
- `get :lift` returns a `lift` command for the current lift height.
- `get :snap` returns the current angle snap status.
- `get :hwdpi`, `get :hwdpisel`, `get :hwlift`, and `get :hwsnap` return the same properties, but for the current hardware profile.
- `get :keys` and `get :i` return the current keypress status and indicator status, respectively. They will indicate all currently pressed keys and all currently active indicators, like `key +enter` and `i +num`.

Like `notify`, you must prefix your command with `@<node>` to get data printed to a node other than `notify0`.

Firmware updates

WARNING: Improper use of `fwupdate` may brick your device; use this command *at your own risk*. I accept no responsibility for broken keyboards.

The latest firmware versions and their URLs can be found in the `FIRMWARE` document. To update your keyboard's firmware, first extract the contents of the zip file and then issue the command `fwupdate /path/to/fw/file.bin` to the keyboard you wish to update. The path name must be absolute and must not include spaces. If it succeeded, you should see `fwupdate <path> ok` logged to the keyboard's notification node and then the device will disconnect and reconnect. If you see `fwupdate <path> invalid` it means that the firmware file was not valid for the device; more info may be available in the daemon's `stdout`. If you see `fwupdate <path> fail` it means that the file was valid but the update failed at a hardware level. The keyboard may disconnect/reconnect anyway or it may remain in operation.

When the device reconnects you should see the new firmware version in its `fwversion` node; if you see `0000` instead it means that the keyboard did not update successfully and will need another `fwupdate` command in order to function again. If the update fails repeatedly, try connecting the keyboard to a Windows PC and using the official firmware update in CUE.

Restart

Because sometimes the communication between the daemon and the keyboard is corrupted after resuming from standby or suspend, a restart function is implemented. It first calls the `quit()` function, then it calls `main()` again with the original parameter list.

There are two ways to restart the daemon:

- send the string "restart some-description-as-one-word" to the `cmd-pipe` (normally `/dev/input/ckb1/cmd` or `/dev/input/ckb2/cmd`, depending on what device gets which ID).
- send `SIGUSR1` to the daemon process (as root).

Later on, there may be a user interface in the client for the first method.

Security

By default, all of the `ckb*` nodes may be accessed by any user. For most single-user systems this should not present any security issues, since only one person will have access to the computer anyway. However, if you'd like to restrict the users that can write to the `cmd` nodes or read from the `notify` nodes, you can specify the `--gid=<group>` option at start up. For instance, on most systems you could run `ckb-daemon --gid=1000` to make them accessible only by the system's primary user. `ckb-daemon` must still be run as root, regardless of which `gid` you specify. The `gid` option may be set only at startup and cannot be changed while the daemon is running.

The daemon additionally supports a `--nonotify` option to disable key notifications, to prevent unauthorized programs from logging key input. Note that this will interfere with some of `ckb`'s abilities. It is also highly unlikely to increase security unless you are using the program in a stripped down terminal environment without Xorg. For most use cases there are many other (more likely) ways that a keylogger program could compromise your system. Nevertheless, the option is provided for the sake of paranoia. If you'd like to disable key rebinding as well, launch the daemon with `--nobind`. `--nobind` implies `--nonotify`, so notifications will also be disabled. As with `--gid`, these options must be set at startup and cannot be changed while the daemon is running.

Chapter 5

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 dmutex 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 `inputupdate_keys` (usbdevice *kb)

If we want to get all keys typed while a macro is played, add the code for it here.

Global `macro_pt_enqueue` ()

find a better exit strategy if no more mem available.

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 6

Data Structure Index

6.1 Data Structures

Here are the data structures with brief descriptions:

devcmd.__unnamed__	27
--	----

Chapter 7

File Index

7.1 File List

Here is a list of all files with brief descriptions:

src/ckb-daemon/command.c	31
src/ckb-daemon/command.h	36
src/ckb-daemon/device.c	45
src/ckb-daemon/device.h	49
src/ckb-daemon/device_keyboard.c	59
src/ckb-daemon/device_mouse.c	63
src/ckb-daemon/device_vtable.c	66
src/ckb-daemon/devnode.c	68
src/ckb-daemon/devnode.h	82
src/ckb-daemon/dpi.c	90
src/ckb-daemon/dpi.h	95
src/ckb-daemon/extra_mac.c	100
src/ckb-daemon/firmware.c	100
src/ckb-daemon/firmware.h	105
src/ckb-daemon/includes.h	107
src/ckb-daemon/input.c	111
src/ckb-daemon/input.h	125
src/ckb-daemon/input_linux.c	136
src/ckb-daemon/input_mac.c	143
src/ckb-daemon/input_mac_mouse.c	143
src/ckb-daemon/keymap.c	143
src/ckb-daemon/keymap.h	148
src/ckb-daemon/keymap_mac.h	156
src/ckb-daemon/led.c	157
src/ckb-daemon/led.h	163
src/ckb-daemon/led_keyboard.c	174
src/ckb-daemon/led_mouse.c	207
src/ckb-daemon/main.c	210
src/ckb-daemon/notify.c	219
src/ckb-daemon/notify.h	228
src/ckb-daemon/os.h	232
src/ckb-daemon/profile.c	234
src/ckb-daemon/profile.h	250
src/ckb-daemon/profile_keyboard.c	265
src/ckb-daemon/profile_mouse.c	268
src/ckb-daemon/structures.h	270
src/ckb-daemon/usb.c	289

src/ckb-daemon/ usb.h	
Definitions for using USB interface	307
src/ckb-daemon/ usb_linux.c	346
src/ckb-daemon/ usb_mac.c	372

Chapter 8

Data Structure Documentation

8.1 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\(* updatedpi \)\(usbdevice *kb, int force\)](#)

8.1.1 Detailed Description

Definition at line 78 of file command.h.

8.1.2 Field Documentation

8.1.2.1

8.1.2.2

8.1.2.3

8.1.2.4

8.1.2.5

8.1.2.6

8.1.2.7

8.1.2.8

8.1.2.9

8.1.2.10

8.1.2.11

8.1.2.12

8.1.2.13

8.1.2.14

8.1.2.15

8.1.2.16

8.1.2.17

8.1.2.18

8.1.2.19

8.1.2.20

8.1.2.21

8.1.2.22

8.1.2.23

8.1.2.24

8.1.2.25

8.1.2.26

8.1.2.27

8.1.2.28

8.1.2.29

8.1.2.30

8.1.2.31

8.1.2.32

8.1.2.33

8.1.2.34

8.1.2.35

8.1.2.36

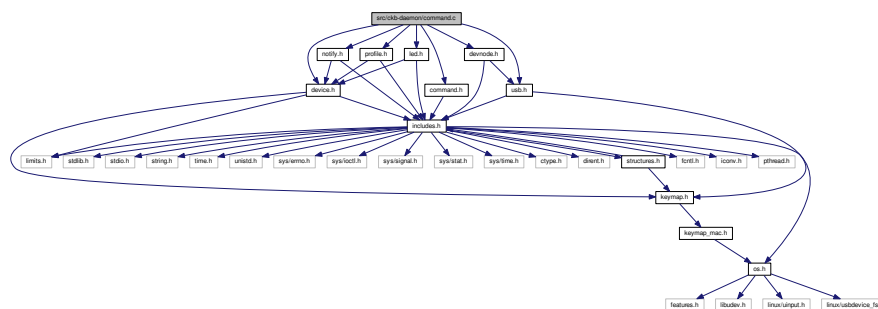
The documentation for this struct was generated from the following files:

File Documentation

9.2 DAEMON.md File Reference

9.4 ROADMAP.md File Reference

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



- #define TRY_WITH_RESET(action)

Functions

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

Variables

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

9.5.1 Macro Definition Documentation

9.5.1.1 #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()`.

9.5.2 Function Documentation

9.5.2.1 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 ==
95             SCROLLSPEED)
96             command = NONE;
97 #endif
98         // Most commands require parameters, but a few are actions in and of themselves
99         if(command != SWITCH
100             && command != HWLOAD && command != HWSAVE
101             && command != ACTIVE && command != IDLE
102             && command != ERASE && command != ERASEPROFILE
103             && command != RESTART)
104             goto next_loop;
105         break;
106     }
107
108     // Set current notification node when given @number
109     int newnotify;
110     if(sscanf(word, "%u", &newnotify) == 1 && newnotify < OUTFIFO_MAX){
111         notifynumber = newnotify;
112         continue;
113     }
114
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
127     // Reject anything not related to fwupdate if device has a bricked FW
128     if(NEEDS_FW_UPDATE(kb) && command != FWUPDATE && command !=
129     NOTIFYON && command != NOTIFYOFF)
130         continue;
131
132     // Specially handled commands - these are available even when keyboard is IDLE
133     switch(command){
134     case NOTIFYON: {
135         // Notification node on
136         int notify;
137         if(sscanf(word, "%u", &notify) == 1)
138             mknotifynode(kb, notify);
139         continue;
140     } case NOTIFYOFF: {
141         // Notification node off
142         int notify;
143         if(sscanf(word, "%u", &notify) == 1 && notify != 0) // notify0 can't be removed
144             rmnotifynode(kb, notify);
145         continue;
146     } case GET:
147         // Output data to notification node
148         vt->get(kb, mode, notifynumber, 0, word);
149         continue;
150     case LAYOUT:
151         // OSX: switch ANSI/ISO keyboard layout
152         if(!strcmp(word, "ansi"))
153             kb->features = (kb->features & ~FEAT_LMASK) |
154             FEAT_ANSI;
155         else if(!strcmp(word, "iso"))
156             kb->features = (kb->features & ~FEAT_LMASK) |
157             FEAT_ISO;
158         continue;
159 #ifdef OS_MAC
160     case ACCEL:
161         // OSX mouse acceleration on/off
162         if(!strcmp(word, "on"))
163             kb->features |= FEAT_MOUSEACCEL;
164         else if(!strcmp(word, "off"))
165             kb->features &= ~FEAT_MOUSEACCEL;
166         continue;
167     case SCROLLSPEED: {
168         int newscroll;
169         if(sscanf(word, "%d", &newscroll) != 1)
170             break;
171         if(newscroll < SCROLL_MIN)
172             newscroll = SCROLL_ACCELERATED;
173         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         long int delay;
205         if(sscanf(word, "%ld", &delay) == 1 && 0 <= delay && delay < UINT_MAX) {
206             // Add delay of 'newdelay' microseconds to macro playback
207             kb->delay = (unsigned int)delay;
208         } else if(strcmp(word, "on") == 0) {
209             // allow previous syntax, 'delay on' means use old 'long macro delay'
210             kb->delay = UINT_MAX;
211         } else {
212             // bad parameter to handle false commands like "delay off"
213             kb->delay = 0; // No delay.
214         }
215         continue;
216     }
217     case RESTART: {
218         char mybuffer[] = "no reason specified";
219         if (sscanf(line, "%[^\\n]", word) == -1) {
220             word = mybuffer;
221         }
222         vt->do_cmd[command](kb, mode, notifiynumber, 0, word);
223         continue;
224     }
225     default:;
226 }
227
228 // If a keyboard is inactive, it must be activated before receiving any other commands
229 if(!kb->active){
230     if(command == ACTIVE)
231         TRY_WITH_RESET(vt->active(kb, mode, notifiynumber, 0, 0));
232     continue;
233 }
234 // Specially handled commands only available when keyboard is ACTIVE
235 switch(command){
236 case IDLE:
237     TRY_WITH_RESET(vt->idle(kb, mode, notifiynumber, 0, 0));
238     continue;
239 case SWITCH:
240     if(profile->currentmode != mode){
241         profile->currentmode = mode;
242         // Set mode light for non-RGB K95
243         int index = INDEX_OF(mode, profile->mode);
244         vt->setmodeindex(kb, index);
245     }
246     continue;
247 case HWLOAD: case HWSAVE:{
248     char delay = kb->usbdelay;
249     // Ensure delay of at least 10ms as the device can get overwhelmed otherwise
250     if(delay < 10)

```



```

251         kb->usbdelay = 10;
252         // Try to load/save the hardware profile. Reset on failure, disconnect if reset fails.
253         TRY_WITH_RESET(vt->do_io[command](kb, mode, notifynumber, 1, 0));
254         // Re-send the current RGB state as it sometimes gets scrambled
255         TRY_WITH_RESET(vt->updatergb(kb, 1));
256         kb->usbdelay = delay;
257         continue;
258     }
259     case FWUPDATE:
260         // FW update parses a whole word. Unlike hwload/hwsave, there's no try again on failure.
261         if(vt->fwupdate(kb, mode, notifynumber, 0, word)){
262             free(word);
263             return 1;
264         }
265         continue;
266     case POLLRATE: {
267         uint rate;
268         if(sscanf(word, "%u", &rate) == 1 && (rate == 1 || rate == 2 || rate == 4 || rate == 8))
269             TRY_WITH_RESET(vt->pollrate(kb, mode, notifynumber, rate, 0));
270         continue;
271     }
272     case ERASEPROFILE:
273         // Erase the current profile
274         vt->eraseprofile(kb, mode, notifynumber, 0, 0);
275         // Update profile/mode pointers
276         profile = kb->profile;
277         mode = profile->currentmode;
278         continue;
279     case ERASE: case NAME: case IOFF: case ION: case IAUTO: case
INOTIFY: case PROFILENAME: case ID: case PROFILEID: case
DPISEL: case LIFT: case SNAP:
280         // All of the above just parse the whole word
281         vt->do_cmd[command](kb, mode, notifynumber, 0, word);
282         continue;
283     case RGB: {
284         // RGB command has a special response for a single hex constant
285         int r, g, b;
286         if(sscanf(word, "%02x%02x%02x", &r, &g, &b) == 3){
287             // Set all keys
288             for(int i = 0; i < N_KEYS_EXTENDED; i++)
289                 vt->rgb(kb, mode, notifynumber, i, word);
290             continue;
291         }
292         break;
293     }
294     case MACRO:
295         if(!strcmp(word, "clear")){
296             // Macro has a special clear command
297             vt->macro(kb, mode, notifynumber, 0, 0);
298             continue;
299         }
300         break;
301     case RESTART:
302         vt->do_cmd[command](kb, mode, notifynumber, 0, word);
303         break;
304     default:;
305     }
306     // For anything else, split the parameter at the colon
307     int left = -1;
308     sscanf(word, "%*[^:]%n", &left);
309     if(left <= 0)
310         continue;
311     const char* right = word + left;
312     if(right[0] == ':')
313         right++;
314     // Macros and DPI have a separate left-side handler
315     if(command == MACRO || command == DPI){
316         word[left] = 0;
317         vt->do_macro[command](kb, mode, notifynumber, word, right);
318         continue;
319     }
320     // Scan the left side for key names and run the requested command
321     int position = 0, field = 0;
322     char keyname[11];
323     while(position < left && sscanf(word + position, "%10[^:,%n", keyname, &field) == 1){
324         int keycode;
325         if(!strcmp(keyname, "all")){
326             // Set all keys
327             for(int i = 0; i < N_KEYS_EXTENDED; i++)
328                 vt->do_cmd[command](kb, mode, notifynumber, i, right);
329         } else if((sscanf(keyname, "%d", &keycode) && keycode >= 0 && keycode <
N_KEYS_EXTENDED)
330 || (sscanf(keyname, "%x%x", &keycode) && keycode >= 0 && keycode <
N_KEYS_EXTENDED)){
331             // Set a key numerically
332             vt->do_cmd[command](kb, mode, notifynumber, keycode, right);
333         } else {

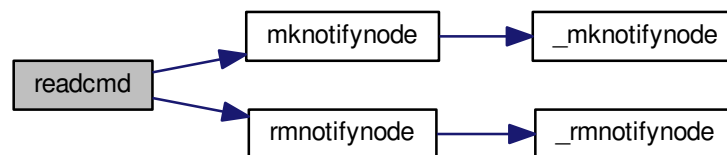
```

```

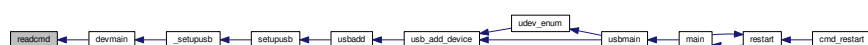
334         // Find this key in the keymap
335         for(unsigned i = 0; i < N_KEYS_EXTENDED; i++){
336             if(keymap[i].name && !strcmp(keyname, keymap[i].name)){
337                 vt->do_cmd[command](kb, mode, notifynumber, i, right);
338                 break;
339             }
340         }
341     }
342     if(word[position += field] == ',')
343         position++;
344 }
345 }
346
347 // Finish up
348 if(!NEEDS_FW_UPDATE(kb)){
349     TRY_WITH_RESET(vt->updatergb(kb, 0));
350     TRY_WITH_RESET(vt->updatedpi(kb, 0));
351 }
352 free(word);
353 return 0;
354 }

```

Here is the call graph for this function:



Here is the caller graph for this function:



9.5.3 Variable Documentation

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

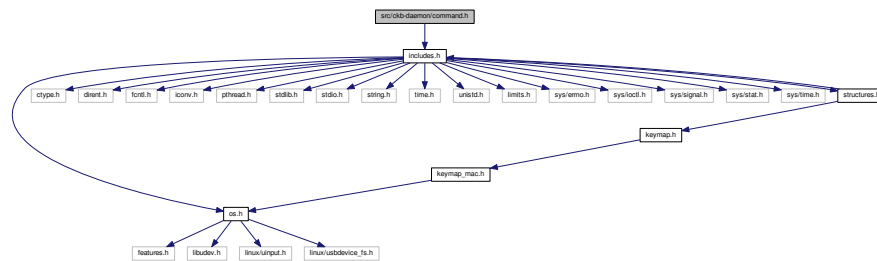
Definition at line 9 of file `command.c`.

Referenced by `readcmd()`.

9.6 `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

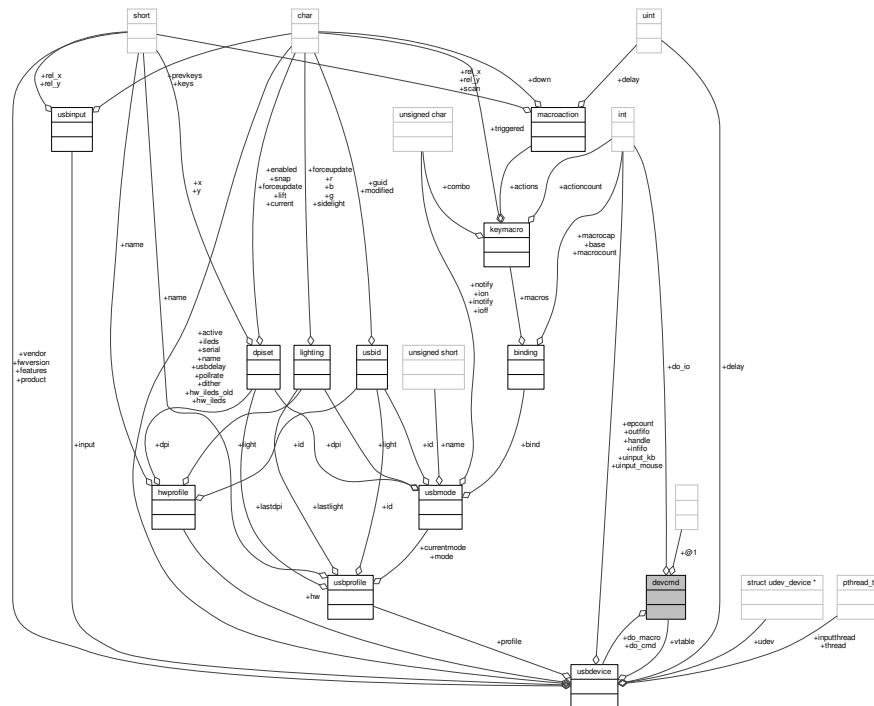
- int `readcmd` (`usbdevice` *`kb`, const char *`line`)

9.6.1 Data Structure Documentation

9.6.1.1 union devcmd

Definition at line 73 of file `command.h`.

Collaboration diagram for `devcmd`:



Data Fields

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

9.6.2 Macro Definition Documentation

9.6.2.1 #define CMD_COUNT (CMD_LAST - CMD_FIRST + 2)

Definition at line 65 of file `command.h`.

Referenced by `readcmd()`.

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

Definition at line 66 of file command.h.

9.6.3 Typedef Documentation

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

Definition at line 70 of file command.h.

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

Definition at line 71 of file command.h.

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

Definition at line 72 of file command.h.

9.6.3.4 `typedef union devcmd devcmd`

9.6.4 Enumeration Type Documentation

9.6.4.1 `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;

```

9.6.5 Function Documentation

9.6.5.1 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, DPISEL, ERASE, ERASEPROFILE, FEAT_ANSI, FEAT_BIND, FEAT_ISO, FEAT_LMASK, FEAT_MOUSEACCEL, FEAT_NOTIFY, usbdevice::features, lighting::forceupdate, FPS, FWUPDATE, GET, HAS_FEATURES, HWLOAD, HWSAVE, IAUTO, ID, IDLE, INDEX_OF, INOTIFY, IOFF, ION, IS_FULLRANGE, IS_MOUSE_DEV, keymap, LAYOUT, LIFT, usbmode::light, MACRO, mknotifynode(), MODE, usbprofile::mode, MODE_COUNT, N_KEYS_EXTENDED, NAME, NEEDS_FW_UPDATE, NONE, NOTIFY, NOTIFYOFF, NOTIFYON, OUTFIFO_MAX, POLLRATE, usbdevice::profile, PROFILEID, PROFILENAME, REBIND, RESTART, RGB, rmnotifynode(), SCROLL_ACCELERATED, SCROLL_MAX, SCROLL_MIN, SCROLLSPEED, SNAP, SWITCH, TRY_WITH_RESET, UNBIND, usbdevice::usbdelay, and usbdevice::vtable.

Referenced by devmain().

```

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

```

```

103         goto next_loop;
104         break;
105     }
106 }
107
108 // Set current notification node when given @number
109 int newnotify;
110 if(sscanf(word, "%u", &newnotify) == 1 && newnotify < OUTFIFO_MAX){
111     notifynumber = newnotify;
112     continue;
113 }
114
115 // Reject unrecognized commands. Reject bind or notify related commands if the keyboard doesn't
116 // have the feature enabled.
117 if(command == NONE
118     || ((!HAS_FEATURES(kb, FEAT_BIND) && (command ==
119     BIND || command == UNBIND || command == REBIND || command ==
120     MACRO || command == DELAY))
121     || (!HAS_FEATURES(kb, FEAT_NOTIFY) && command ==
122     NOTIFY))) {
123     next_loop:
124     continue;
125 }
126 // Reject anything not related to fwupdate if device has a bricked FW
127 if(NEEDS_FW_UPDATE(kb) && command != FWUPDATE && command !=
128     NOTIFYON && command != NOTIFYOFF)
129     continue;
130
131 // Specially handled commands - these are available even when keyboard is IDLE
132 switch(command) {
133 case NOTIFYON: {
134     // Notification node on
135     int notify;
136     if(sscanf(word, "%u", &notify) == 1)
137         mknotifynode(kb, notify);
138     continue;
139 } case NOTIFYOFF: {
140     // Notification node off
141     int notify;
142     if(sscanf(word, "%u", &notify) == 1 && notify != 0) // notify0 can't be removed
143         rmnotifynode(kb, notify);
144     continue;
145 } case GET:
146     // Output data to notification node
147     vt->get(kb, mode, notifynumber, 0, word);
148     continue;
149 case LAYOUT:
150     // OSX: switch ANSI/ISO keyboard layout
151     if(!strcmp(word, "ansi"))
152         kb->features = (kb->features & ~FEAT_LMASK) |
153         FEAT_ANSI;
154     else if(!strcmp(word, "iso"))
155         kb->features = (kb->features & ~FEAT_LMASK) |
156         FEAT_ISO;
157     continue;
158 #ifdef OS_MAC
159 case ACCEL:
160     // OSX mouse acceleration on/off
161     if(!strcmp(word, "on"))
162         kb->features |= FEAT_MOUSEACCEL;
163     else if(!strcmp(word, "off"))
164         kb->features &= ~FEAT_MOUSEACCEL;
165     continue;
166 case SCROLLSPEED: {
167     int newscroll;
168     if(sscanf(word, "%d", &newscroll) != 1)
169         break;
170     if(newscroll < SCROLL_MIN)
171         newscroll = SCROLL_ACCELERATED;
172     if(newscroll > SCROLL_MAX)
173         newscroll = SCROLL_MAX;
174     kb->scroll_rate = newscroll;
175     continue;
176 }
177 #endif
178 case MODE: {
179     // Select a mode number (1 - 6)
180     int newmode;
181     if(sscanf(word, "%u", &newmode) == 1 && newmode > 0 && newmode <=
182     MODE_COUNT)
183         mode = profile->mode + newmode - 1;
184     continue;
185 }
186 case FPS: {
187     // USB command delay (2 - 10ms)
188     uint framerate;
189     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     long int delay;
205     if(sscanf(word, "%ld", &delay) == 1 && 0 <= delay && delay < UINT_MAX) {
206         // Add delay of 'newdelay' microseconds to macro playback
207         kb->delay = (unsigned int)delay;
208     } else if(strcmp(word, "on") == 0) {
209         // allow previous syntax, 'delay on' means use old 'long macro delay'
210         kb->delay = UINT_MAX;
211     } else {
212         // bad parameter to handle false commands like "delay off"
213         kb->delay = 0; // No delay.
214     }
215     continue;
216 }
217 case RESTART: {
218     char mybuffer[] = "no reason specified";
219     if (sscanf(line, "%[^\n]", word) == -1) {
220         word = mybuffer;
221     }
222     vt->do_cmd[command](kb, mode, notifiynumber, 0, word);
223     continue;
224 }
225 default;;
226 }
227
228 // If a keyboard is inactive, it must be activated before receiving any other commands
229 if(!kb->active){
230     if(command == ACTIVE)
231         TRY_WITH_RESET(vt->active(kb, mode, notifiynumber, 0, 0));
232     continue;
233 }
234 // Specially handled commands only available when keyboard is ACTIVE
235 switch(command){
236 case IDLE:
237     TRY_WITH_RESET(vt->idle(kb, mode, notifiynumber, 0, 0));
238     continue;
239 case SWITCH:
240     if(profile->currentmode != mode){
241         profile->currentmode = mode;
242         // Set mode light for non-RGB K95
243         int index = INDEX_OF(mode, profile->mode);
244         vt->setmodeindex(kb, index);
245     }
246     continue;
247 case HWLOAD: case HWSAVE:{
248     char delay = kb->usbdelay;
249     // Ensure delay of at least 10ms as the device can get overwhelmed otherwise
250     if(delay < 10)
251         kb->usbdelay = 10;
252     // Try to load/save the hardware profile. Reset on failure, disconnect if reset fails.
253     TRY_WITH_RESET(vt->do_io[command](kb, mode, notifiynumber, 1, 0));
254     // Re-send the current RGB state as it sometimes gets scrambled
255     TRY_WITH_RESET(vt->updatergb(kb, 1));
256     kb->usbdelay = delay;
257     continue;
258 }
259 case FWUPDATE:
260     // FW update parses a whole word. Unlike hwload/hwsave, there's no try again on failure.
261     if(vt->fwupdate(kb, mode, notifiynumber, 0, word)){
262         free(word);
263         return 1;
264     }
265     continue;
266 case POLLRATE: {
267     uint rate;

```

```

268         if(sscanf(word, "%u", &rate) == 1 && (rate == 1 || rate == 2 || rate == 4 || rate == 8))
269             TRY_WITH_RESET(vt->pollrate(kb, mode, notifynumber, rate, 0));
270         continue;
271     }
272     case ERASEPROFILE:
273         // Erase the current profile
274         vt->eraseprofile(kb, mode, notifynumber, 0, 0);
275         // Update profile/mode pointers
276         profile = kb->profile;
277         mode = profile->currentmode;
278         continue;
279     case ERASE: case NAME: case IOFF: case ION: case IAUTO: case
INOTIFY: case PROFILENAME: case ID: case PROFILEID: case
DPISEL: case LIFT: case SNAP:
280         // All of the above just parse the whole word
281         vt->do_cmd[command](kb, mode, notifynumber, 0, word);
282         continue;
283     case RGB: {
284         // RGB command has a special response for a single hex constant
285         int r, g, b;
286         if(sscanf(word, "%02x%02x%02x", &r, &g, &b) == 3){
287             // Set all keys
288             for(int i = 0; i < N_KEYS_EXTENDED; i++){
289                 vt->rgb(kb, mode, notifynumber, i, word);
290             }
291             break;
292         }
293     }
294     case MACRO:
295         if(!strcmp(word, "clear")){
296             // Macro has a special clear command
297             vt->macro(kb, mode, notifynumber, 0, 0);
298             continue;
299         }
300         break;
301     case RESTART:
302         vt->do_cmd[command](kb, mode, notifynumber, 0, word);
303         break;
304     default:;
305     }
306     // For anything else, split the parameter at the colon
307     int left = -1;
308     sscanf(word, "%*[^:]%n", &left);
309     if(left <= 0)
310         continue;
311     const char* right = word + left;
312     if(right[0] == ':')
313         right++;
314     // Macros and DPI have a separate left-side handler
315     if(command == MACRO || command == DPI){
316         word[left] = 0;
317         vt->do_macro[command](kb, mode, notifynumber, word, right);
318         continue;
319     }
320     // Scan the left side for key names and run the requested command
321     int position = 0, field = 0;
322     char keyname[11];
323     while(position < left && sscanf(word + position, "%10[^:]%n", keyname, &field) == 1){
324         int keycode;
325         if(!strcmp(keyname, "all")){
326             // Set all keys
327             for(int i = 0; i < N_KEYS_EXTENDED; i++){
328                 vt->do_cmd[command](kb, mode, notifynumber, i, right);
329             } else if((sscanf(keyname, "%d", &keycode) && keycode >= 0 && keycode <
N_KEYS_EXTENDED)
330                 || (sscanf(keyname, "%x%x", &keycode) && keycode >= 0 && keycode <
N_KEYS_EXTENDED)){
331                 // Set a key numerically
332                 vt->do_cmd[command](kb, mode, notifynumber, keycode, right);
333             } else {
334                 // Find this key in the keymap
335                 for(unsigned i = 0; i < N_KEYS_EXTENDED; i++){
336                     if(keymap[i].name && !strcmp(keyname, keymap[i].name)){
337                         vt->do_cmd[command](kb, mode, notifynumber, i, right);
338                         break;
339                     }
340                 }
341             }
342             if(word[position += field] == ':')
343                 position++;
344         }
345     }
346
347     // Finish up
348     if(!NEEDS_FW_UPDATE(kb)){
349         TRY_WITH_RESET(vt->updatergb(kb, 0));
350         TRY_WITH_RESET(vt->updatedpi(kb, 0));

```

Here is the call graph for this function:



Include dependency graph for device.c:



- Generated on Mon Jun 5 2017 23:09:31 for ckb-next by Doxygen

Variables

- int `hwload_mode` = 1
hwload_mode = 1 means read hardware once. should be enough
- usbdevice keyboard [9]
remember all usb devices. Needed for `closeusb()`.
- pthread_mutex_t `devlistmutex` = PTHREAD_MUTEX_INITIALIZER
- pthread_mutex_t `devmutex` [9] = { [0 ... 9 -1] = PTHREAD_MUTEX_INITIALIZER }
Mutex for handling the usbdevice structure.
- pthread_mutex_t `inputmutex` [9] = { [0 ... 9 -1] = PTHREAD_MUTEX_INITIALIZER }
Mutex for dealing with usb input frames.
- pthread_mutex_t `macromutex` [9] = { [0 ... 9 -1] = PTHREAD_MUTEX_INITIALIZER }
Protecting macros against lightning: Both use `usb_send`.
- pthread_mutex_t `macromutex2` [9] = { [0 ... 9 -1] = PTHREAD_MUTEX_INITIALIZER }
Protecting the single link list of threads and the macrovar.
- pthread_cond_t `macrovar` [9] = { [0 ... 9 -1] = PTHREAD_COND_INITIALIZER }
This variable is used to stop and wakeup all macro threads which have to wait.

9.7.1 Function Documentation

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

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

Parameters

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

Returns

0 if success, other else

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

Definition at line 25 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()`.

```

25     {
26     // Get the firmware version from the device
27     if (kb->pollrate == 0){
35         if (!hwload_mode || (HAS_FEATURES(kb, FEAT_HWLOAD) &&
getfwversion(kb))) {
36             if (hwload_mode == 2)
37                 // hwload=always. Report setup failure.
38                 return -1;
39             else if (hwload_mode){
40                 // hwload=once. Log failure, prevent trying again, and continue.
41                 ckb_warn("Unable to load firmware version/poll rate\n");
42                 kb->features &= ~FEAT_HWLOAD;
43             }
44             kb->pollrate = 0;
45             kb->features &= ~(FEAT_POLLRATE | FEAT_ADJRATE);
46             if (kb->fwversion == 0)
47                 kb->features &= ~(FEAT_FWVERSION |
FEAT_FWUPDATE);
48         }
49     }
54     if (NEEDS_FW_UPDATE(kb)){
56         ckb_info("Device needs a firmware update. Please issue a fwupdate command.\n");
57         kb->features = FEAT_RGB | FEAT_FWVERSION |
FEAT_FWUPDATE;
58         kb->active = 1;
59         return 0;
60     }
66     if (!kb->hw && hwload_mode && HAS_FEATURES(kb,
FEAT_HWLOAD)) {
67         if (hwloadprofile(kb, 1)){
68             if (hwload_mode == 2)
69                 return -1;
70             ckb_warn("Unable to load hardware profile\n");
71             kb->features &= ~FEAT_HWLOAD;
72         }
73     }
74     // Active software mode if requested
75     if (makeactive)
76         return setactive(kb, 1);
77     return 0;
78 }

```

Here is the call graph for this function:



Here is the caller graph for this function:



9.7.1.2 int start_dev (usbdevice * kb, int makeactive)

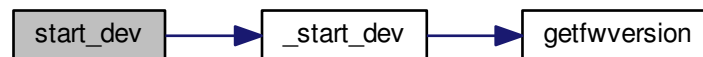
Definition at line 80 of file device.c.

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

```

80                                     {
81     // Force USB interval to 10ms during initial setup phase; return to nominal 5ms after setup completes.
82     kb->usbdelay = 10;
83     int res = _start_dev(kb, makeactive);
84     kb->usbdelay = USB_DELAY_DEFAULT;
85     return res;
86 }
```

Here is the call graph for this function:



9.7.2 Variable Documentation

9.7.2.1 `pthread_mutex_t devlistmutex = PTHREAD_MUTEX_INITIALIZER`

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

9.7.2.2 `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()`.

9.7.2.3 `int hwload_mode = 1`

`hwload_mode` is defined in [device.c](#)

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

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

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

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

9.7.2.5 `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()`.

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

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

9.7.2.7 pthread_mutex_t macromutex2[9] = { [0 ... 9 -1] = PTHREAD_MUTEX_INITIALIZER }

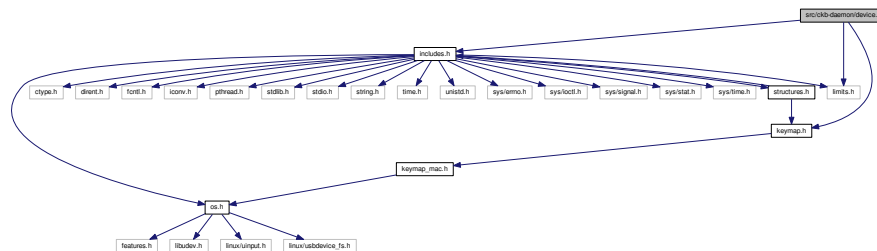
Definition at line 15 of file device.c.

9.7.2.8 pthread_cond_t macrovar[9] = { [0 ... 9 -1] = PTHREAD_COND_INITIALIZER }

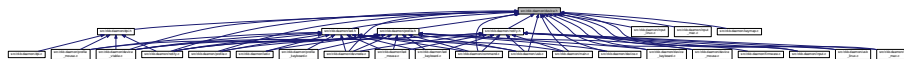
Definition at line 16 of file device.c.

9.8 src/ckb-daemon/device.h File Reference

```
#include "includes.h"
#include "keymap.h"
#include <limits.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 mmutex(kb) (macromutex + INDEX_OF(kb, keyboard))`
- `#define mmutex2(kb) (macromutex2 + INDEX_OF(kb, keyboard))`
- `#define mvar(kb) (macrovar + 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]
remember all usb devices. Needed for `closeusb()`.
- `pthread_mutex_t devmutex` [9]
Mutex for handling the usbdevice structure.
- `pthread_mutex_t inputmutex` [9]
Mutex for dealing with usb input frames.
- `pthread_mutex_t macromutex` [9]
Protecting macros against lightning: Both use `usb_send`.
- `pthread_mutex_t macromutex2` [9]
Protecting the single link list of threads and the macrovar.
- `pthread_cond_t macrovar` [9]
This variable is used to stop and wakeup all macro threads which have to wait.

9.8.1 Macro Definition Documentation

9.8.1.1 `#define ACT_LIGHT 1`

Definition at line 68 of file `device.h`.

Referenced by `setactive_kb()`.

9.8.1.2 `#define ACT_LOCK 8`

Definition at line 71 of file `device.h`.

Referenced by `setactive_kb()`.

9.8.1.3 `#define ACT_M1 10`

Definition at line 73 of file `device.h`.

Referenced by `setactive_kb()`.

9.8.1.4 `#define ACT_M2 11`

Definition at line 74 of file device.h.

Referenced by `setactive_kb()`.

9.8.1.5 `#define ACT_M3 12`

Definition at line 75 of file device.h.

Referenced by `setactive_kb()`.

9.8.1.6 `#define ACT_MR_RING 9`

Definition at line 72 of file device.h.

Referenced by `setactive_kb()`.

9.8.1.7 `#define ACT_NEXT 3`

Definition at line 69 of file device.h.

9.8.1.8 `#define ACT_NEXT_NOWRAP 5`

Definition at line 70 of file device.h.

9.8.1.9 `#define DEV_MAX 9`

Definition at line 8 of file device.h.

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

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

Definition at line 18 of file device.h.

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

9.8.1.11 `#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()`.

9.8.1.12 `#define IN_CORSAIR 0x40`

Definition at line 65 of file device.h.

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

9.8.1.13 `#define IN_HID 0x80`

Definition at line 64 of file device.h.

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

9.8.1.14 `#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()`.

9.8.1.15 `#define mmutex(kb) (macromutex + INDEX_OF(kb, keyboard))`

Definition at line 26 of file device.h.

Referenced by `_usbrecv()`, `_usbseend()`, and `play_macro()`.

9.8.1.16 `#define mmutex2(kb) (macromutex2 + INDEX_OF(kb, keyboard))`

Definition at line 28 of file device.h.

Referenced by `play_macro()`.

9.8.1.17 `#define mvar(kb) (macrovar + INDEX_OF(kb, keyboard))`

Definition at line 30 of file device.h.

Referenced by `play_macro()`.

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

Definition at line 44 of file device.h.

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

9.8.2 Function Documentation

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

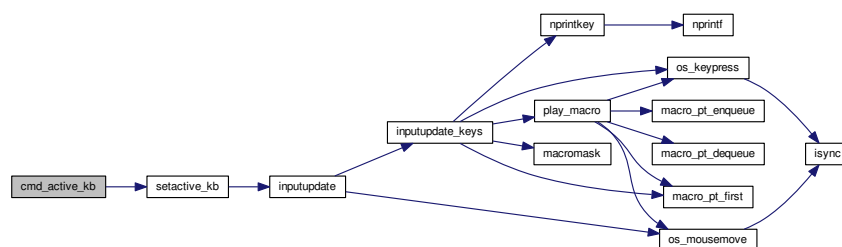
Definition at line 112 of file device_keyboard.c.

References `setactive_kb()`.

```

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

Here is the call graph for this function:



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

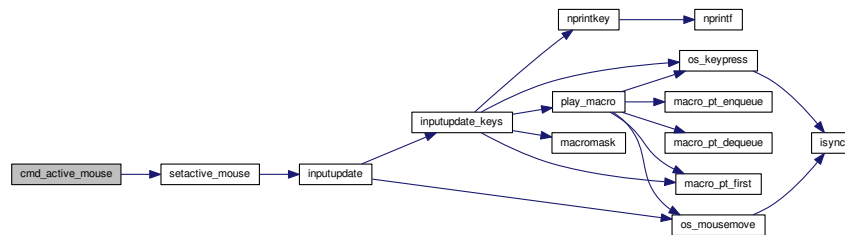
Definition at line 44 of file device_mouse.c.

References `setactive_mouse()`.

```

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

Here is the call graph for this function:



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

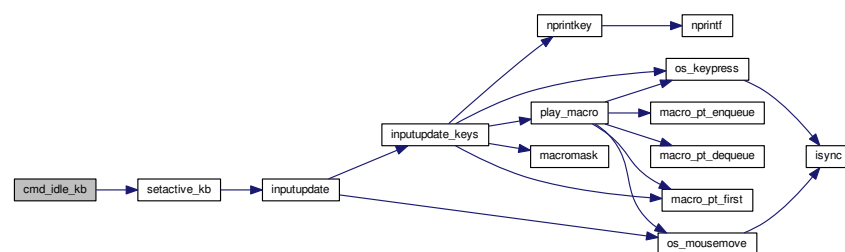
Definition at line 116 of file device_keyboard.c.

References `setactive_kb()`.

```

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

Here is the call graph for this function:



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

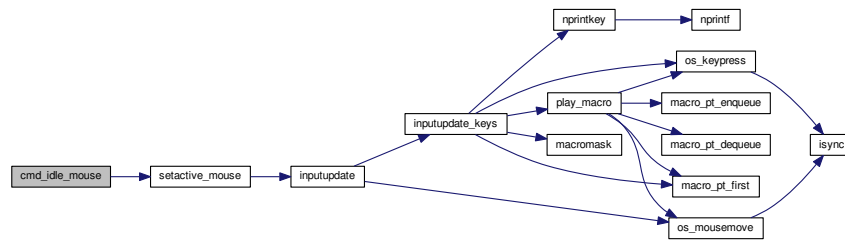
Definition at line 48 of file device_mouse.c.

References `setactive_mouse()`.

```

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

Here is the call graph for this function:



9.8.2.5 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 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 field in case it doesn't
59     kb->pollrate = rate;
60     return 0;
61 }

```

9.8.2.6 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, usbinput::keys, usbprofile::lastlight, MSG_SIZE, N_KEYS_HW, NEEDS_FW_UPDATE, usbdevice::profile, usbsend, 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(!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.

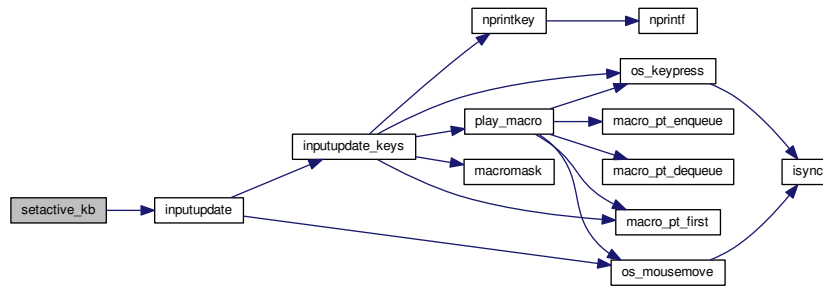
```

```

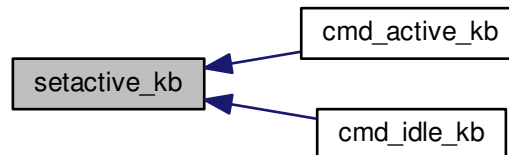
43     for(int key = 0; key < N_KEYS_HW; ){
44         int pair;
45         for(pair = 0; pair < 30 && key < N_KEYS_HW; pair++, key++){
46             // Select both standard and Corsair input. The standard input will be ignored except in
47             BIOS mode.
48             uchar action = IN_HID | IN_CORSAIR;
49             // Additionally, make MR activate the MR ring (this is disabled for now, may be back later)
50             //if(keymap[key].name && !strcmp(keymap[key].name, "mr"))
51             //    action |= ACT_MR_RING;
52             msg[1][4 + pair * 2] = key;
53             msg[1][5 + pair * 2] = action;
54         }
55         // Byte 2 = pair count (usually 30, less on final message)
56         msg[1][2] = pair;
57         if(!usbSend(kb, msg[1], 1))
58             return -1;
59     }
60     // Commit new input settings
61     if(!usbSend(kb, msg[2], 1))
62         return -1;
63     DELAY_MEDIUM(kb);
64 } else {
65     // Set the M-keys back into hardware mode, restore hardware RGB profile. It has to be sent twice
66     for some reason.
67     msg[0][2] = 1;
68     if(!usbSend(kb, msg[0], 1))
69         return -1;
70     DELAY_MEDIUM(kb);
71     if(!usbSend(kb, msg[0], 1))
72         return -1;
73     DELAY_MEDIUM(kb);
74 #ifdef OS_LINUX
75     // On OSX the default key mappings are fine. On Linux, the G keys will freeze the keyboard. Set the
76     keyboard entirely to HID input.
77     for(int key = 0; key < N_KEYS_HW; ){
78         int pair;
79         for(pair = 0; pair < 30 && key < N_KEYS_HW; pair++, key++){
80             uchar action = IN_HID;
81             // Enable hardware actions
82             if(keymap[key].name){
83                 if(!strcmp(keymap[key].name, "mr"))
84                     action = ACT_MR_RING;
85                 else if(!strcmp(keymap[key].name, "m1"))
86                     action = ACT_M1;
87                 else if(!strcmp(keymap[key].name, "m2"))
88                     action = ACT_M2;
89                 else if(!strcmp(keymap[key].name, "m3"))
90                     action = ACT_M3;
91                 else if(!strcmp(keymap[key].name, "light"))
92                     action = ACT_LIGHT;
93                 else if(!strcmp(keymap[key].name, "lock"))
94                     action = ACT_LOCK;
95             }
96             msg[1][4 + pair * 2] = key;
97             msg[1][5 + pair * 2] = action;
98         }
99         // Byte 2 = pair count (usually 30, less on final message)
100         msg[1][2] = pair;
101         if(!usbSend(kb, msg[1], 1))
102             return -1;
103     }
104     // Commit new input settings
105     if(!usbSend(kb, msg[2], 1))
106         return -1;
107     DELAY_MEDIUM(kb);
108 #endif
109 }
110 // Update indicator LEDs if the profile contains settings for them
111 kb->vtable->updateIndicators(kb, 0);
112 return 0;
113 }

```

Here is the call graph for this function:



Here is the caller graph for this function:



9.8.2.7 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 }, // 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){

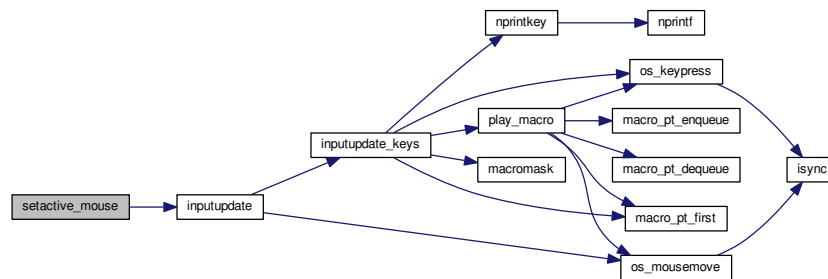
```

```

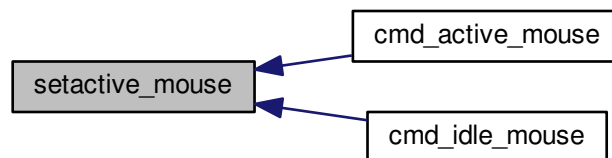
33     // Set up key input
34     if(!usbSend(kb, msg[1], 1))
35         return -1;
36     for(int i = 0; i < keycount; i++){
37         msg[1][i * 2 + 4] = i + 1;
38         msg[1][i * 2 + 5] = (i < 6 ? IN_HID : IN_CORSAIR);
39     }
40 }
41 return 0;
42 }

```

Here is the call graph for this function:



Here is the caller graph for this function:



9.8.2.8 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 }

```

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

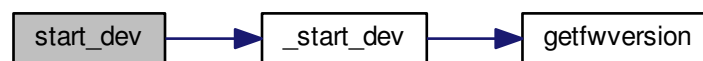
Definition at line 80 of file device.c.

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

```

80                                     {
81     // Force USB interval to 10ms during initial setup phase; return to nominal 5ms after setup completes.
82     kb->usbdelay = 10;
83     int res = _start_dev(kb, makeactive);
84     kb->usbdelay = USB_DELAY_DEFAULT;
85     return res;
86 }
```

Here is the call graph for this function:



9.8.2.10 `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 }
```

9.8.3 Variable Documentation

9.8.3.1 `pthread_mutex_t devmutex[9]`

Definition at line 12 of file device.c.

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

9.8.3.2 `pthread_mutex_t inputmutex[9]`

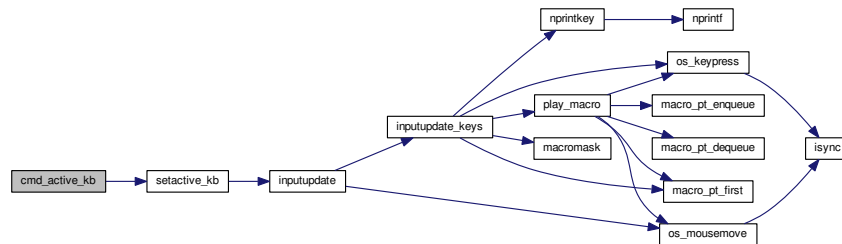
Definition at line 13 of file device.c.

9.8.3.3 `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()`.

Here is the call graph for this function:



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

Definition at line 116 of file device_keyboard.c.

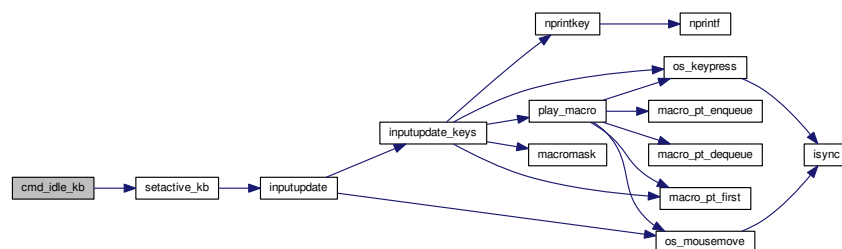
References setactive_kb().

```

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

```

Here is the call graph for this function:



9.9.1.3 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::vtable, 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);

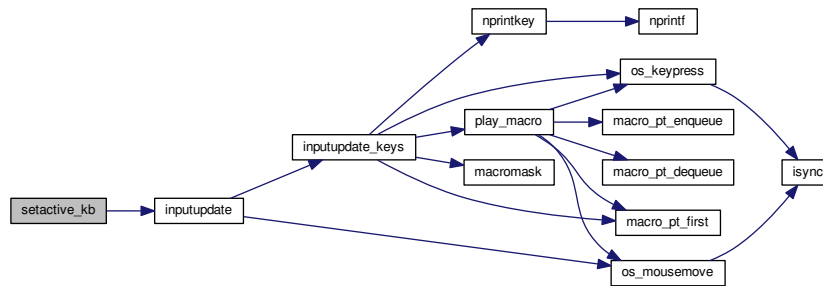
```

```

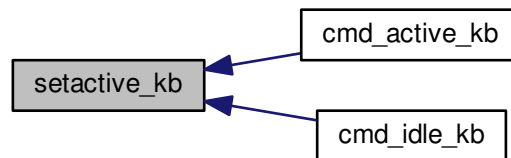
28 pthread_mutex_unlock(imutex(kb));
29
30 uchar msg[3][MSG_SIZE] = {
31     { 0x07, 0x04, 0 }, // Disables or enables HW control for top row
32     { 0x07, 0x40, 0 }, // Selects key input
33     { 0x07, 0x05, 2, 0, 0x03, 0x00 } // Commits key input selection
34 };
35 if(active){
36     // Put the M-keys (K95) as well as the Brightness/Lock keys into software-controlled mode.
37     msg[0][2] = 2;
38     if(!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:



9.9.1.4 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 }

```

9.9.1.5 int start_kb_nrgb (usbdevice * kb, int makeactive)

Definition at line 9 of file device_keyboard.c.

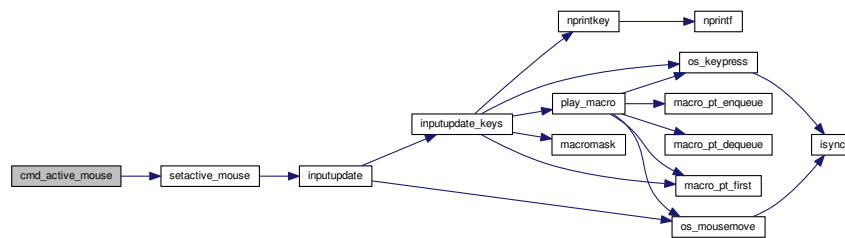
References usbdevice::active, NK95_HWOFF, nk95cmd, and usbdevice::pollrate.

```

9                                     {
10     // Put the non-RGB K95 into software mode. Nothing else needs to be done hardware wise
11     nk95cmd(kb, NK95_HWOFF);

```


Here is the call graph for this function:



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

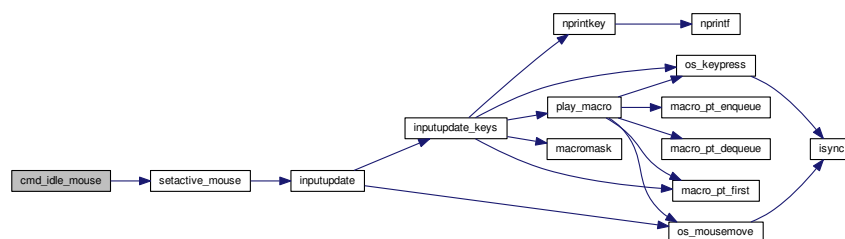
Definition at line 48 of file device_mouse.c.

References setactive_mouse().

```

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

Here is the call graph for this function:



9.10.1.3 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 field in case it doesn't
59     kb->pollrate = rate;
60     return 0;
61 }
  
```

9.10.1.4 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.

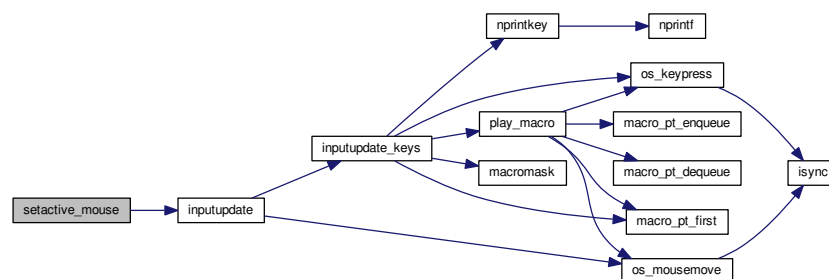
Referenced by cmd_active_mouse(), and cmd_idle_mouse().

```

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 },
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:



- const [devcmd vtable_keyboard_norrgb](#)
- const [devcmd vtable_mouse](#)

9.11.1 Function Documentation

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

Definition at line 13 of file device_vtable.c.

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

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

Definition at line 16 of file device_vtable.c.

```
16     {  
17 }
```

9.11.1.3 `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 }
```

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

Definition at line 23 of file device_vtable.c.

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

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

Definition at line 21 of file device_vtable.c.

```
21                                     {  
22 }
```

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

Definition at line 18 of file device_vtable.c.

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


Functions

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

Variables

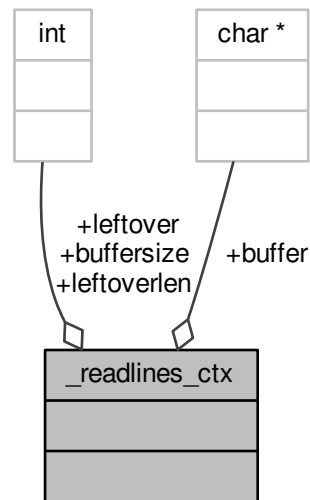
- const char *const [devpath](#) = "/dev/input/ckb"
- long [gid](#) = -1
Group ID for the control nodes. -1 to give read/write access to everybody.

9.12.1 Data Structure Documentation

9.12.1.1 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	

9.12.2 Macro Definition Documentation

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

Definition at line 334 of file `devnode.c`.

Referenced by `readlines()`.

9.12.2.2 `#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()`.

9.12.3 Function Documentation

9.12.3.1 `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->ininfo = 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->ininfo = 0;
195             return -1;
196         }
197         if(gid >= 0)
198             fchown(kb->ininfo - 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));

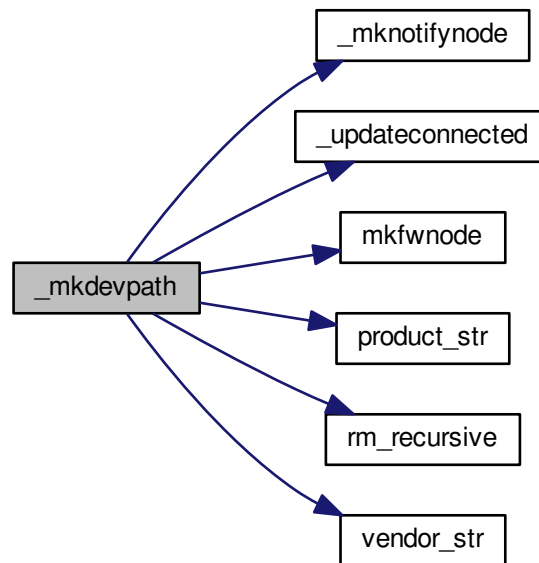
```

```

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:



9.12.3.2 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     }

```

Here is the caller graph for this function:



9.12.3.3 int_rmnotifynode (usbdevice * kb, int notify)

Definition at line 115 of file devnode.c.

References devpath, INDEX_OF, keyboard, usbdevice::outfifo, and UTFIFO_MAX.

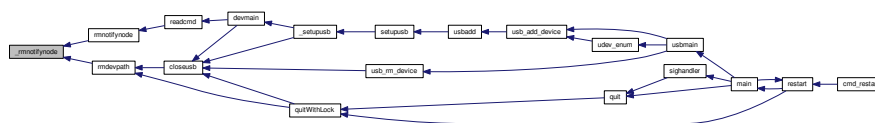
Referenced by rmdevpath(), and rmnotifynode().

```

115 {
116     if(notify < 0 || notify >= UTFIFO_MAX || !kb->outfifo[notify])
117         return -1;
118     int index = INDEX_OF(kb, keyboard);
119     char outpath[strlen(devpath) + 10];
120     snprintf(outpath, sizeof(outpath), "%s%d/notify%d", devpath, index, notify);
121     // Close FIFO
122     close(kb->outfifo[notify] - 1);
123     kb->outfifo[notify] = 0;
124     // Delete node
125     int res = remove(outpath);
126     return res;
127 }

```

Here is the caller graph for this function:



9.12.3.4 void_updateconnected ()

<devicepath> normally is /dev/input/ckb or /input/ckb.

Open the normal file under <devicepath>0/connected for writing. For each device connected, print its devcepath+number, the serial number of the usb device and the usb name of the device connected to that usb interface.

eg:

```
/dev/input/ckb1 0F022014ABABABABABABABABABA999 Corsair K95 RGB Gaming Keyboard
```

```
/dev/input/ckb2 0D02303DBACBACBACBACBACBACBAC998 Corsair M65 RGB Gaming Mouse
```

Set the file ownership to root. If the glob var gid is explicitly set to something different from -1 (the initial value), set file permission to 640, else to 644. This is used if you start the daemon with -gid=<GID> Parameter.

Because several independent threads may call [updateconnected\(\)](#), protect that procedure with locking/unlocking of **devmutex**.

Definition at line 55 of file devnode.c.

References ckb_warn, DEV_MAX, devmutex, devpath, gid, IS_CONNECTED, keyboard, and S_GID_READ.

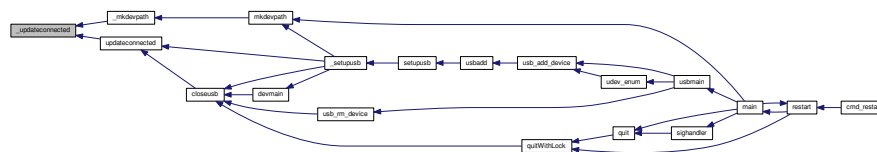
Referenced by _mkdevpath(), and updateconnected().


```

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,
70 keyboard[i].name);
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:



9.12.3.5 int mkdevpath (usbdevice * kb)

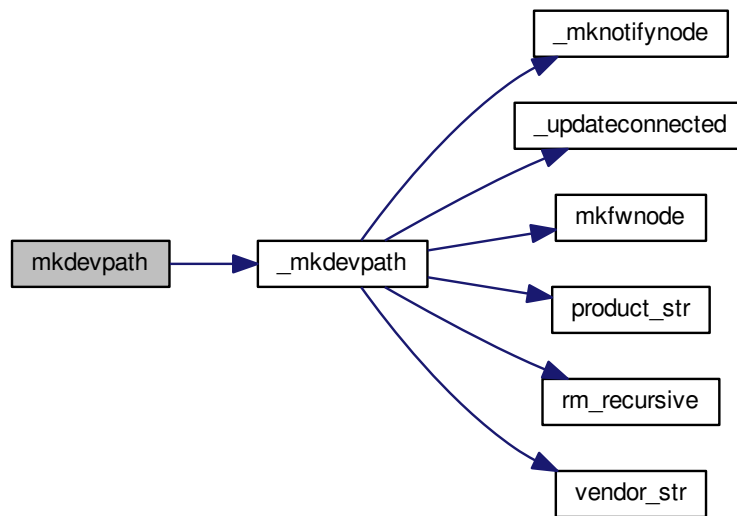
Definition at line 268 of file devnode.c.

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

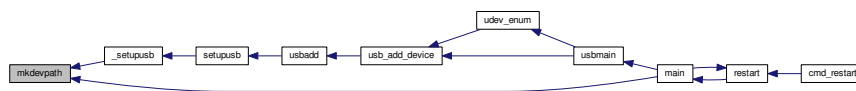
Referenced by `_setupusb()`, and `main()`.

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

Here is the call graph for this function:



Here is the caller graph for this function:



9.12.3.6 int mkfwnode (usbdevice * kb)

Definition at line 299 of file devnode.c.

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

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:0x%04x:0x%04x", kb->fwversion, kb->vendor, kb->
product);
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", fspath, strerror(errno));
328     remove(ppath);
329     return -2;
330 }
331 return 0;
332 }

```

Here is the caller graph for this function:



9.12.3.7 int mknotifynode (usbdevice * kb, int notify)

Definition at line 108 of file devnode.c.

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

Referenced by `readcmd()`.

```

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

```

Here is the call graph for this function:



Here is the caller graph for this function:



9.12.3.8 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:



9.12.3.9 void readlines_ctx_free (readlines_ctx ctx)

Definition at line 348 of file `devnode.c`.

References `_readlines_ctx::buffer`.

Referenced by `devmain()`.

348

{

Here is the caller graph for this function:



Definition at line 341 of file devnode.c.

```

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     }

```

```

graph LR
    readlines_ctx_init --> devmain
    devmain --> _setupusb
    _setupusb --> setupusb
    setupusb --> usbadd
    usbadd --> usb_add_device
    usb_add_device --> udev_enum
    udev_enum --> usbmain
    usbmain --> main
    main --> restart
    restart --> cmd_restart
    cmd_restart --> main

```

Definition at line 19 of file devnode.c.

```

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 }

```

[illegible]

9.12.3.12 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::ininfo`, `keyboard`, `OUTFIFO_MAX`, and `rm_recursive()`.

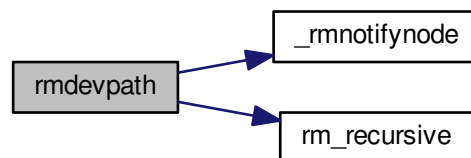
Referenced by `closeusb()`, and `quitWithLock()`.

```

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



9.12.3.13 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:



9.12.3.14 void updateconnected ()

Definition at line 81 of file devnode.c.

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

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

```

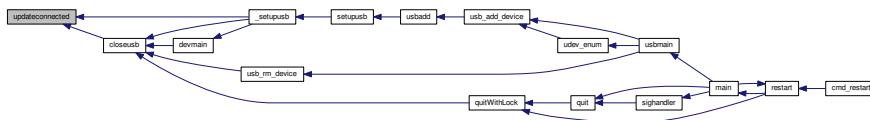
81     {
82     euid_guard_start;
83     _updateconnected();
84     euid_guard_stop;
85 }

```

Here is the call graph for this function:



Here is the caller graph for this function:



9.12.4 Variable Documentation

9.12.4.1 `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()`.

9.12.4.2 `long gid = -1`

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

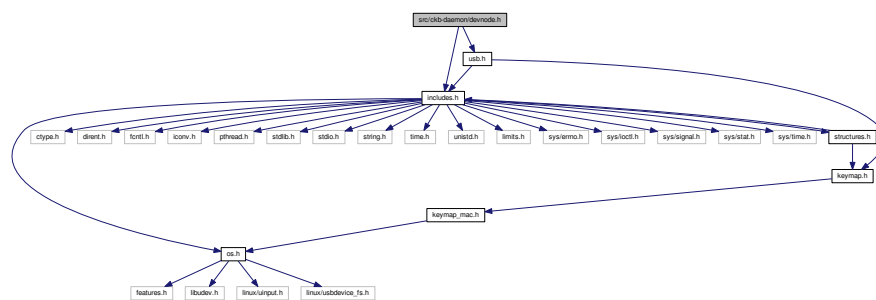
Referenced by `_mkdevpath()`, `_mknotifynode()`, `_updateconnected()`, `main()`, and `mkfwnode()`.

9.13 `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.

9.13.1 Macro Definition Documentation

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

Definition at line 17 of file devnode.h.

Referenced by `_mkdevpath()`.

9.13.1.2 #define S_CUSTOM_R (S_IRUSR | S_IWUSR | S_IRGRP)

Definition at line 18 of file devnode.h.

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

Definition at line 15 of file devnode.h.

Referenced by `_mkdevpath()`.

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

Definition at line 14 of file devnode.h.

Referenced by `_mkdevpath()`.

9.13.1.5 #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()`.

9.13.2 Typedef Documentation

9.13.2.1 typedef struct _readlines_ctx* readlines_ctx

Definition at line 39 of file devnode.h.

9.13.3 Function Documentation

9.13.3.1 int mkdevpath (usbdevice * kb)

Definition at line 268 of file devnode.c.

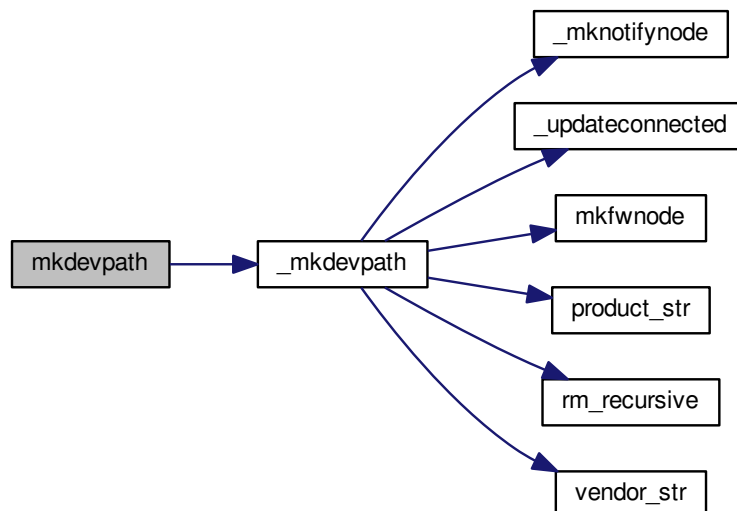
References `_mkdevpath()`, `euid_guard_start`, and `euid_guard_stop`.

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

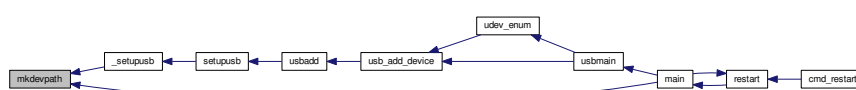
```

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:



9.13.3.2 int mkfwnode (usbdevice * kb)

Definition at line 299 of file devnode.c.

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

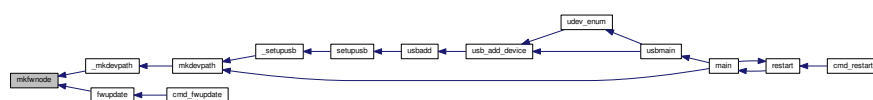
Referenced by `_mkdevpath()`, and `fwupdate()`.

```

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:0x%04x:0x%04x", kb->fwversion, kb->vendor, kb->
product);
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:



9.13.3.3 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             euid_guard_start;
110             int res = _mknotifyfn(kb, notify);
111             euid_guard_stop;
112             return res;
113         }
```

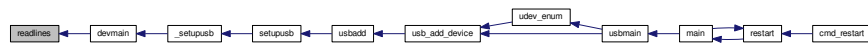


```

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:



9.13.3.5 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:



9.13.3.6 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:



9.13.3.7 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()`.

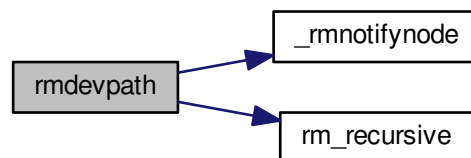
Referenced by `closeusb()`, and `quitWithLock()`.

```

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:



9.13.3.8 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:



9.13.3.9 void updateconnected ()

Definition at line 81 of file devnode.c.

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

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

```

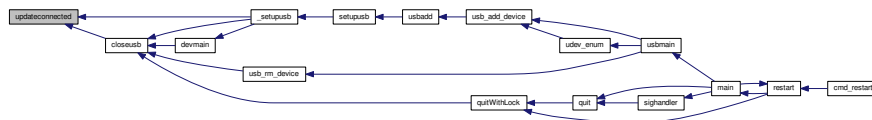
81     {
82     euid_guard_start;
83     _updateconnected();
84     euid_guard_stop;
85 }

```

Here is the call graph for this function:



Here is the caller graph for this function:




```

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 }

```

9.14.1.2 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 }

```

9.14.1.3 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 }

```

9.14.1.4 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 }

```

9.14.1.5 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`.

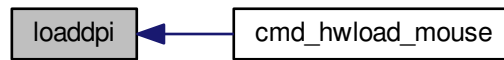
Referenced by `cmd_hwload_mouse()`.

```

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:



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

Definition at line 69 of file dpi.c.

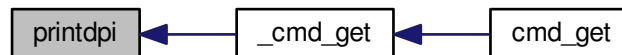
References `_readlines_ctx::buffer`, `DPI_COUNT`, `dpiset::enabled`, `dpiset::x`, and `dpiset::y`.

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:



9.14.1.7 `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`.

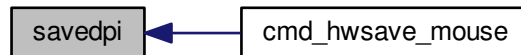
Referenced by `cmd_hwsave_mouse()`.

```

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:



9.14.1.8 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 }

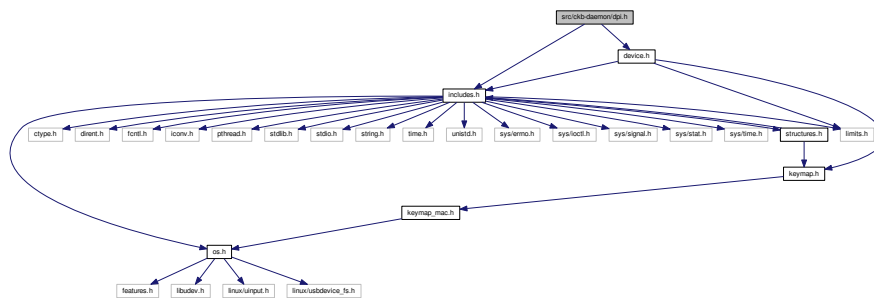
```

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

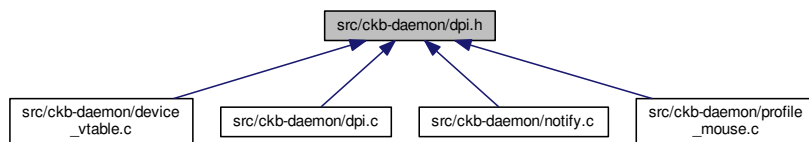
```
#include "includes.h"
```

```
#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_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)

9.15.1 Function Documentation

9.15.1.1 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 }
```

9.15.1.2 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 }
```

9.15.1.3 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 }
```

9.15.1.4 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 }
```

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

Definition at line 152 of file dpi.c.

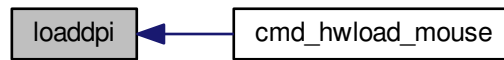
References `lighting::b`, `ckb_err`, `dpiset::current`, `DPI_COUNT`, `dpiset::enabled`, `lighting::g`, `LED_MOUSE`, `dpiset::lift`, `LIFT_MAX`, `LIFT_MIN`, `MSG_SIZE`, `N_MOUSE_ZONES`, `lighting::r`, `dpiset::snap`, `usbrecv`, `dpiset::x`, and `dpiset::y`.

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:



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

Definition at line 69 of file dpi.c.

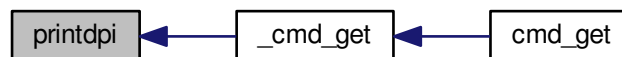
References `_readlines_ctx::buffer`, `DPI_COUNT`, `dpiset::enabled`, `dpiset::x`, and `dpiset::y`.

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:



9.15.1.7 `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`.

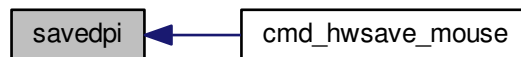
Referenced by `cmd_hwsave_mouse()`.


```

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:



9.15.1.8 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 },

```


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)

9.17.1 Macro Definition Documentation

9.17.1.1 #define FW_MAXSIZE (255 * 256)

Definition at line 51 of file firmware.c.

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

9.17.1.2 #define FW_NOFILE -1

Definition at line 7 of file firmware.c.

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

9.17.1.3 #define FW_OK 0

Definition at line 6 of file firmware.c.

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

9.17.1.4 #define FW_USBFAIL -3

Definition at line 9 of file firmware.c.

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

9.17.1.5 #define FW_WRONGDEV -2

Definition at line 8 of file firmware.c.

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

9.17.2 Function Documentation

9.17.2.1 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\(\)](#).

```

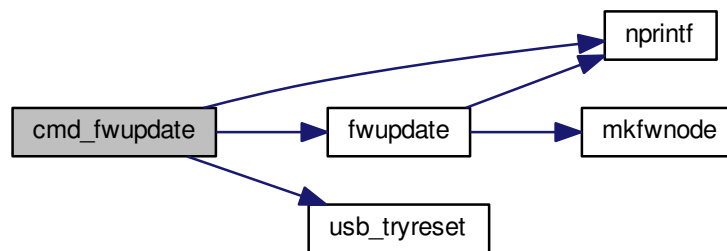
154
155     if(!HAS_FEATURES(kb, FEAT_FWUPDATE))
156         return 0;
157     // Update the firmware
158     int ret = fwupdate(kb, path, nnumber);
159     while(ret == FW_USBFAIL){
160         // Try to reset the device if it fails
161         if(usb_tryreset(kb))
162             break;
163         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:



9.17.2.2 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 <= 0 || 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)

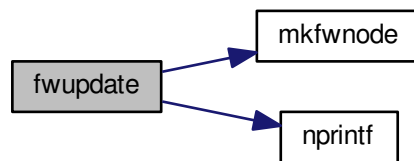
```

```

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:



9.17.2.3 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){

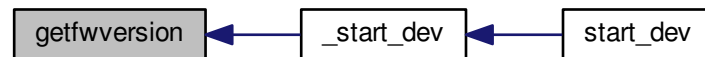
```

```

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 }

```

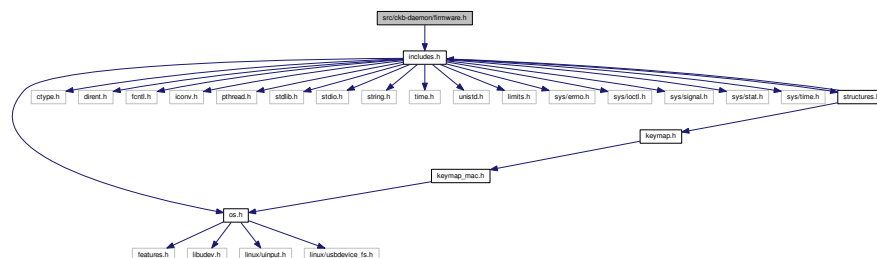
Here is the caller graph for this function:



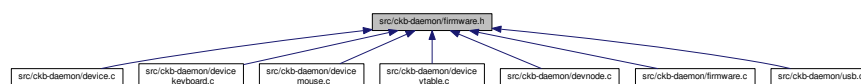
9.18 src/ckb-daemon/firmware.h File Reference

```
#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)`

9.18.1 Function Documentation

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

Definition at line 154 of file firmware.c.

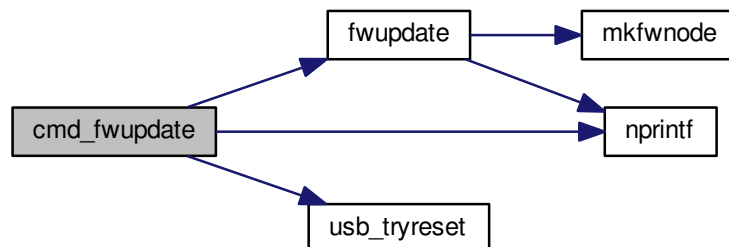
References `FEAT_FWUPDATE`, `FW_NOFILE`, `FW_OK`, `FW_USBFAIL`, `FW_WRONGDEV`, `fwupdate()`, `HAS_FEATURES`, `nprintf()`, and `usb_tryreset()`.

```

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:



9.18.1.2 `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;

```

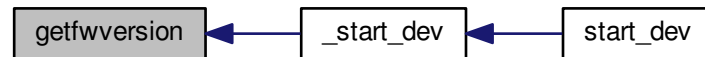


```

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:



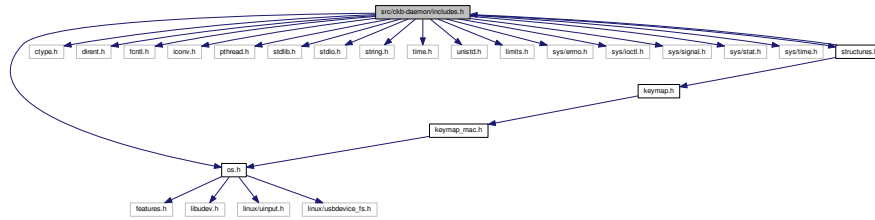
9.19 src/ckb-daemon/includes.h File Reference

```

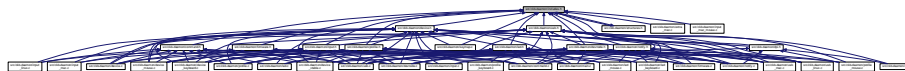
#include "os.h"
#include <ctype.h>
#include <dirent.h>
#include <fcntl.h>
#include <iconv.h>
#include <pthread.h>
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <time.h>
#include <unistd.h>
#include <limits.h>
#include <sys/errno.h>
#include <sys/ioctl.h>
#include <sys/signal.h>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

9.19.1 Macro Definition Documentation

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

Definition at line 41 of file includes.h.

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

Definition at line 50 of file includes.h.

Referenced by `_mkdevpath()`, `fwupdate()`, `getfwversion()`, `loaddpi()`, `loadrgb_kb()`, `loadrgb_mouse()`, `macro_pt_dequeue()`, `os_inputmain()`, `os_sendindicators()`, `os_setupusb()`, `restart()`, `setupusb()`, `uinputopen()`, `usb_tryreset()`, `usbadd()`, and `usbclaim()`.

9.19.1.3 `#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 49 of file includes.h.

Referenced by `_nk95cmd()`, `_usbrecv()`, `os_usbrecv()`, and `os_usbsend()`.

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

Definition at line 48 of file includes.h.

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

Definition at line 47 of file includes.h.

Referenced by `usbmain()`.

9.19.1.6 `#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 46 of file includes.h.

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

Definition at line 45 of file includes.h.

Referenced by `main()`.

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

Definition at line 56 of file includes.h.

Referenced by `_setupusb()`, `_start_dev()`, `closeusb()`, `cmd_restart()`, `fwupdate()`, `main()`, `os_inputmain()`, `os_setupusb()`, `quitWithLock()`, `rmdevpath()`, `usb_tryreset()`, `usbadd()`, and `usbclaim()`.

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

Definition at line 55 of file includes.h.

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

Definition at line 54 of file includes.h.

Referenced by main().

9.19.1.11 `#define ckb_s_err stdout`

Definition at line 37 of file includes.h.

9.19.1.12 `#define ckb_s_out stdout`

Definition at line 36 of file includes.h.

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

Definition at line 53 of file includes.h.

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

9.19.1.14 `#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 52 of file includes.h.

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

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

Definition at line 51 of file includes.h.

Referenced by main().

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

Definition at line 28 of file includes.h.

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

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

Definition at line 61 of file includes.h.

9.19.1.18 `#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 62 of file includes.h.

9.19.1.19 `#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 60 of file includes.h.

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

Definition at line 64 of file includes.h.

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

Definition at line 63 of file includes.h.

9.19.2 Typedef Documentation

9.19.2.1 `typedef unsigned char uchar`

Definition at line 25 of file includes.h.

9.19.2.2 `typedef unsigned short ushort`

Definition at line 26 of file includes.h.

9.19.3 Function Documentation

9.19.3.1 `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 }
```

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

```

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


9.20.1 Macro Definition Documentation

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

Referenced by `inputupdate_keys()`.

9.20.2 Function Documentation

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

Definition at line 344 of file `input.c`.

References `keymacro::actioncount`, `keymacro::actions`, `usbmode::bind`, `keymacro::combo`, `macroaction::delay`, `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()`.

```

344                                     {
345     binding* bind = &mode->bind;
346     if(!keys && !assignment){
347         // Null strings = "macro clear" -> erase the whole thing
348         for(int i = 0; i < bind->macrocount; i++)
349             free(bind->macros[i].actions);
350         bind->macrocount = 0;
351         return;
352     }
353     if(bind->macrocount >= MACRO_MAX)
354         return;
355     // Create a key macro
356     keymacro macro;
357     memset(&macro, 0, sizeof(macro));
358     // Scan the left side for key names, separated by +
359     int empty = 1;
360     int left = strlen(keys), right = strlen(assignment);
361     int position = 0, field = 0;
362     char keyname[24];
363     while(position < left && sscanf(keys + position, "%10[^\n]", keyname, &field) == 1){
364         int keycode;
365         if((sscanf(keyname, "%d", &keycode) && keycode >= 0 && keycode <
N_KEYS_INPUT)
366         || (sscanf(keyname, "%x", &keycode) && keycode >= 0 && keycode <
N_KEYS_INPUT)){
367             // Set a key numerically
368             SET_KEYBIT(macro.combo, keycode);
369             empty = 0;
370         } else {
371             // Find this key in the keymap
372             for(unsigned i = 0; i < N_KEYS_INPUT; i++){
373                 if(keymap[i].name && !strcmp(keyname, keymap[i].name)){
374                     macro.combo[i / 8] |= 1 << (i % 8);
375                     empty = 0;
376                     break;
377                 }
378             }
379         }
380         if(keys[position += field] == '+')
381             position++;
382     }
383     if(empty)
384         return;
385     // Count the number of actions (comma separated)
386     int count = 1;
387     for(const char* c = assignment; *c != 0; c++){
388         if(*c == ',')
389             count++;
390     }
391     // Allocate a buffer for them
392     macro.actions = calloc(count, sizeof(macroaction));
393     macro.actioncount = 0;
394     // Scan the actions
395     position = 0;
396     field = 0;
397     // max action = old 11 chars plus 12 chars which is the max 32-bit int 4294967295 size
398     while(position < right && sscanf(assignment + position, "%23[^\n]", keyname, &field) == 1){
399         if(!strcmp(keyname, "clear"))
400             break;

```

```

401
402 // Check for local key delay of the form '[+<->key]<=<delay>'
403 long int long_delay; // scanned delay value, used to keep delay in range.
404 unsigned int delay = UINT_MAX; // computed delay value. UINT_MAX means use global delay value.
405 char real_keyname[12]; // temp to hold the left side (key) of the <key>=<delay>
406 int scan_matches = sscanf(keyname, "%11[^]=%ld", real_keyname, &long_delay);
407 if (scan_matches == 2) {
408     if (0 <= long_delay && long_delay < UINT_MAX) {
409         delay = (unsigned int)long_delay;
410         strcpy(keyname, real_keyname); // keyname[24], real_keyname[12]
411     }
412 }
413
414 int down = (keyname[0] == '+');
415 if(down || keyname[0] == '-') {
416     int keycode;
417     if(sscanf(keyname + 1, "%d", &keycode) && keycode >= 0 && keycode < N_KEYS_INPUT)
418         || (sscanf(keyname + 1, "%x%x", &keycode) && keycode >= 0 && keycode <
N_KEYS_INPUT)){
419         // Set a key numerically
420         macro.actions[macro.actioncount].scan =
keymap[keycode].scan;
421         macro.actions[macro.actioncount].down = down;
422         macro.actions[macro.actioncount].delay = delay;
423         macro.actioncount++;
424     } else {
425         // Find this key in the keymap
426         for(unsigned i = 0; i < N_KEYS_INPUT; i++){
427             if(keymap[i].name && !strcmp(keyname + 1, keymap[i].name)){
428                 macro.actions[macro.actioncount].scan =
keymap[i].scan;
429                 macro.actions[macro.actioncount].down = down;
430                 macro.actions[macro.actioncount].delay = delay;
431                 macro.actioncount++;
432                 break;
433             }
434         }
435     }
436 }
437 if(assignment[position += field] == ',')
438     position++;
439 }
440
441 // See if there's already a macro with this trigger
442 keymacro* macros = bind->macros;
443 for(int i = 0; i < bind->macrocount; i++){
444     if(!memcmp(macros[i].combo, macro.combo, N_KEYBYTES_INPUT)){
445         free(macros[i].actions);
446         // If the new macro has no actions, erase the existing one
447         if(!macro.actioncount){
448             for(int j = i + 1; j < bind->macrocount; j++)
449                 memcpy(macros + j - 1, macros + j, sizeof(keymacro));
450             bind->macrocount--;
451         } else
452             // If there are actions, replace the existing with the new
453             memcpy(macros + i, &macro, sizeof(keymacro));
454         return;
455     }
456 }
457
458 // Add the macro to the device settings if not empty
459 if(macro.actioncount < 1)
460     return;
461 memcpy(bind->macros + (bind->macrocount++), &macro, sizeof(
keymacro));
462 if(bind->macrocount >= bind->macrocap)
463     bind->macros = realloc(bind->macros, (bind->macrocap += 16) * sizeof(
keymacro));
464 }

```

Here is the caller graph for this function:



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

Definition at line 306 of file input.c.

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

```

306                                     {
307     if(keyindex >= N_KEYS_INPUT)
308         return;
309     // Find the key to bind to
310     int tocode = 0;
311     if(sscanf(to, "%x%ux", &tocode) != 1 && sscanf(to, "%#u", &tocode) == 1 && tocode <
N_KEYS_INPUT){
312         pthread_mutex_lock(&imutex(kb));
313         mode->bind.base[keyindex] = tocode;
314         pthread_mutex_unlock(&imutex(kb));
315         return;
316     }
317     // If not numeric, look it up
318     for(int i = 0; i < N_KEYS_INPUT; i++){
319         if(keymap[i].name && !strcmp(to, keymap[i].name)){
320             pthread_mutex_lock(&imutex(kb));
321             mode->bind.base[keyindex] = keymap[i].scan;
322             pthread_mutex_unlock(&imutex(kb));
323             return;
324         }
325     }
326 }
```

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

Definition at line 466 of file input.c.

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

```

466     {
467         pthread_mutex_lock(&imutex(kb));
468         _cmd_macro(mode, keys, assignment);
469         pthread_mutex_unlock(&imutex(kb));
470     }
```

Here is the call graph for this function:



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

Definition at line 336 of file input.c.

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

```

336                                     {
337     if(keyindex >= N_KEYS_INPUT)
338         return;
339     pthread_mutex_lock(&imutex(kb));
340     mode->bind.base[keyindex] = keymap[keyindex].scan;
341     pthread_mutex_unlock(&imutex(kb));
342 }
```

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

Definition at line 328 of file input.c.

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

```

328                                     {
329     if(keyindex >= N_KEYS_INPUT)
330         return;
331     pthread_mutex_lock(imutex(kb));
332     mode->bind.base[keyindex] = KEY_UNBOUND;
333     pthread_mutex_unlock(imutex(kb));
334 }
```

9.20.2.6 void freebind (binding * bind)

Definition at line 299 of file input.c.

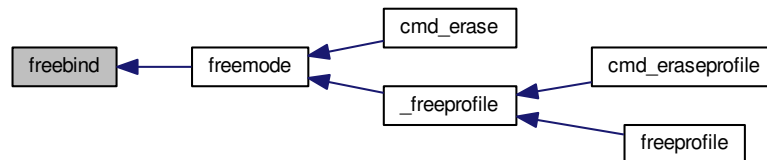
References keymacro::actions, binding::macrocount, and binding::macros.

Referenced by freemode().

```

299                                     {
300     for(int i = 0; i < bind->macrocount; i++)
301         free(bind->macros[i].actions);
302     free(bind->macros);
303     memset(bind, 0, sizeof(*bind));
304 }
```

Here is the caller graph for this function:



9.20.2.7 void initbind (binding * bind)

Definition at line 291 of file input.c.

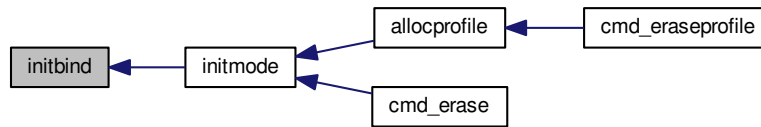
References binding::base, keymap, binding::macrocap, binding::macrocount, binding::macros, N_KEYS_INPUT, and key::scan.

Referenced by initmode().

```

291                                     {
292     for(int i = 0; i < N_KEYS_INPUT; i++)
293         bind->base[i] = keymap[i].scan;
294     bind->macros = calloc(32, sizeof(keymacro));
295     bind->macrocap = 32;
296     bind->macrocount = 0;
297 }
```

Here is the caller graph for this function:



9.20.2.8 void inputupdate (usbdevice * kb)

Definition at line 240 of file input.c.

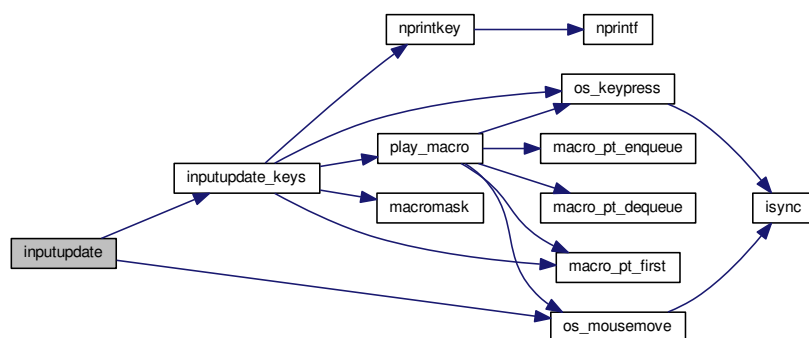
References `usbdevice::input`, `inputupdate_keys()`, `os_mousemove()`, `usbdevice::profile`, `usbinput::rel_x`, `usbinput::rel_y`, `usbdevice::uinput_kb`, and `usbdevice::uinput_mouse`.

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

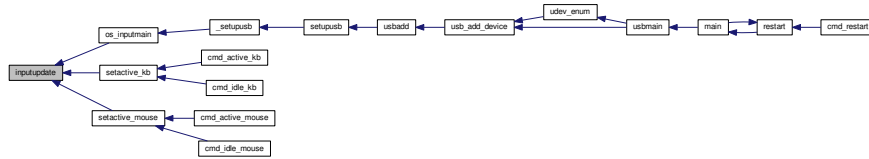
```

240                                     {
241 #ifdef OS_LINUX
242     if ((!kb->uinput_kb || !kb->uinput_mouse)
243 #else
244     if (!kb->event
245 #endif
246         || !kb->profile)
247         return;
248     // Process key/button input
249     inputupdate_keys(kb);
250     // Process mouse movement
251     usbinput* input = &kb->input;
252     if(input->rel_x != 0 || input->rel_y != 0){
253         os_mousemove(kb, input->rel_x, input->rel_y);
254         input->rel_x = input->rel_y = 0;
255     }
256     // Finish up
257     memcpy(input->prevkeys, input->keys, N_KEYBYTES_INPUT);
258 }
  
```

Here is the call graph for this function:



Here is the caller graph for this function:



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

Parameters

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

Process all queued keypresses if no macro is running yet.

Todo If we want to get all keys typed while a macro is played, add the code for it here.

Definition at line 133 of file input.c.

References `usbdevice::active`, `binding::base`, `usbmode::bind`, `keymacro::combo`, `usbprofile::currentmode`, `usbdevice::input`, `IS_MOD`, `IS_WHEEL`, `parameter::kb`, `keymap`, `usbinput::keys`, `parameter::macro`, `macro_pt_first()`, `binding::macrocount`, `macromask()`, `binding::macros`, `N_KEYBYTES_INPUT`, `N_KEYS_INPUT`, `usbmode::notify`, `nprintkey()`, `os_keypress()`, `OUTFIFO_MAX`, `play_macro()`, `usbinput::prevkeys`, `usbdevice::profile`, `key::scan`, `SCAN_SILENT`, and `keymacro::triggered`.

Referenced by `inputupdate()`.

```

133                                     {
134     usbmode* mode = kb->profile->currentmode;
135     binding* bind = &mode->bind;
136     usbinput* input = &kb->input;
137
138     // Don't do anything if the state hasn't changed
139     if(!memcmp(input->prevkeys, input->keys, N_KEYBYTES_INPUT))
140         return;
141     // Look for macros matching the current state
142     if (kb->active) {
143         for (int i = 0; i < bind->macrocount; i++) {
144             keymacro* macro = &bind->macros[i];
145             if (macromask(input->keys, macro->combo)) {
146                 if (!macro->triggered) {
147                     parameter_t* params = malloc(sizeof(parameter_t));
148                     if (params == 0) {
149                         perror("inputupdate_keys got no more mem.");
150                     } else {
151                         pthread_t thread = 0;
152                         params->kb = kb;
153                         params->macro = macro;
154                         int retval = pthread_create(&thread, 0, play_macro, (void*)params);
155                         if (retval) {
156                             perror("inputupdate_keys: Creating thread returned not null");
157                         } else {
158                             macro->triggered = 1;
159                         }
160                     }
161                 }
162             } else macro->triggered = 0;
163         }
164     }
165     // Make a list of keycodes to send. Rearrange them so that modifier keydowns always come first
166     // and modifier keyups always come last. This ensures that shortcut keys will register properly
167     // even if both keydown events happen at once.
168     // N_KEYS + 4 is used because the volume wheel generates keydowns and keyups at the same time
169     // (it's currently impossible to press all four at once, but safety first)
170     int events[N_KEYS_INPUT + 4];
171     int modcount = 0, keycount = 0, rmodcount = 0;
172     for(int byte = 0; byte < N_KEYBYTES_INPUT; byte++){
173         char oldb = input->prevkeys[byte], newb = input->keys[byte];
174         if(oldb == newb)

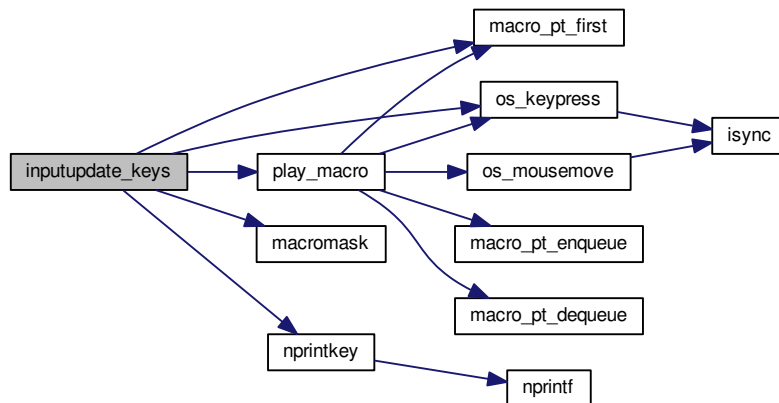
```

```

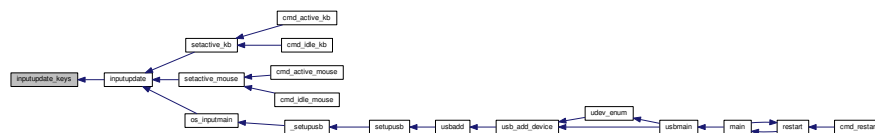
175         continue;
176     for(int bit = 0; bit < 8; bit++){
177         int keyindex = byte * 8 + bit;
178         if(keyindex >= N_KEYS_INPUT)
179             break;
180         const key* map = keymap + keyindex;
181         int scancode = (kb->active) ? bind->base[keyindex] : map->
scan;
182         char mask = 1 << bit;
183         char old = oldb & mask, new = newb & mask;
184         // If the key state changed, send it to the input device
185         if(old != new){
186             // Don't echo a key press if there's no scancode associated
187             if(!(scancode & SCAN_SILENT)){
188                 if(IS_MOD(scancode)){
189                     if(new){
190                         // Modifier down: Add to the end of modifier keys
191                         for(int i = keycount + rmodcount; i > 0; i--){
192                             events[modcount + i] = events[modcount + i - 1];
193                             // Add 1 to the scancode because A is zero on OSX
194                             // Positive code = keydown, negative code = keyup
195                             events[modcount++] = scancode + 1;
196                         } else {
197                             // Modifier up: Add to the end of everything
198                             events[modcount + keycount + rmodcount++] = -(scancode + 1);
199                         }
200                     } else {
201                         // Regular keypress: add to the end of regular keys
202                         for(int i = rmodcount; i > 0; i--){
203                             events[modcount + keycount + i] = events[modcount + keycount + i - 1];
204                             events[modcount + keycount++] = new ? (scancode + 1) : -(scancode + 1);
205                             // The volume wheel and the mouse wheel don't generate keyups, so create them
206                             automatically
207                             #define IS_WHEEL(scan, kb) (((scan) == KEY_VOLUMEUP || (scan) == KEY_VOLUMEDOWN || (scan) == BTN_WHEELUP
|| (scan) == BTN_WHEELDOWN) && !IS_K65(kb))
208                             if(new && IS_WHEEL(map->scan, kb)){
209                                 for(int i = rmodcount; i > 0; i--){
210                                     events[modcount + keycount + i] = events[modcount + keycount + i - 1];
211                                     events[modcount + keycount++] = -(scancode + 1);
212                                     input->keys[byte] &= ~mask;
213                                 }
214                             }
215                             // Print notifications if desired
216                             if(kb->active){
217                                 for(int notify = 0; notify < OUTFIFO_MAX; notify++){
218                                     if(mode->notify[notify][byte] & mask){
219                                         nprintkey(kb, notify, keyindex, new);
220                                         // Wheels doesn't generate keyups
221                                         if(new && IS_WHEEL(map->scan, kb))
222                                             nprintkey(kb, notify, keyindex, 0);
223                                     }
224                                 }
225                             }
226                         }
227                     }
228                 }
229             }
230         }
231         if (!macro_pt_first()) {
232             int totalkeys = modcount + keycount + rmodcount;
233             for(int i = 0; i < totalkeys; i++){
234                 int scancode = events[i];
235                 os_keypress(kb, (scancode < 0 ? -scancode : scancode) - 1, scancode > 0);
236             }
237         }
238     }

```

Here is the call graph for this function:



Here is the caller graph for this function:



9.20.2.10 static pthread_t macro_pt_dequeue () [static]

Returns

the pthread_id of the first element. If list is empty, return 0.

Attention

Because multiple threads may use this function in parallel, save the critical section with a mutex.

< why are we called?

< Was last element in the list, so clear tail.

< save the return value before deleting element

Definition at line 48 of file input.c.

References ckb_err, ptlist::next, pt_head, and ptlist::thread_id.

Referenced by play_macro().

```

48     {
49     pthread_t retval = 0;
50     ptlist_t* elem = 0;
51     if (pt_head == 0 && pt_tail == 0) {
52         ckb_err("macro_pt_dequeue: called on empty list.\n");
53         return 0;
54     }
55     elem = pt_head;
56     pt_head = pt_head->next;
  
```

```
57     if (pt_head == 0) pt_tail = 0;
58     retval = elem->thread_id;
59     free(elem);
60     return retval;
61 }
```

Here is the caller graph for this function:



9.20.2.11 static void macro_pt_enqueue () [static]

Attention

Because multiple threads may use this function in parallel, save the critical section with a mutex.

```
< exit on critical situation;
```

Todo find a better exit strategy if no more mem available.

< The element knows its ID by itself

< new list, first element

< existing list, append on last element (FIFO)

Definition at line 26 of file input.c.

References `ptlist::next`, and `ptlist::thread_id`.

Referenced by `play_macro()`.

```

26         {
27     ptlist_t* new_elem = malloc(sizeof(ptlist_t));
28     if (!new_elem) {
29         perror("macro_pt_enqueue: ");
30         exit (-1);
31     }
32     new_elem->next = 0;
33     new_elem->thread_id = pthread_self();
34     if (pt_head == 0) {
35         pt_head = pt_tail = new_elem;
36     } else {
37         pt_tail->next = new_elem;
38         pt_tail = new_elem;
39     }
40 }

```

Here is the caller graph for this function:



9.20.2.12 `static pthread_t macro_pt_first () [static]`

Parameters

<i>param</i>	<i>parameter_t</i> to store Kb-ptr and macro-ptr (thread may get only one user-parameter)
--------------	---

Returns

0 on success, -1 else (no one is interested in it except the kernel...)

First have a look if we are the first and only macro-thread to run. If not, wait. So enqueue our thread first, so it is remembered for us and can be seen by all others.

< If the first thread in the list is not our, another one is running

< Give all new threads the chance to enter the block.

Send events for each keypress in the macro

< Synchronization between macro output and color information

< use this unlock / relock for enabling the parallel running colorization

< local delay set

< use default global delay

< use delays depending on macro length

< protect the linked list and the mvar

< Wake up all waiting threads

< for the linked list and the mvar

< Sync keyboard input/output and colorization

Definition at line 78 of file input.c.

References keymacro::actioncount, keymacro::actions, macroaction::delay, usbdevice::delay, macroaction::down, parameter::kb, parameter::macro, macro_pt_dequeue(), macro_pt_enqueue(), macro_pt_first(), mmutex, mmutex2, mvar, os_keypress(), os_mousemove(), macroaction::rel_x, macroaction::rel_y, and macroaction::scan.

Referenced by inputupdate_keys().

```

78
79     parameter_t* ptr = (parameter_t*) param;
80     usbdevice* kb = ptr->kb;
81     keymacro* macro = ptr->macro;
82
83     pthread_mutex_lock(mmutex2(kb));
84     macro_pt_enqueue();
85     // ckb_info("Entering critical section with 0x%x. Queue head is 0x%x\n", (unsigned long
86     int)pthread_self(), (unsigned long int)macro_pt_first());
87     while (macro_pt_first() != pthread_self()) {
88         // ckb_info("Now waiting with 0x%x because of 0x%x\n", (unsigned long int)pthread_self(),
89         (unsigned long int)macro_pt_first());
90         pthread_cond_wait(mvar(kb), mmutex2(kb));
91         // ckb_info("Waking up with 0x%x\n", (unsigned long int)pthread_self());
92     }
93     pthread_mutex_unlock(mmutex2(kb));
94
95     pthread_mutex_lock(mmutex(kb));
96     for (int a = 0; a < macro->actioncount; a++) {
97         macroaction* action = macro->actions + a;
98         if (action->rel_x != 0 || action->rel_y != 0)
99             os_mousemove(kb, action->rel_x, action->rel_y);
100         else {
101             os_keypress(kb, action->scan, action->down);
102             pthread_mutex_unlock(mmutex(kb));
103             if (action->delay != UINT_MAX) {
104                 usleep(action->delay);
105             } else if (kb->delay != UINT_MAX) {
106                 usleep(kb->delay);
107             } else if (a < (macro->actioncount - 1)) {
108                 if (a > 200) {
109                     usleep(100);
110                 } else if (a > 20) {
111                     usleep(30);
112                 }
113             }

```

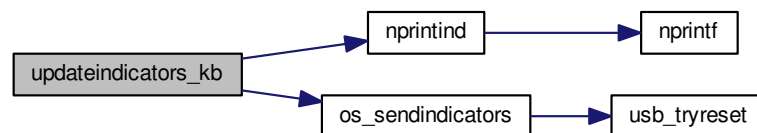


```

266     usbmode* mode = kb->profile->currentmode;
267     new = (new & ~mode->ioff) | mode->ion;
268 }
269 kb->ileds = new;
270 kb->hw_ileds_old = hw_new;
271 if(old != new || force){
272     DELAY_SHORT(kb);
273     os_sendindicators(kb);
274 }
275 // Print notifications if desired
276 if(!kb->active)
277     return;
278 usbmode* mode = kb->profile->currentmode;
279 uchar indicators[] = { I_NUM, I_CAPS, I_SCROLL };
280 for(unsigned i = 0; i < sizeof(indicators) / sizeof(uchar); i++){
281     uchar mask = indicators[i];
282     if((hw_old & mask) == (hw_new & mask))
283         continue;
284     for(int notify = 0; notify < OUTFIFO_MAX; notify++){
285         if(mode->inotify[notify] & mask)
286             nprintind(kb, notify, mask, hw_new & mask);
287     }
288 }
289 }

```

Here is the call graph for this function:



9.20.3 Variable Documentation

9.20.3.1 ptlist_t* pt_head = 0 [static]

Definition at line 17 of file input.c.

Referenced by macro_pt_dequeue().

9.20.3.2 ptlist_t* pt_tail = 0 [static]

Definition at line 19 of file input.c.

9.21 src/ckb-daemon/input.h File Reference

```

#include "includes.h"
#include "usb.h"

```

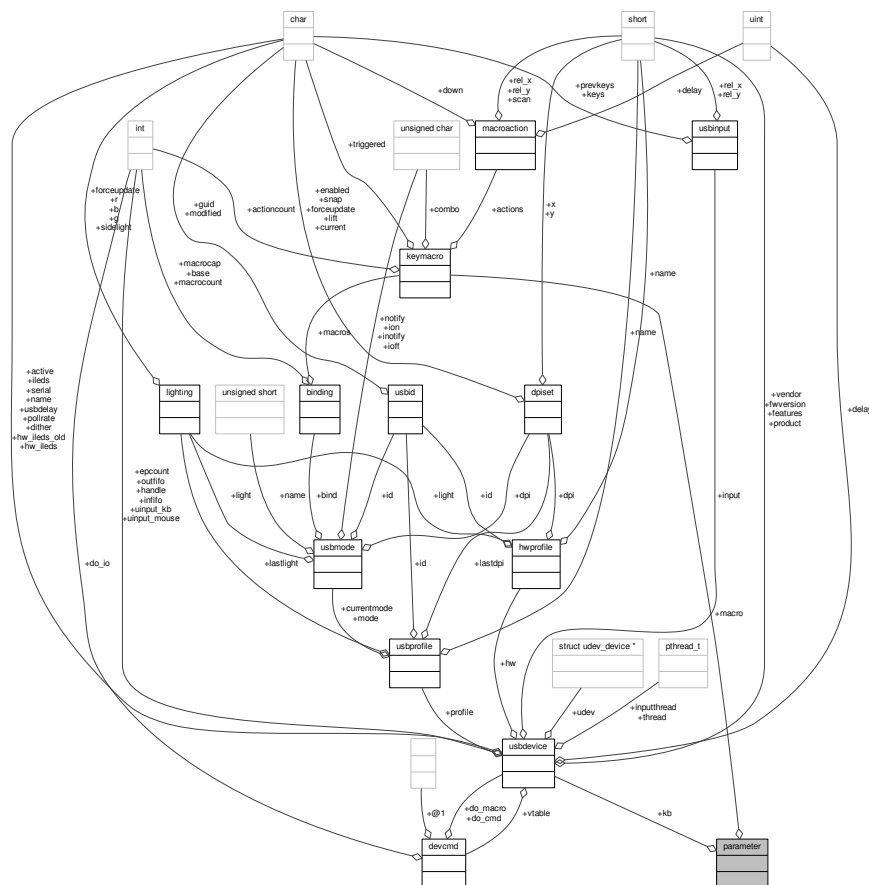

- void [freebind](#) (binding *bind)
- void [cmd_bind](#) (usbdevice *kb, usbmode *mode, int dummy, int keyindex, const char *to)
- void [cmd_unbind](#) (usbdevice *kb, usbmode *mode, int dummy, int keyindex, const char *ignored)
- void [cmd_rebind](#) (usbdevice *kb, usbmode *mode, int dummy, int keyindex, 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)

9.21.1 Data Structure Documentation

9.21.1.1 struct parameter

Definition at line 54 of file input.h.

Collaboration diagram for parameter:



Data Fields

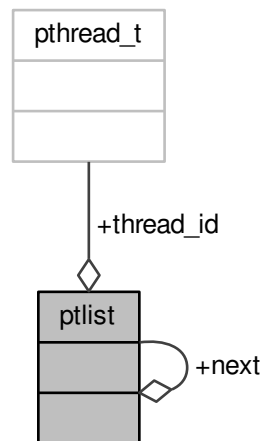
usbdevice *	kb
-----------------------------	----

keymacro *	macro	
----------------------------	-------	--

9.21.1.2 struct ptlist

Definition at line 62 of file input.h.

Collaboration diagram for ptlist:



Data Fields

struct ptlist *	next	
pthread_t	thread_id	

9.21.2 Macro Definition Documentation

9.21.2.1 `#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()`.

9.21.3 Typedef Documentation

9.21.3.1 `typedef struct parameter parameter_t`

9.21.3.2 `typedef struct ptlist ptlist_t`

9.21.4 Function Documentation

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

Definition at line 306 of file input.c.

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

```

306                                     {
307     if(keyindex >= N_KEYS_INPUT)
308         return;
309     // Find the key to bind to
310     int tocode = 0;
311     if(sscanf(to, "%x%ux", &tocode) != 1 && sscanf(to, "%#u", &tocode) == 1 && tocode <
N_KEYS_INPUT){
312         pthread_mutex_lock(&imutex(kb));
313         mode->bind.base[keyindex] = tocode;
314         pthread_mutex_unlock(&imutex(kb));
315         return;
316     }
317     // If not numeric, look it up
318     for(int i = 0; i < N_KEYS_INPUT; i++){
319         if(keymap[i].name && !strcmp(to, keymap[i].name)){
320             pthread_mutex_lock(&imutex(kb));
321             mode->bind.base[keyindex] = keymap[i].scan;
322             pthread_mutex_unlock(&imutex(kb));
323             return;
324         }
325     }
326 }
```

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

Definition at line 466 of file input.c.

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

```

466     {
467         pthread_mutex_lock(&imutex(kb));
468         _cmd_macro(mode, keys, assignment);
469         pthread_mutex_unlock(&imutex(kb));
470     }
```

Here is the call graph for this function:



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

Definition at line 336 of file input.c.

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

```

336                                     {
337     if(keyindex >= N_KEYS_INPUT)
338         return;
339     pthread_mutex_lock(&imutex(kb));
340     mode->bind.base[keyindex] = keymap[keyindex].scan;
341     pthread_mutex_unlock(&imutex(kb));
342 }
```

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

Definition at line 328 of file input.c.

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

```

328                                     {
329     if(keyindex >= N_KEYS_INPUT)
330         return;
331     pthread_mutex_lock(imutex(kb));
332     mode->bind.base[keyindex] = KEY_UNBOUND;
333     pthread_mutex_unlock(imutex(kb));
334 }
```

9.21.4.5 void freebind (binding * bind)

Definition at line 299 of file input.c.

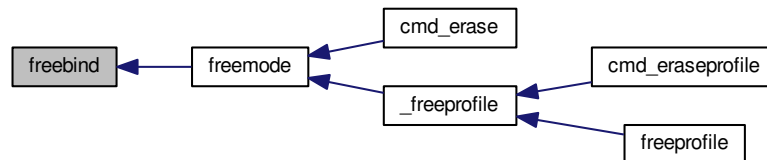
References keymacro::actions, binding::macrocount, and binding::macros.

Referenced by freemode().

```

299                                     {
300     for(int i = 0; i < bind->macrocount; i++)
301         free(bind->macros[i].actions);
302     free(bind->macros);
303     memset(bind, 0, sizeof(*bind));
304 }
```

Here is the caller graph for this function:



9.21.4.6 void initbind (binding * bind)

Definition at line 291 of file input.c.

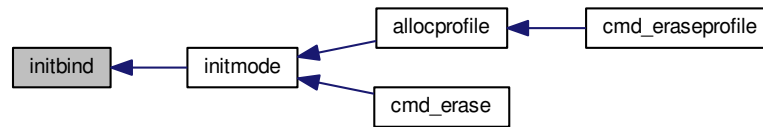
References binding::base, keymap, binding::macrocap, binding::macrocount, binding::macros, N_KEYS_INPUT, and key::scan.

Referenced by initmode().

```

291                                     {
292     for(int i = 0; i < N_KEYS_INPUT; i++)
293         bind->base[i] = keymap[i].scan;
294     bind->macros = calloc(32, sizeof(keymacro));
295     bind->macrocap = 32;
296     bind->macrocount = 0;
297 }
```


Here is the caller graph for this function:



9.21.4.7 void inputupdate (usbdevice * kb)

Definition at line 240 of file input.c.

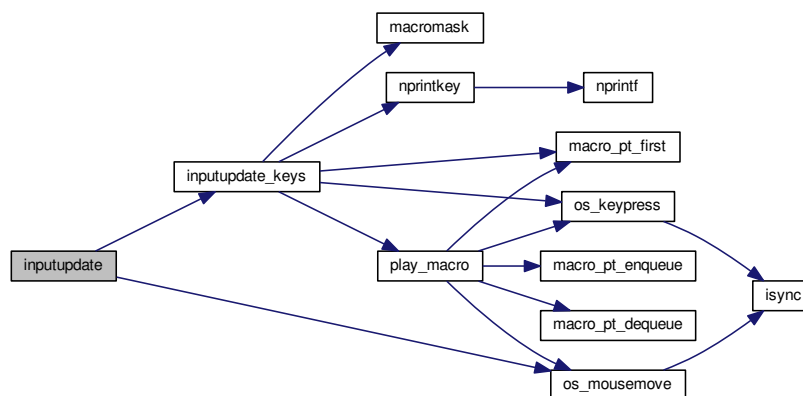
References `usbdevice::input`, `inputupdate_keys()`, `os_mousemove()`, `usbdevice::profile`, `usbinput::rel_x`, `usbinput::rel_y`, `usbdevice::uinput_kb`, and `usbdevice::uinput_mouse`.

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

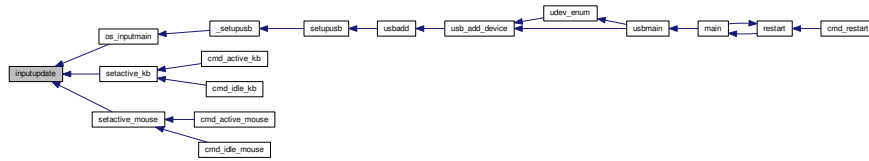
```

240                                     {
241 #ifdef OS_LINUX
242     if ((!kb->uinput_kb || !kb->uinput_mouse)
243 #else
244     if (!kb->event
245 #endif
246         || !kb->profile)
247         return;
248     // Process key/button input
249     inputupdate_keys(kb);
250     // Process mouse movement
251     usbinput* input = &kb->input;
252     if(input->rel_x != 0 || input->rel_y != 0){
253         os_mousemove(kb, input->rel_x, input->rel_y);
254         input->rel_x = input->rel_y = 0;
255     }
256     // Finish up
257     memcpy(input->prevkeys, input->keys, N_KEYBYTES_INPUT);
258 }
  
```

Here is the call graph for this function:



Here is the caller graph for this function:



9.21.4.8 void os_inputclose (usbdevice * kb)

Definition at line 76 of file input_linux.c.

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

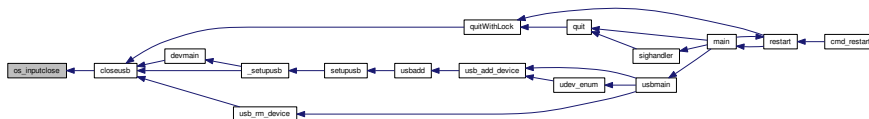
Referenced by closeusb().

```

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:



9.21.4.9 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:



9.21.4.10 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`.

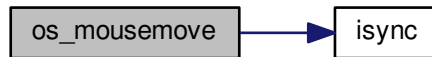
Referenced by `inputupdate_keys()`, and `play_macro()`.


```

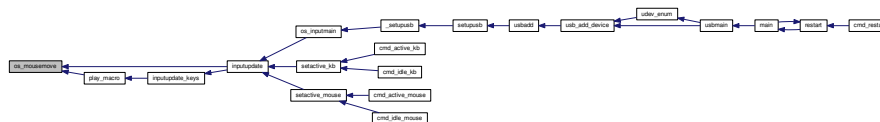
156         event.code = REL_Y;
157         event.value = y;
158         if(write(kb->uinput_mouse - 1, &event, sizeof(event)) <= 0)
159             ckb_warn("uinput write failed: %s\n", strerror(errno));
160         else
161             isync(kb);
162     }
163 }

```

Here is the call graph for this function:



Here is the caller graph for this function:



9.21.4.12 int os_setupindicators (usbdevice * kb)

Definition at line 189 of file input_linux.c.

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

Referenced by `_setupusb()`.

```

189                                     {
190     // Initialize LEDs to all off
191     kb->hw_ileds = kb->hw_ileds_old = kb->ileds = 0;
192     // Create and detach thread to read LED events
193     pthread_t thread;
194     int err = pthread_create(&thread, 0, _ledthread, kb);
195     if(err != 0)
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:



9.21.4.13 void updateindicators_kb (usbdevice * kb, int force)

Definition at line 260 of file input.c.

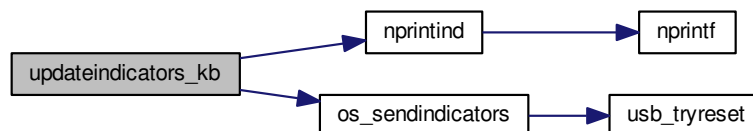
References `usbdevice::active`, `usbprofile::currentmode`, `DELAY_SHORT`, `usbdevice::hw_ileds`, `usbdevice::hw_ileds_old`, `I_CAPS`, `I_NUM`, `I_SCROLL`, `usbdevice::ileds`, `usbmode::inotify`, `usbmode::ioff`, `usbmode::ion`, `nprintind()`, `os_sendindicators()`, `OUTFIFO_MAX`, and `usbdevice::profile`.

```

260                                     {
261     // Read current hardware indicator state (set externally)
262     uchar old = kb->ileds, hw_old = kb->hw_ileds_old;
263     uchar new = kb->hw_ileds, hw_new = new;
264     // Update them if needed
265     if(kb->active){
266         usbmode* mode = kb->profile->currentmode;
267         new = (new & ~mode->ioff) | mode->ion;
268     }
269     kb->ileds = new;
270     kb->hw_ileds_old = hw_new;
271     if(old != new || force){
272         DELAY_SHORT(kb);
273         os_sendindicators(kb);
274     }
275     // Print notifications if desired
276     if(!kb->active)
277         return;
278     usbmode* mode = kb->profile->currentmode;
279     uchar indicators[] = { I_NUM, I_CAPS, I_SCROLL };
280     for(unsigned i = 0; i < sizeof(indicators) / sizeof(uchar); i++){
281         uchar mask = indicators[i];
282         if((hw_old & mask) == (hw_new & mask))
283             continue;
284         for(int notify = 0; notify < OUTFIFO_MAX; notify++){
285             if(mode->inotify[notify] & mask)
286                 nprintind(kb, notify, mask, hw_new & mask);
287         }
288     }
289 }

```

Here is the call graph for this function:



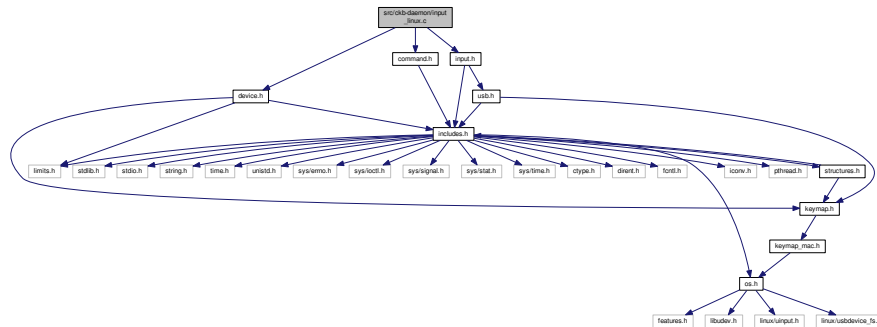
9.22 src/ckb-daemon/input_linux.c File Reference

```

#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)

9.22.1 Function Documentation

9.22.1.1 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 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:



9.22.1.2 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:



9.22.1.3 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

```

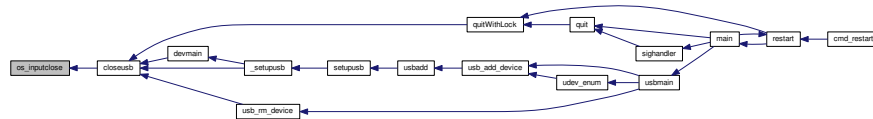


```

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:



9.22.1.4 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:



9.22.1.5 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`.

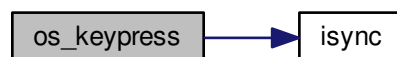
Referenced by `inputupdate_keys()`, and `play_macro()`.

```

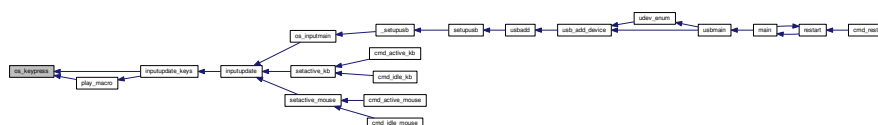
118                                     {
119     struct input_event event;
120     memset(&event, 0, sizeof(event));
121     int is_mouse = 0;
122     if(scancode == BTN_WHEELUP || scancode == BTN_WHEELDOWN) {
123         // The mouse wheel is a relative axis
124         if(!down)
125             return;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



9.22.1.8 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 }

```

Here is the caller graph for this function:



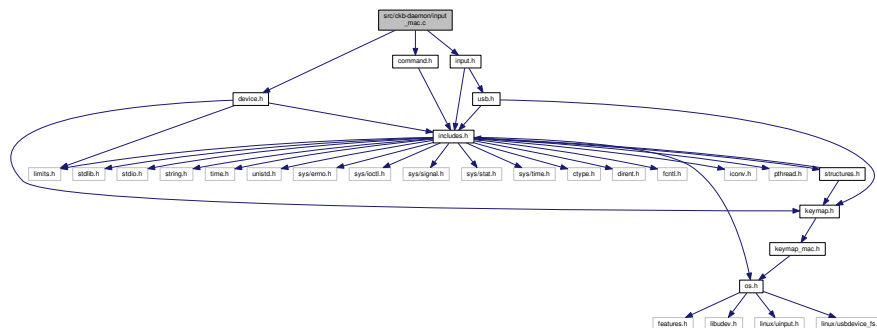
9.23 src/ckb-daemon/input_mac.c File Reference

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

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

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

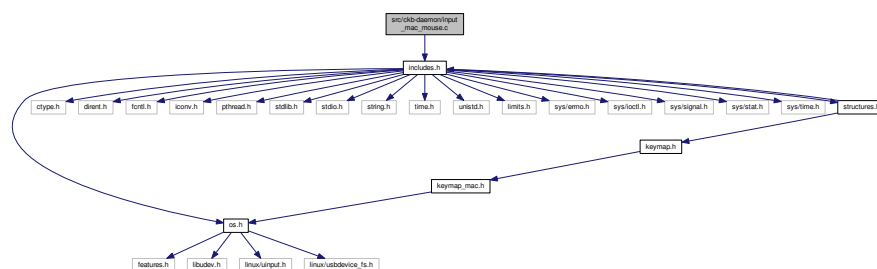
Include dependency graph for input_mac.c:



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

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

Include dependency graph for input_mac_mouse.c:



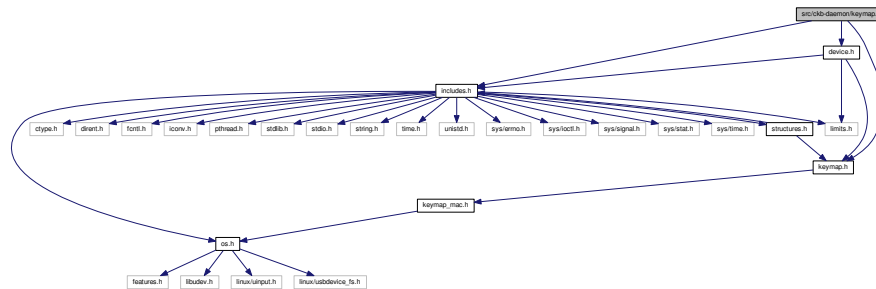
9.25 src/ckb-daemon/keymap.c File Reference

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

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

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

Include dependency graph for keymap.c:



Macros

- `#define BUTTON_HID_COUNT 5`

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)]

9.25.1 Macro Definition Documentation

9.25.1.1 #define BUTTON_HID_COUNT 5

Definition at line 364 of file keymap.c.

Referenced by `corsair_mousecopy()`, and `hid_mouse_translate()`.

9.25.2 Function Documentation

9.25.2.1 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:



9.25.2.2 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:



9.25.2.3 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, -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);    // 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:



9.25.2.4 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:



9.25.3 Variable Documentation

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

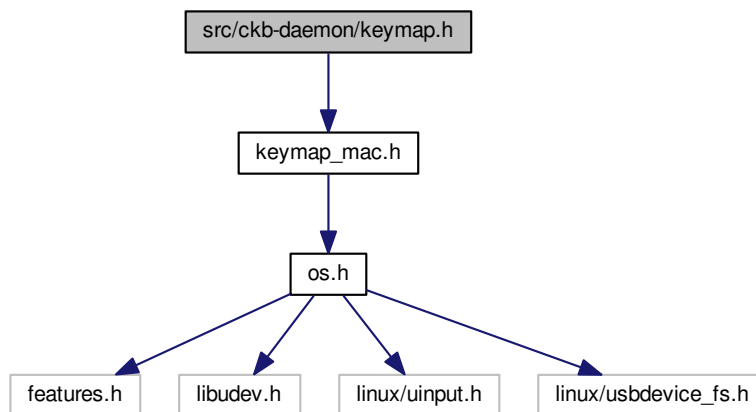
Definition at line 5 of file keymap.c.

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

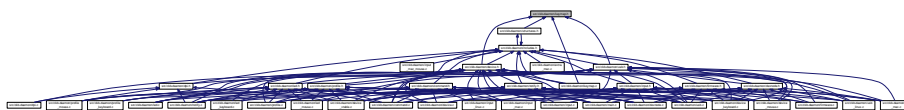
9.26 src/ckb-daemon/keymap.h File Reference

```
#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

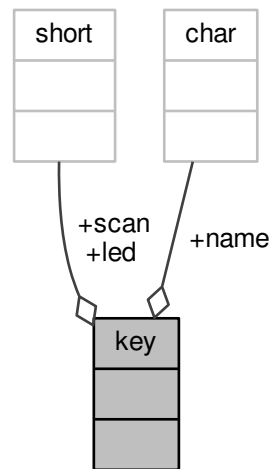
- const [key](#) [keymap](#) [(((152+3+12)+25)+11)]

9.26.1 Data Structure Documentation

9.26.1.1 struct key

Definition at line 49 of file keymap.h.

Collaboration diagram for key:



Data Fields

short	led	
const char *	name	
short	scan	

9.26.2 Macro Definition Documentation

9.26.2.1 #define BTN_WHEELDOWN 0x1f02

Definition at line 13 of file keymap.h.

Referenced by os_keypress().

9.26.2.2 #define BTN_WHEELUP 0x1f01

Definition at line 12 of file keymap.h.

Referenced by os_keypress().

9.26.2.3 #define KEY_BACKSLASH_ISO KEY_BACKSLASH

Definition at line 20 of file keymap.h.

9.26.2.4 #define KEY_CORSAIR -2

Definition at line 8 of file keymap.h.

9.26.2.5 #define KEY_NONE -1

Definition at line 7 of file keymap.h.

9.26.2.6 #define KEY_UNBOUND -3

Definition at line 9 of file keymap.h.

Referenced by cmd_unbind().

9.26.2.7 #define LED_DPI (LED_MOUSE + 2)

Definition at line 43 of file keymap.h.

Referenced by loadrgb_mouse(), and savergb_mouse().

9.26.2.8 #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().

9.26.2.9 #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().

9.26.2.10 #define MOUSE_EXTRA_FIRST (MOUSE_BUTTON_FIRST + N_BUTTONS_HW)

Definition at line 34 of file keymap.h.

Referenced by hid_mouse_translate().

9.26.2.11 #define N_BUTTONS_EXTENDED 25

Definition at line 32 of file keymap.h.

9.26.2.12 #define N_BUTTONS_HW 20

Definition at line 31 of file keymap.h.

Referenced by corsair_mousecopy().

9.26.2.13 #define N_KEY_ZONES 3

Definition at line 27 of file keymap.h.

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

Definition at line 46 of file keymap.h.

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

Definition at line 25 of file keymap.h.

Referenced by corsair_kbcopy().

9.26.2.16 `#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()`.

9.26.2.17 `#define N_KEYS_EXTENDED (N_KEYS_INPUT + N_MOUSE_ZONES_EXTENDED)`

Definition at line 45 of file keymap.h.

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

9.26.2.18 `#define N_KEYS_EXTRA 12`

Definition at line 29 of file keymap.h.

9.26.2.19 `#define N_KEYS_HW 152`

Definition at line 24 of file keymap.h.

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

9.26.2.20 `#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()`.

9.26.2.21 `#define N_MOUSE_ZONES 5`

Definition at line 40 of file keymap.h.

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

9.26.2.22 `#define N_MOUSE_ZONES_EXTENDED 11`

Definition at line 41 of file keymap.h.

9.26.2.23 `#define SCAN_KBD 0`

Definition at line 57 of file keymap.h.

9.26.2.24 `#define SCAN_MOUSE 0x1000`

Definition at line 58 of file keymap.h.

Referenced by `os_keypress()`.

9.26.2.25 `#define SCAN_SILENT 0x8000`

Definition at line 56 of file keymap.h.

Referenced by `inputupdate_keys()`.

9.26.3 Function Documentation

9.26.3.1 `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:

9.26.3.2 `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:



9.26.3.3 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:



9.26.3.4 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
388                 CLEAR_KEYBIT(kbinput, MOUSE_EXTRA_FIRST);
389             if(wheel < 0)
390                 SET_KEYBIT(kbinput, MOUSE_EXTRA_FIRST + 1); // wheeldn
391             else
392                 CLEAR_KEYBIT(kbinput, MOUSE_EXTRA_FIRST + 1);
393         }

```

Here is the caller graph for this function:



9.26.4 Variable Documentation

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

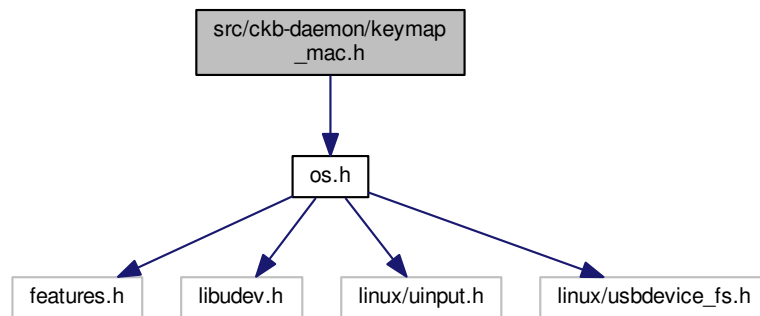
Definition at line 5 of file `keymap.c`.

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

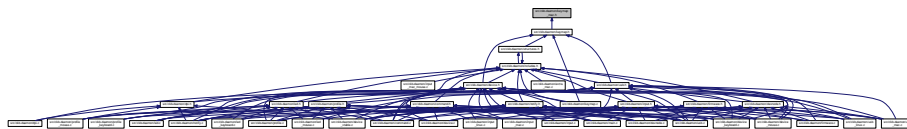
9.27 `src/ckb-daemon/keymap_mac.h` File Reference

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

Include dependency graph for keymap_mac.h:



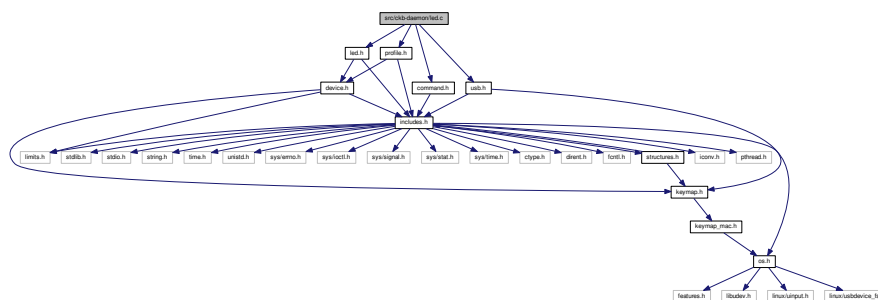
This graph shows which files directly or indirectly include this file:



9.28 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_loff` (`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`)

9.28.1 Function Documentation

9.28.1.1 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:



9.28.1.2 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:



9.28.1.3 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:



9.28.1.4 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:



9.28.1.5 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 }

```

9.28.1.6 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:



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

Definition at line 25 of file led.c.

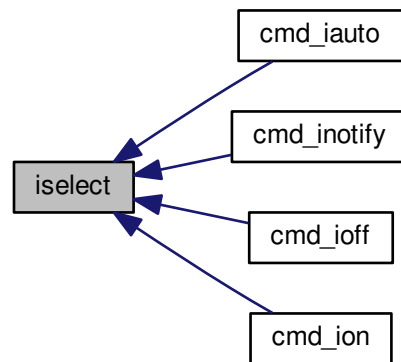
References I_CAPS, I_NUM, and I_SCROLL.

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:



9.28.1.8 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 }
```

```

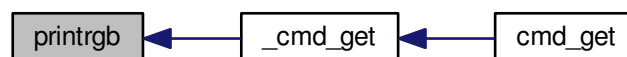
123         strncpy(names[i], name, 11);
124     }
125     // Check to make sure these aren't all the same color
126     int same = 1;
127     for(int i = 1; i < N_KEYS_EXTENDED; i++){
128         if(!names[i][0])
129             continue;
130         if(r[i] != r[0] || g[i] != g[0] || b[i] != b[0]){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

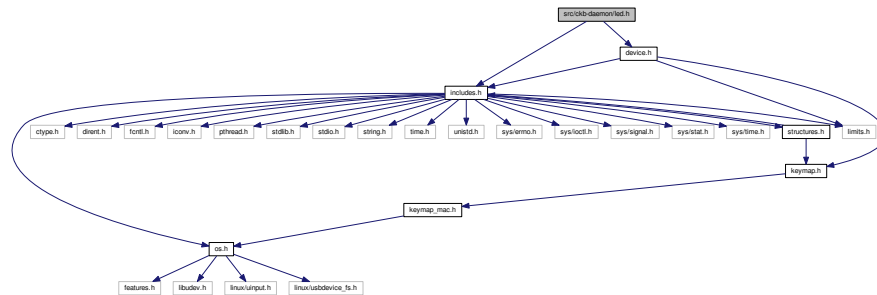


Here is the caller graph for this function:



9.29 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)

9.29.1 Function Documentation

9.29.1.1 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:



9.29.1.2 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:



9.29.1.3 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:



9.29.1.4 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:



9.29.1.5 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 }
```

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

Since Firmware Version 2.05 for K95RGB 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. Up to now we know, that K70RGB Pro and K70 Lux RGB have firmware version 2.04 and having the problem also. So we have to determine in the most inner loop the firmware version and type of KB to select the correct compare-table.

Read colors

< That is the old comparison method: you get back what you sent.

Normally a firmware version ≥ 2.05 runs with the new compare array. Up to now there is a 2.04 running in K70 RGB Lux with the same behavior. It seems that K70RGB has the same problem

Definition at line 181 of file led_keyboard.c.

References lighting::b, ckb_err, usbdevice::fwversion, lighting::g, MSG_SIZE, N_KEYS_HW, P_K70_LUX, P_K70_LUX_NRGB, usbdevice::product, 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
205         uchar cmp_pkt[4][4] = {
206             { 0x0e, 0x14, 0x03, 0x01 },
207             { 0x0e, 0xff, 0x01, 60 },
208             { 0x0e, 0xff, 0x02, 60 },
209             { 0x0e, 0xff, 0x03, 24 },
210         };
211
212         uchar* colors[3] = { light->r, light->g, light->b };
213         for(int clr = 0; clr < 3; clr++){
214             for(int i = 0; i < 4; i++){
215                 if(!usbrecv(kb, data_pkt[i + clr * 4], in_pkt[i]))
216                     return -1;
217
218                 uchar* comparePacket = data_pkt[i + clr * 4];
219                 if ((kb->fwversion >= 0x205)
220                     || ((kb->fwversion >= 0x204)
221                         && ((kb->product == P_K70_LUX_NRGB) || (kb->
222 product == P_K70_LUX)))) {
223                     comparePacket = cmp_pkt[i];
224                 }
225
226                 if (memcmp(in_pkt[i], comparePacket, 4)) {
227                     ckb_err("Bad input header\n");
228                     ckb_err("color = %d, i = %d, mode = %d\nOutput (Request): %2.2x %2.2x %2.2x
229 %2.2x\nInput (Reply): %2.2x %2.2x %2.2x %2.2x %2.2x %2.2x %2.2x %2.2x\n", clr, i, mode,
230 comparePacket[0], comparePacket[1], comparePacket[2], comparePacket[3],
231 in_pkt[i][0], in_pkt[i][1], in_pkt[i][2], in_pkt[i][3], in_pkt[i][4], in_pkt[i][5],
232 in_pkt[i][6], in_pkt[i][7]);
233                     in_pkt[2][0] = 0x99;
234                     in_pkt[2][1] = 0x99;
235                     in_pkt[2][2] = 0x99;
236                     in_pkt[2][3] = 0x99;
237                     usbrecv(kb, in_pkt[2], in_pkt[2]); // just to find it in the wireshark log
238                     return -1;
239                 }
240             }
241         }

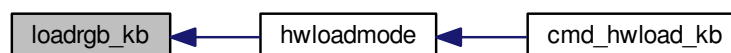
```

```

245     }
246     // Copy colors to lighting. in_pkt[0] is irrelevant.
247     memcpy(colors[clr], in_pkt[1] + 4, 60);
248     memcpy(colors[clr] + 60, in_pkt[2] + 4, 60);
249     memcpy(colors[clr] + 120, in_pkt[3] + 4, 24);
250 }
251 } else {
252     uchar data_pkt[5][MSG_SIZE] = {
253         { 0x0e, 0x14, 0x02, 0x01, 0x01, mode + 1, 0 },
254         { 0xff, 0x01, 60, 0 },
255         { 0xff, 0x02, 60, 0 },
256         { 0xff, 0x03, 60, 0 },
257         { 0xff, 0x04, 36, 0 },
258     };
259     uchar in_pkt[4][MSG_SIZE] = {
260         { 0xff, 0x01, 60, 0 },
261         { 0xff, 0x02, 60, 0 },
262         { 0xff, 0x03, 60, 0 },
263         { 0xff, 0x04, 36, 0 },
264     };
265     // Write initial packet
266     if(!usbSEND(kb, data_pkt[0], 1))
267         return -1;
268     // Read colors
269     for(int i = 1; i < 5; i++){
270         if(!usbRECV(kb, data_pkt[i], in_pkt[i - 1]))
271             return -1;
272         if(memcmp(in_pkt[i - 1], data_pkt[i], 4)){
273             ckb_err("Bad input header\n");
274             return -1;
275         }
276     }
277     // Copy the data back to the mode
278     uint8_t mr[N_KEYS_HW / 2], mg[N_KEYS_HW / 2], mb[
N_KEYS_HW / 2];
279     memcpy(mr, in_pkt[0] + 4, 60);
280     memcpy(mr + 60, in_pkt[1] + 4, 12);
281     memcpy(mg, in_pkt[1] + 16, 48);
282     memcpy(mg + 48, in_pkt[2] + 4, 24);
283     memcpy(mb, in_pkt[2] + 28, 36);
284     memcpy(mb + 36, in_pkt[3] + 4, 36);
285     // Unpack LED data to 8bpc format
286     for(int i = 0; i < N_KEYS_HW; i++){
287         int i_2 = i / 2;
288         uint8_t r, g, b;
289
290         // 3-bit intensities stored in alternate nybbles.
291         if (i & 1) {
292             r = 7 - (mr[i_2] >> 4);
293             g = 7 - (mg[i_2] >> 4);
294             b = 7 - (mb[i_2] >> 4);
295         } else {
296             r = 7 - (mr[i_2] & 0x0F);
297             g = 7 - (mg[i_2] & 0x0F);
298             b = 7 - (mb[i_2] & 0x0F);
299         }
300         // Scale 3-bit values up to 8 bits.
301         light->r[i] = r << 5 | r << 2 | r >> 1;
302         light->g[i] = g << 5 | g << 2 | g >> 1;
303         light->b[i] = b << 5 | b << 2 | b >> 1;
304     }
305 }
306 return 0;
307 }

```

Here is the caller graph for this function:



9.29.1.7 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:



9.29.1.8 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++){

```

```

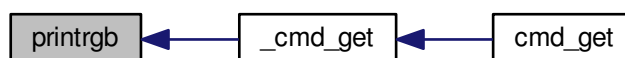
119     const char* name = keymap[i].name;
120     if(keymap[i].led < 0 || !has_key(name, kb))
121         names[i][0] = 0;
122     else
123         strncpy(names[i], name, 11);
124 }
125 // Check to make sure these aren't all the same color
126 int same = 1;
127 for(int i = 1; i < N_KEYS_EXTENDED; i++){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

```

Here is the call graph for this function:



Here is the caller graph for this function:



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

Definition at line 139 of file led_keyboard.c.

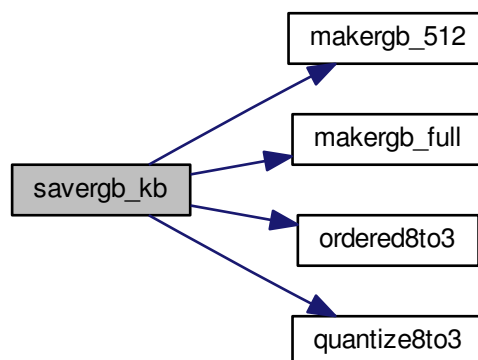
References `usbdevice::dither`, `usbdevice::fwversion`, `IS_STRAFE`, `makergb_512()`, `makergb_full()`, `MSG_SIZE`, `ordered8to3()`, `quantize8to3()`, and `usbsend`.

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 :
175         quantize8to3);
176         if(!usbsend(kb, data_pkt[0], 5))
177             return -1;
178     }
179     return 0;
180 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



9.29.1.10 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:



9.29.1.11 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`.

```

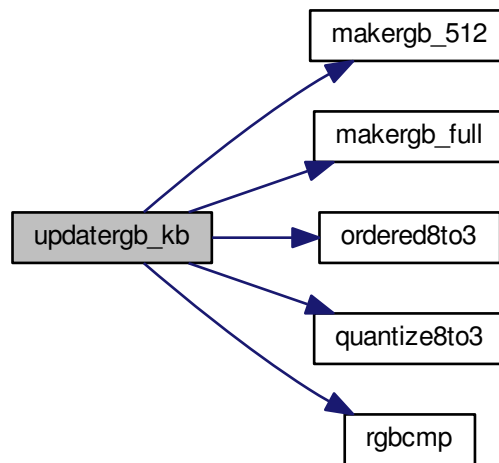
77
78     {
  
```

```

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:



9.29.1.12 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
27         && !rgbcmp(lastlight, newlight))
28         return 0;
29     lastlight->forceupdate = newlight->forceupdate = 0;
30
31     // Send the RGB values for each zone to the mouse
32     uchar data_pkt[2][MSG_SIZE] = {
33         { 0x07, 0x22, N_MOUSE_ZONES, 0x01, 0 }, // RGB colors
34         { 0x07, 0x05, 0x02, 0 } // Lighting on/off
35     };
36     uchar* rgb_data = &data_pkt[0][4];
37     for(int i = 0; i < N_MOUSE_ZONES; i++){
38         *rgb_data++ = i + 1;
39         *rgb_data++ = newlight->r[LED_MOUSE + i];
40         *rgb_data++ = newlight->g[LED_MOUSE + i];
41         *rgb_data++ = newlight->b[LED_MOUSE + i];
42     }
43     // Send RGB data
44     if(!usb send(kb, data_pkt[0], 1))
45         return -1;
46     int was_black = isblack(kb, lastlight), is_black = isblack(kb, newlight);
47     if(is_black){
48         // If the lighting is black, send the deactivation packet (M65 only)
49         if(!usb send(kb, data_pkt[1], 1))
50             return -1;
51     } else if(was_black || force){
52         // If the lighting WAS black, or if we're on forced update, send the activation packet
53         data_pkt[1][4] = 1;
54         if(!usb send(kb, data_pkt[1], 1))

```


- `#define O7(i) O6(i) O6((i) + 64)`
- `#define O8(i) O7(i) O7((i) + 127)`

Functions

- static `uchar ordered8to3` (int index, `uchar` value)
- static `uchar quantize8to3` (int index, `uchar` value)
- static void `makergb_512` (const `lighting` *light, `uchar` data_pkt[5][64], `uchar`(*ditherfn)(int, `uchar`))
- static void `makergb_full` (const `lighting` *light, `uchar` data_pkt[12][64])
- static int `rgbcmp` (const `lighting` *lhs, const `lighting` *rhs)
- int `updatergb_kb` (`usbdevice` *kb, int force)
- int `savergb_kb` (`usbdevice` *kb, `lighting` *light, int mode)
- int `loadrgb_kb` (`usbdevice` *kb, `lighting` *light, int mode)

Variables

- [illegible]

Generated on Mon Jun 5 2017 23:09:31 for ckb-next by Doxygen

Generated on Mon Jun 5 2017 23:09:31 for ckb-next by Doxygen

Generated on Mon Jun 5 2017 23:09:31 for ckb-next by Doxygen

Generated on Mon Jun 5 2017 23:09:31 for ckb-next by Doxygen

Generated on Mon Jun 5 2017 23:09:31 for ckb-next by Doxygen

Generated on Mon Jun 5 2017 23:09:31 for ckb-next by Doxygen

Generated on Mon Jun 5 2017 23:09:31 for ckb-next by Doxygen

Generated on Mon Jun 5 2017 23:09:31 for ckb-next by Doxygen

Generated on Mon Jun 5 2017 23:09:31 for ckb-next by Doxygen

Generated on Mon Jun 5 2017 23:09:31 for ckb-next by Doxygen

9.30.1 Macro Definition Documentation

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

Definition at line 9 of file led_keyboard.c.

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

Definition at line 10 of file led_keyboard.c.

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

Definition at line 11 of file led_keyboard.c.

9.30.1.4 **#define O0(i) BR4(i),**

Definition at line 12 of file led_keyboard.c.

9.30.1.5 **#define O1(i) O0(i) O0((i) + 1)**

Definition at line 13 of file led_keyboard.c.

9.30.1.6 **#define O2(i) O1(i) O1((i) + 2)**

Definition at line 14 of file led_keyboard.c.

9.30.1.7 **#define O3(i) O2(i) O2((i) + 4)**

Definition at line 15 of file led_keyboard.c.

9.30.1.8 **#define O4(i) O3(i) O3((i) + 8)**

Definition at line 16 of file led_keyboard.c.

9.30.1.9 **#define O5(i) O4(i) O4((i) + 16)**

Definition at line 17 of file led_keyboard.c.

9.30.1.10 **#define O6(i) O5(i) O5((i) + 32)**

Definition at line 18 of file led_keyboard.c.

9.30.1.11 **#define O7(i) O6(i) O6((i) + 64)**

Definition at line 19 of file led_keyboard.c.

9.30.1.12 **#define O8(i) O7(i) O7((i) + 127)**

Definition at line 20 of file led_keyboard.c.

9.30.2 Function Documentation

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

Since Firmware Version 2.05 for K95RGB 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. Up to now we know, that K70RGB Pro and K70 Lux RGB have firmware version 2.04 and having the problem also. So we have to determine in the most inner loop the firmware version and type of KB to select the correct compare-table.

Read colors

< That is the old comparison method: you get back what you sent.

Normally a firmware version ≥ 2.05 runs with the new compare array. Up to now there is a 2.04 running in K70 RGB Lux with the same behavior. It seems that K70RGB has the same problem

Definition at line 181 of file led_keyboard.c.

References lighting::b, ckb_err, usbdevice::fwversion, lighting::g, MSG_SIZE, N_KEYS_HW, P_K70_LUX, P_K70_LUX_NRGB, usbdevice::product, lighting::r, usbrecv, and usb send.

Referenced by hloadmode().

```

181                                     {
182     if(kb->fwversion >= 0x0120){
183         uchar data_pkt[12][MSG_SIZE] = {
184             { 0x0e, 0x14, 0x03, 0x01, 0x01, mode + 1, 0x01 },
185             { 0xff, 0x01, 60, 0 },
186             { 0xff, 0x02, 60, 0 },
187             { 0xff, 0x03, 24, 0 },
188             { 0x0e, 0x14, 0x03, 0x01, 0x01, mode + 1, 0x02 },
189             { 0xff, 0x01, 60, 0 },
190             { 0xff, 0x02, 60, 0 },
191             { 0xff, 0x03, 24, 0 },
192             { 0x0e, 0x14, 0x03, 0x01, 0x01, mode + 1, 0x03 },
193             { 0xff, 0x01, 60, 0 },
194             { 0xff, 0x02, 60, 0 },
195             { 0xff, 0x03, 24, 0 },
196         };
197         uchar in_pkt[4][MSG_SIZE] = {
198             { 0x0e, 0x14, 0x03, 0x01 },
199             { 0xff, 0x01, 60, 0 },
200             { 0xff, 0x02, 60, 0 },
201             { 0xff, 0x03, 24, 0 },
202         };
203
204
205         uchar cmp_pkt[4][4] = {
206             { 0x0e, 0x14, 0x03, 0x01 },
207             { 0x0e, 0xff, 0x01, 60 },
208             { 0x0e, 0xff, 0x02, 60 },
209             { 0x0e, 0xff, 0x03, 24 },
210         };
211
212         uchar* colors[3] = { light->r, light->g, light->b };
213         for(int clr = 0; clr < 3; clr++){
214             for(int i = 0; i < 4; i++){
215                 if(!usbrecv(kb, data_pkt[i + clr * 4], in_pkt[i]))
216                     return -1;
217
218                 uchar* comparePacket = data_pkt[i + clr * 4];
219                 if ((kb->fwversion >= 0x205)
220                     || ((kb->fwversion >= 0x204)
221                         && ((kb->product == P_K70_LUX_NRGB) || (kb->
222 product == P_K70_LUX)))) {
223                     comparePacket = cmp_pkt[i];
224                 }
225
226                 if (memcmp(in_pkt[i], comparePacket, 4)) {
227                     ckb_err("Bad input header\n");
228                     ckb_err("color = %d, i = %d, mode = %d\nOutput (Request): %2.2x %2.2x %2.2x
229 %2.2x\nInput (Reply): %2.2x %2.2x %2.2x %2.2x %2.2x %2.2x %2.2x %2.2x\n", clr, i, mode,
230 comparePacket[0], comparePacket[1], comparePacket[2], comparePacket[3],
231 in_pkt[i][0], in_pkt[i][1], in_pkt[i][2], in_pkt[i][3], in_pkt[i][4], in_pkt[i][5],
232 in_pkt[i][6], in_pkt[i][7]);
233                     in_pkt[2][0] = 0x99;
234                     in_pkt[2][1] = 0x99;
235                     in_pkt[2][2] = 0x99;
236                     in_pkt[2][3] = 0x99;

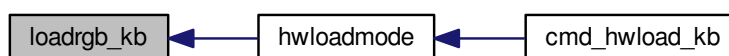
```

```

242         usbrecv(kb, in_pkt[2], in_pkt[2]); // just to find it in the wireshark log
243         return -1;
244     }
245 }
246 // Copy colors to lighting. in_pkt[0] is irrelevant.
247 memcpy(colors[clr], in_pkt[1] + 4, 60);
248 memcpy(colors[clr] + 60, in_pkt[2] + 4, 60);
249 memcpy(colors[clr] + 120, in_pkt[3] + 4, 24);
250 }
251 } else {
252     uchar data_pkt[5][MSG_SIZE] = {
253         { 0x0e, 0x14, 0x02, 0x01, 0x01, mode + 1, 0 },
254         { 0xff, 0x01, 60, 0 },
255         { 0xff, 0x02, 60, 0 },
256         { 0xff, 0x03, 60, 0 },
257         { 0xff, 0x04, 36, 0 },
258     };
259     uchar in_pkt[4][MSG_SIZE] = {
260         { 0xff, 0x01, 60, 0 },
261         { 0xff, 0x02, 60, 0 },
262         { 0xff, 0x03, 60, 0 },
263         { 0xff, 0x04, 36, 0 },
264     };
265     // Write initial packet
266     if(!usbsend(kb, data_pkt[0], 1))
267         return -1;
268     // Read colors
269     for(int i = 1; i < 5; i++){
270         if(!usbrecv(kb, data_pkt[i], in_pkt[i - 1]))
271             return -1;
272         if(memcmp(in_pkt[i - 1], data_pkt[i], 4)){
273             ckb_err("Bad input header\n");
274             return -1;
275         }
276     }
277     // Copy the data back to the mode
278     uint8_t mr[N_KEYS_HW / 2], mg[N_KEYS_HW / 2], mb[
N_KEYS_HW / 2];
279     memcpy(mr, in_pkt[0] + 4, 60);
280     memcpy(mr + 60, in_pkt[1] + 4, 12);
281     memcpy(mg, in_pkt[1] + 16, 48);
282     memcpy(mg + 48, in_pkt[2] + 4, 24);
283     memcpy(mb, in_pkt[2] + 28, 36);
284     memcpy(mb + 36, in_pkt[3] + 4, 36);
285     // Unpack LED data to 8bpc format
286     for(int i = 0; i < N_KEYS_HW; i++){
287         int i_2 = i / 2;
288         uint8_t r, g, b;
289
290         // 3-bit intensities stored in alternate nybbles.
291         if (i & 1) {
292             r = 7 - (mr[i_2] >> 4);
293             g = 7 - (mg[i_2] >> 4);
294             b = 7 - (mb[i_2] >> 4);
295         } else {
296             r = 7 - (mr[i_2] & 0x0F);
297             g = 7 - (mg[i_2] & 0x0F);
298             b = 7 - (mb[i_2] & 0x0F);
299         }
300         // Scale 3-bit values up to 8 bits.
301         light->r[i] = r << 5 | r << 2 | r >> 1;
302         light->g[i] = g << 5 | g << 2 | g >> 1;
303         light->b[i] = b << 5 | b << 2 | b >> 1;
304     }
305 }
306 return 0;
307 }

```

Here is the caller graph for this function:



9.30.2.2 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.

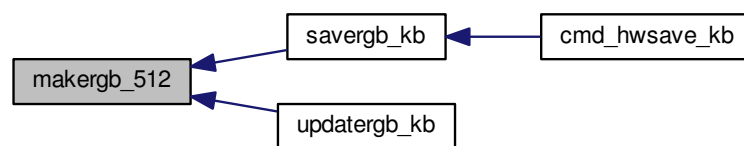
References lighting::b, lighting::g, N_KEYS_HW, and lighting::r.

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:



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

Definition at line 56 of file led_keyboard.c.

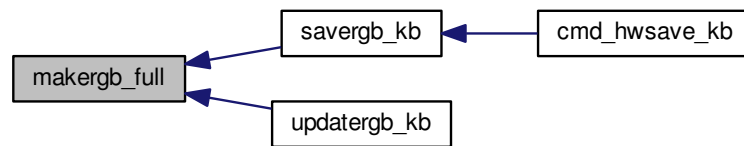
References lighting::b, lighting::g, and lighting::r.

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:



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

Definition at line 24 of file `led_keyboard.c`.

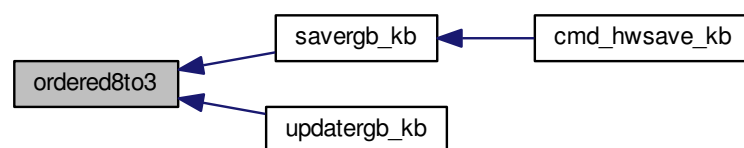
References `bit_reverse_table`.

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:



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

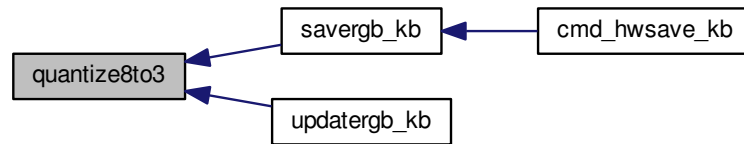
Definition at line 32 of file `led_keyboard.c`.

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

```

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

Here is the caller graph for this function:



9.30.2.6 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->
g, N_KEYS_HW) || memcmp(lhs->b, rhs->b, N_KEYS_HW);
75 }
```

Here is the caller graph for this function:



9.30.2.7 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 `usb send`.

Referenced by `cmd_hwsave_kb()`.

```

139                                     {
140     if(kb->fwversion >= 0x0120){
141         uchar data_pkt[12][MSG_SIZE] = {
142             // Red
143             { 0x7f, 0x01, 60, 0 },
144             { 0x7f, 0x02, 60, 0 },
145             { 0x7f, 0x03, 24, 0 },
146             { 0x07, 0x14, 0x03, 0x01, 0x01, mode + 1, 0x01 },
147             // Green
148             { 0x7f, 0x01, 60, 0 },
149             { 0x7f, 0x02, 60, 0 },
150             { 0x7f, 0x03, 24, 0 },
151         };
152     }
```

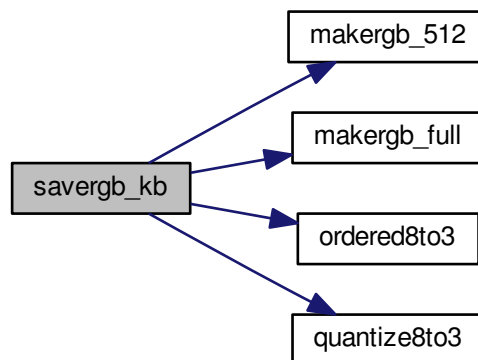


```

151         { 0x07, 0x14, 0x03, 0x01, 0x01, mode + 1, 0x02 },
152         // Blue
153         { 0x7f, 0x01, 60, 0 },
154         { 0x7f, 0x02, 60, 0 },
155         { 0x7f, 0x03, 24, 0 },
156         { 0x07, 0x14, 0x03, 0x01, 0x01, mode + 1, 0x03 }
157     };
158     makergb_full(light, data_pkt);
159     if(!usb_send(kb, data_pkt[0], 12))
160         return -1;
161     if (IS_STRAFE(kb)){ // end save
162         uchar save_end_pkt[MSG_SIZE] = { 0x07, 0x14, 0x04, 0x01, 0x01 };
163         if(!usb_send(kb, save_end_pkt, 1))
164             return -1;
165     }
166     } else {
167         uchar data_pkt[5][MSG_SIZE] = {
168             { 0x7f, 0x01, 60, 0 },
169             { 0x7f, 0x02, 60, 0 },
170             { 0x7f, 0x03, 60, 0 },
171             { 0x7f, 0x04, 36, 0 },
172             { 0x07, 0x14, 0x02, 0x00, 0x01, mode + 1 }
173         };
174         makergb_512(light, data_pkt, kb->dither ? ordered8to3 :
quantize8to3);
175         if(!usb_send(kb, data_pkt[0], 5))
176             return -1;
177     }
178     return 0;
179 }

```

Here is the call graph for this function:



Here is the caller graph for this function:



9.30.2.8 int updatergb_kb (usbdevice * kb, int force)

Definition at line 77 of file led_keyboard.c.

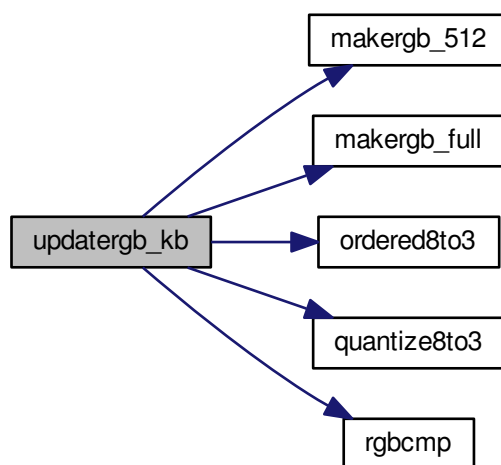
References `usbdevice::active`, `usbprofile::currentmode`, `usbdevice::dither`, `lighting::forceupdate`, `IS_FULLRANGE`, `usbprofile::lastlight`, `usbmode::light`, `makergb_512()`, `makergb_full()`, `MSG_SIZE`, `ordered8to3()`, `usbdevice::profile`, `quantize8to3()`, `rgbcmp()`, `lighting::sidelight`,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

```

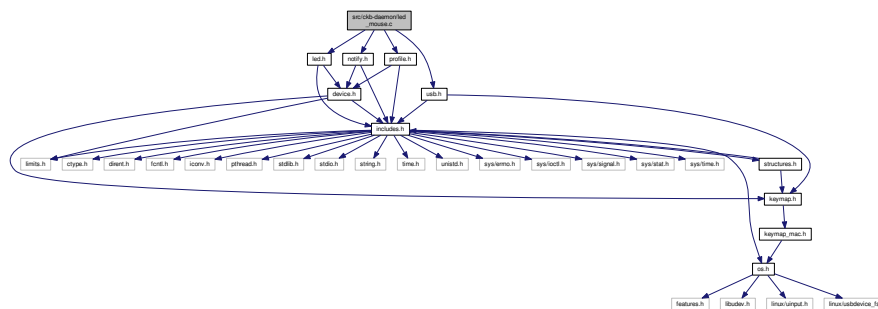
Here is the call graph for this function:



9.30.3 Variable Documentation

9.31 src/ckb-daemon/led_mouse.c File Reference

Include dependency graph for led_mouse.c:



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

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

Referenced by `updatergb_mouse()`.

Generated on Mon Jun 5 2017 23:09:31 for ckb-next by Doxygen

Here is the caller graph for this function:



9.31.1.2 `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:



9.31.1.3 `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`.

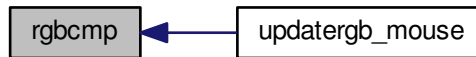
Referenced by `updaterrgb_mouse()`.

```

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:



9.31.1.4 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:



9.31.1.5 int updatergb_mouse (usbdevice * kb, int force)

Definition at line 20 of file led_mouse.c.

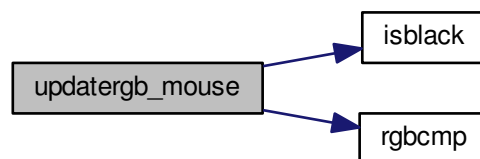
References `usbdevice::active`, `lighting::b`, `usbprofile::currentmode`, `lighting::forceupdate`, `lighting::g`, `isblack()`, `usbprofile::lastlight`, `LED_MOUSE`, `usbmode::light`, `MSG_SIZE`, `N_MOUSE_ZONES`, `usbdevice::profile`, `lighting::r`, `rgbcmp()`, and `usbsend`.

```

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->q[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:



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

```

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

```



```

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

```

9.32.1.2 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     // Check PID, quit if already running
139     char pidpath[strlen(devpath) + 6];
140     snprintf(pidpath, sizeof(pidpath), "%s0/pid", devpath);
141     FILE* pidfile = fopen(pidpath, "r");
142     if(pidfile){
143         pid_t pid;
144         fscanf(pidfile, "%d", &pid);
145         fclose(pidfile);
146         if(pid > 0){
147             // kill -s 0 checks if the PID is active but doesn't send a signal
148             if(!kill(pid, 0)){
149                 ckb_fatal_nofile("ckb-daemon is already running (PID %d). Try 'killall
150                 ckb-daemon'.\n", pid);
151                 ckb_fatal_nofile("(If you're certain the process is dead, delete %s and try

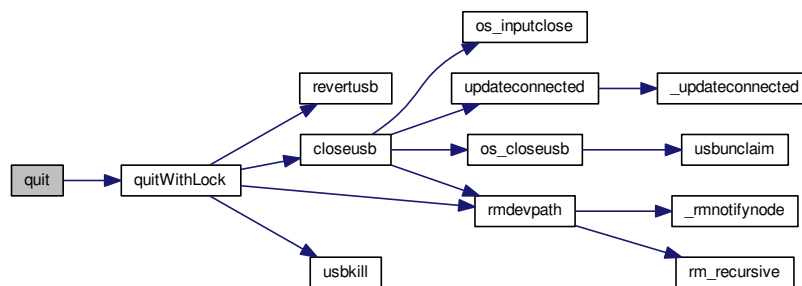
```

```

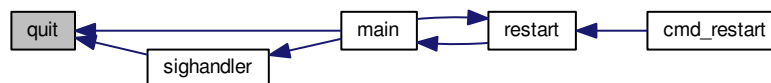
        again)\n", pidpath);
146         return 0;
147     }
148 }
149 }
150
151 // Read parameters
152 int forceroot = 1;
153 for(int i = 1; i < argc; i++){
154     char* argument = argv[i];
155     unsigned newgid;
156     char hwload[7];
157     if(sscanf(argument, "--gid=%u", &newgid) == 1){
158         // 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:



9.32.1.4 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()`.

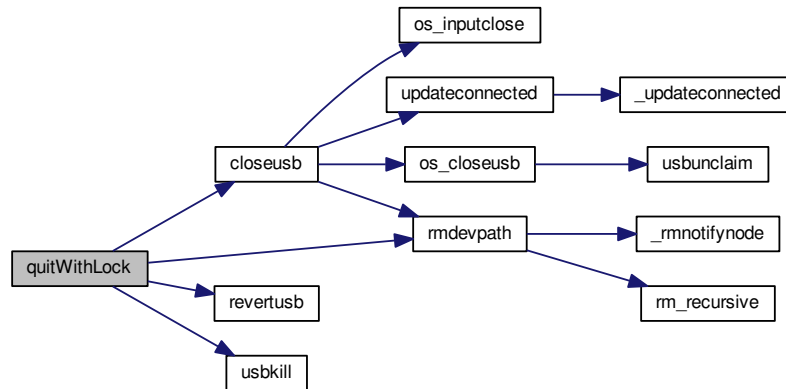
Referenced by `quit()`, and `restart()`.

```

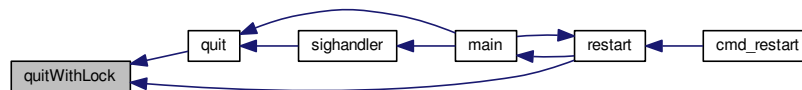
40     {
41         // Abort any USB resets in progress
42         reset_stop = 1;
43         for(int i = 1; i < DEV_MAX; i++){
44             // Before closing, set all keyboards back to HID input mode so that the stock driver can still talk
45             to them
46             if (mut) pthread_mutex_lock(devmutex + 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:



9.32.1.5 int restart ()

Definition at line 228 of file main.c.

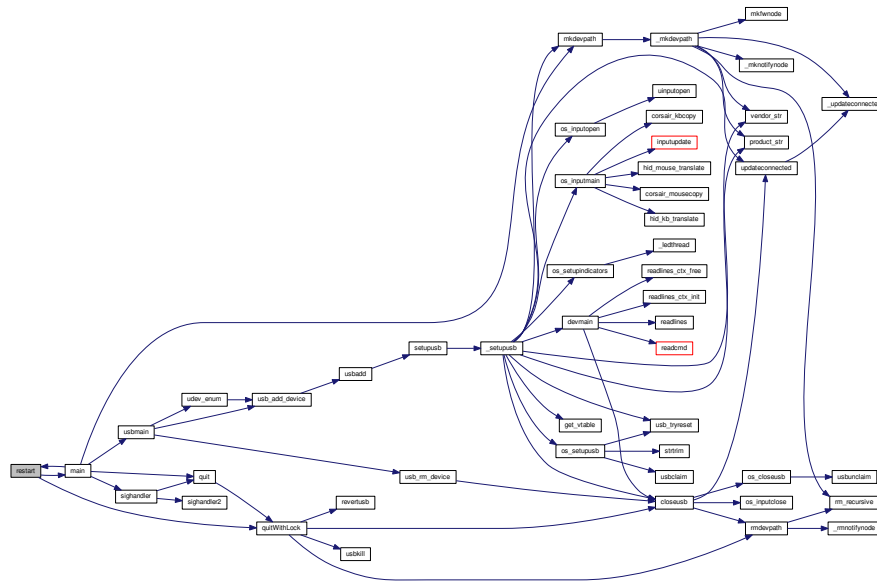
References `ckb_err`, `main()`, `main_ac`, `main_av`, and `quitWithLock()`.

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     }
  
```

Here is the call graph for this function:



Here is the caller graph for this function:



9.32.1.6 void sighandler (int *type*)

Definition at line 62 of file main.c.

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

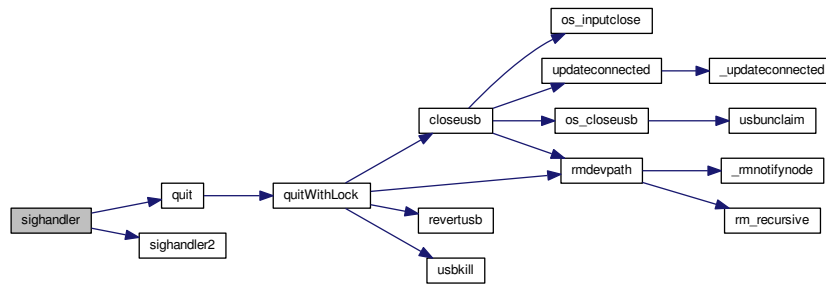
Referenced by main().

```

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

```

Here is the call graph for this function:



Here is the caller graph for this function:



9.32.1.7 void sighandler2 (int type)

Definition at line 57 of file main.c.

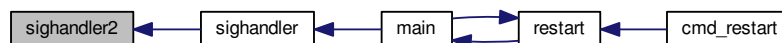
Referenced by sighandler().

```

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:



9.32.1.8 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     }

```


9.32.2 Variable Documentation

9.32.2.1 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\(\)](#).

9.32.2.2 int hwload_mode

Definition at line 7 of file device.c.

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

9.32.2.3 int main_ac [static]

Definition at line 7 of file main.c.

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

9.32.2.4 char** main_av [static]

Definition at line 8 of file main.c.

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

9.32.2.5 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.

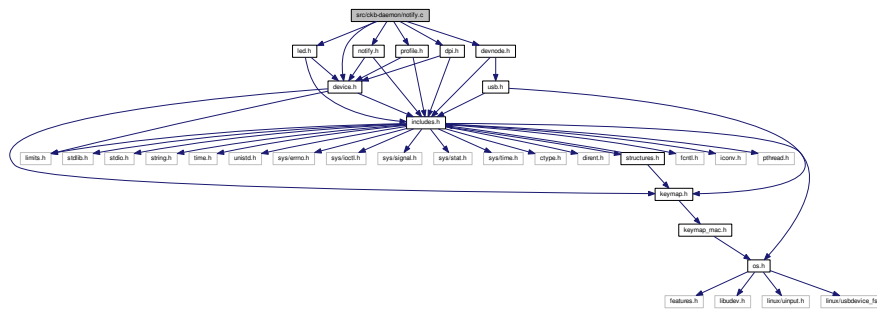
Definition at line 25 of file usb.c.

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

9.33 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"
```

Include dependency graph for notify.c:



Macros

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

Functions

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

9.33.1 Macro Definition Documentation

9.33.1.1 `#define HW_STANDARD`

Value:

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

Definition at line 83 of file notify.c.

Referenced by `_cmd_get()`.

9.33.1.2 `#define HWMODE_OR_RETURN(kb, index)`

Value:

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

Definition at line 73 of file notify.c.

9.33.2 Function Documentation

9.33.2.1 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()`, `nprintkey()`, `printdpi()`, `printrgb()`, `usbdevice::profile`, and `dpiset::snap`.

Referenced by `cmd_get()`.

```

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

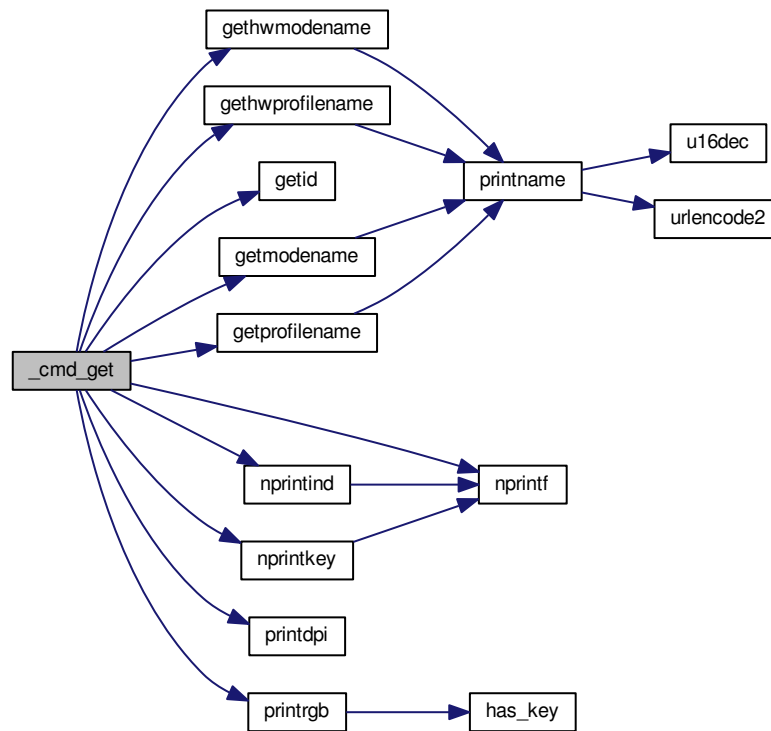
```

```

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

```

Here is the call graph for this function:



Here is the caller graph for this function:



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

Definition at line 215 of file notify.c.

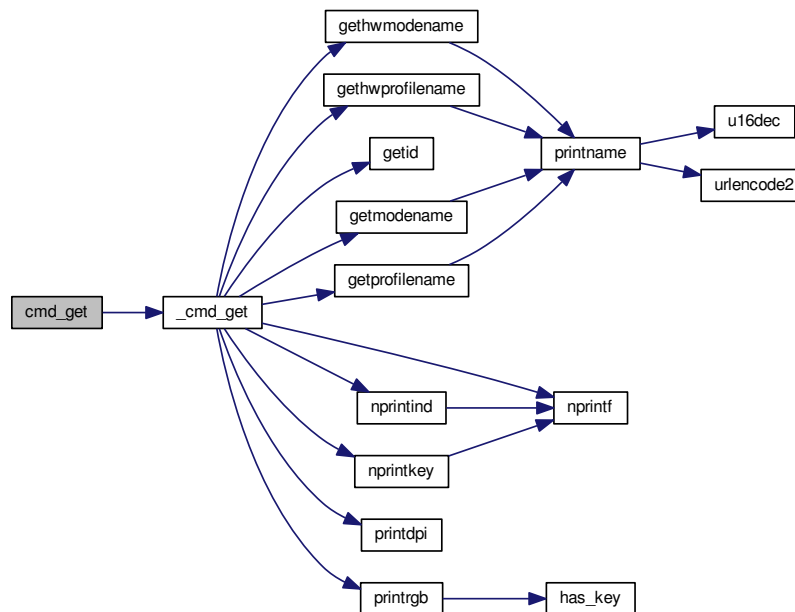
References `_cmd_get()`, and `imutex`.

```

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:



9.33.2.3 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 }

```

9.33.2.4 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()`.

[illegible]

9.33.2.6 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().

```

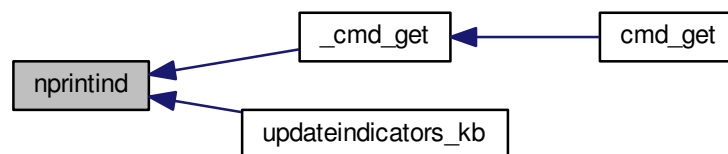
43                                     {
44     const char* name = 0;
45     switch(led){
46     case I_NUM:
47         name = "num";
48         break;
49     case I_CAPS:
50         name = "caps";
51         break;
52     case I_SCROLL:
53         name = "scroll";
54         break;
55     default:
56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



9.33.2.7 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)

```



```

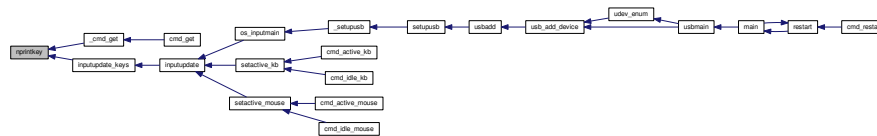
38     nprintf(kb, nnumber, 0, "key %c%s\n", down ? '+' : '-', map->name);
39     else
40     nprintf(kb, nnumber, 0, "key %c#%d\n", down ? '+' : '-', keyindex);
41 }

```

Here is the call graph for this function:



Here is the caller graph for this function:



9.33.2.8 int restart ()

Definition at line 228 of file main.c.

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

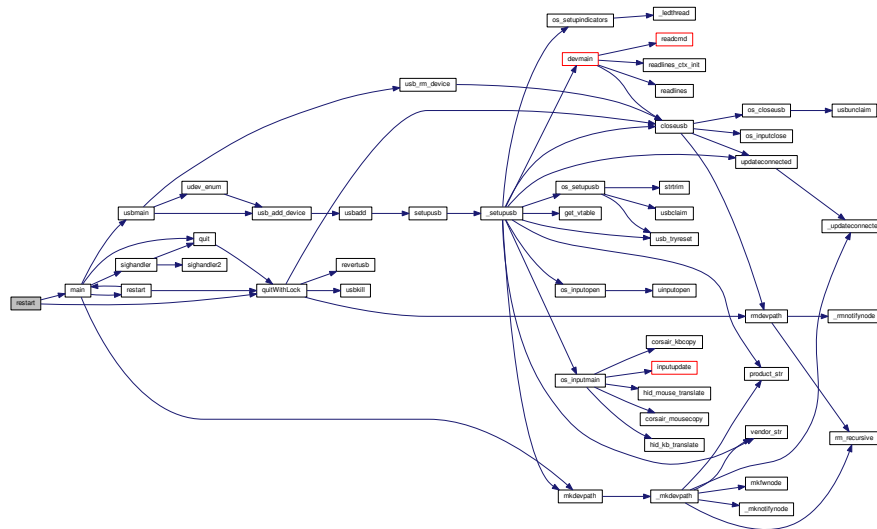
Referenced by `cmd_restart()`, and `main()`.

```

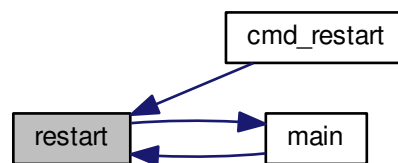
228     {
229     ckb_err("restart called, running quit without mutex-lock.\n");
230     quitWithLock(0);
231     return main(main_ac, main_av);
232 }

```

Here is the call graph for this function:

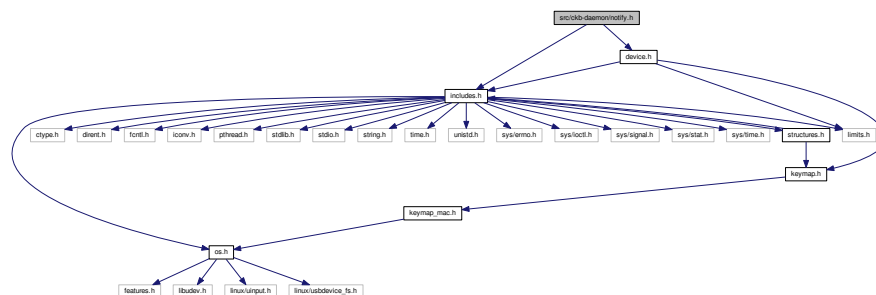


Here is the caller graph for this function:

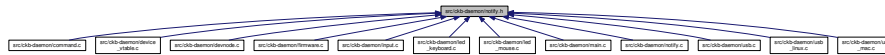


9.34 src/ckb-daemon/notify.h File Reference

```
#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)

9.34.1 Function Documentation

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

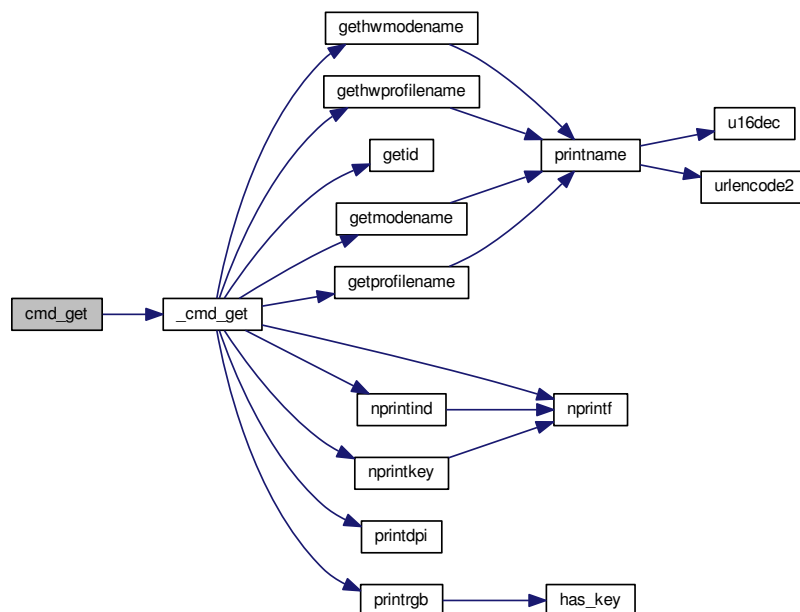
Definition at line 215 of file notify.c.

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:

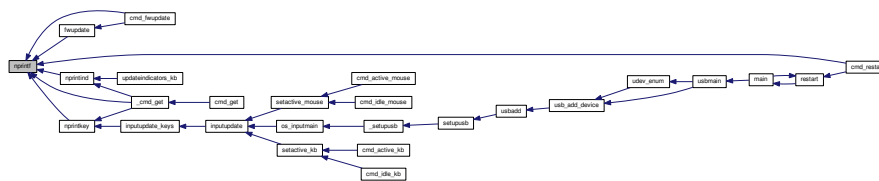



```

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:



9.34.1.5 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 }

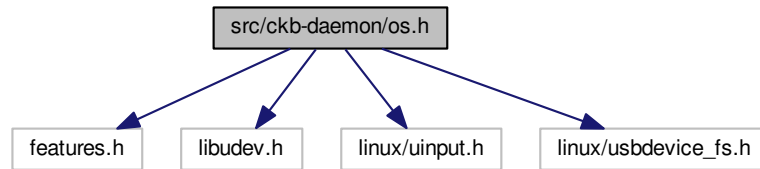
```

Here is the call graph for this function:




```
#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`

9.35.1 Macro Definition Documentation

9.35.1.1 `#define _DEFAULT_SOURCE`

Definition at line 22 of file os.h.

9.35.1.2 `#define _GNU_SOURCE`

Definition at line 26 of file os.h.

9.35.1.3 `#define euid_guard_start`

Definition at line 40 of file os.h.

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

9.35.1.4 `#define euid_guard_stop`

Definition at line 41 of file os.h.

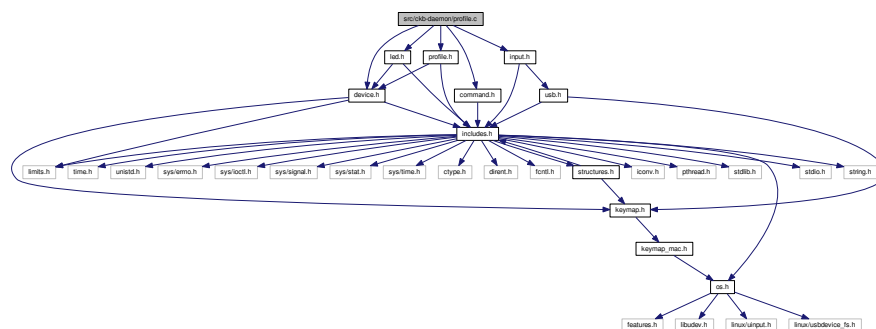
Referenced by `mkdevpath()`, `mknotifynode()`, `rmdevpath()`, `rmnotifynode()`, and `updateconnected()`.

9.35.1.5 #define UINPUT_VERSION 2

Definition at line 35 of file os.h.

9.36 src/ckb-daemon/profile.c File Reference

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



Functions

- void [urldecode2](#) (char *dst, const char *src)
- void [urlencode2](#) (char *dst, const char *src)
- int [setid](#) (usbid *id, const char *guid)
- char * [getid](#) (usbid *id)
- void [u16enc](#) (char *in, ushort *out, size_t *srclen, size_t *dstlen)
- void [u16dec](#) (ushort *in, char *out, size_t *srclen, size_t *dstlen)
- void [cmd_name](#) (usbdevice *kb, usbmode *mode, int dummy1, int dummy2, const char *name)
- void [cmd_profilename](#) (usbdevice *kb, usbmode *mode, int dummy1, int dummy2, int dummy3, const char *name)
- char * [printname](#) (ushort *name, int length)
- char * [getmodename](#) (usbmode *mode)
- char * [getprofilename](#) (usbprofile *profile)
- char * [gethwmodename](#) (hwprofile *profile, int index)
- char * [gethwprofilename](#) (hwprofile *profile)
- void [cmd_id](#) (usbdevice *kb, usbmode *mode, int dummy1, int dummy2, const char *id)
- void [cmd_profileid](#) (usbdevice *kb, usbmode *mode, int dummy1, int dummy2, const char *id)
- static void [initmode](#) (usbmode *mode)
- void [allocprofile](#) (usbdevice *kb)
- int [loadprofile](#) (usbdevice *kb)
- static void [freemode](#) (usbmode *mode)
- void [cmd_erase](#) (usbdevice *kb, usbmode *mode, int dummy1, int dummy2, const char *dummy3)
- static void [_freeprofile](#) (usbdevice *kb)
- void [cmd_eraseprofile](#) (usbdevice *kb, usbmode *mode, int dummy1, int dummy2, int dummy3, const char *dummy4)
- void [freeprofile](#) (usbdevice *kb)
- void [hwtonative](#) (usbprofile *profile, hwprofile *hw, int modecount)
- void [nativetohw](#) (usbprofile *profile, hwprofile *hw, int modecount)

Variables

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

9.36.1 Function Documentation

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

Definition at line 210 of file profile.c.

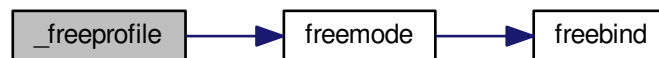
References `freemode()`, `usbprofile::mode`, `MODE_COUNT`, and `usbdevice::profile`.

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

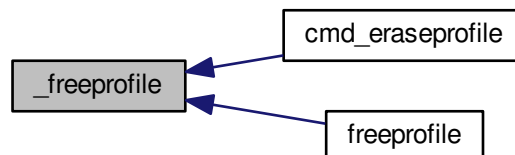
```

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:



9.36.1.2 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`.

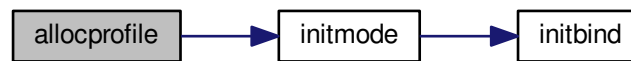
Referenced by `cmd_eraseprofile()`.

```

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:



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

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

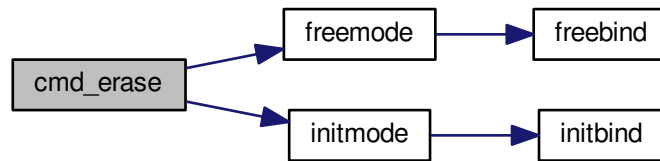
References `freemode()`, `imutex`, and `initmode()`.

```

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:



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

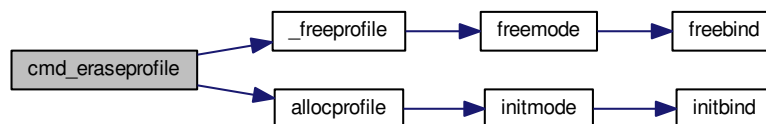
Definition at line 221 of file profile.c.

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:



9.36.1.5 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:



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

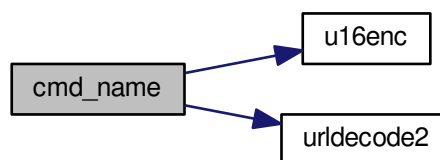
Definition at line 117 of file profile.c.

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:



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

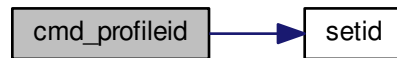
Definition at line 167 of file profile.c.

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:



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

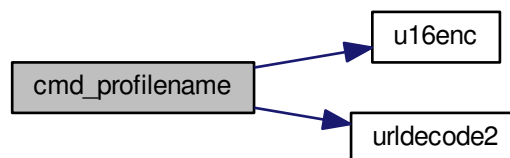
Definition at line 124 of file profile.c.

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:



9.36.1.9 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().

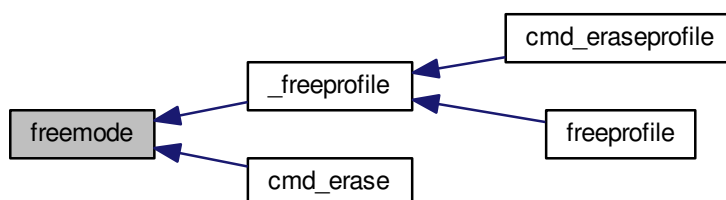
```

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:



9.36.1.10 void freeprofile (usbdevice * kb)

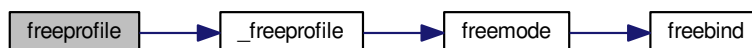
Definition at line 228 of file profile.c.

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:



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

Definition at line 152 of file profile.c.

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

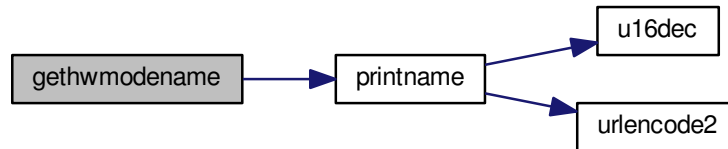
Referenced by _cmd_get().

```

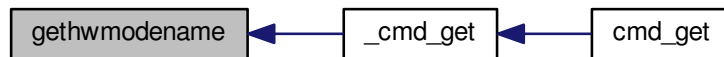
152
153     return printname(profile->name[index + 1], MD_NAME_LEN);
154 }

```

Here is the call graph for this function:



Here is the caller graph for this function:



9.36.1.12 char* gethwprofilename (hwprofile * profile)

Definition at line 156 of file profile.c.

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

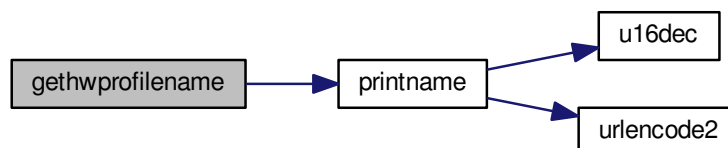
Referenced by _cmd_get().

```

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:



9.36.1.13 `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:



9.36.1.14 `char* getmodename (usbmode * mode)`

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

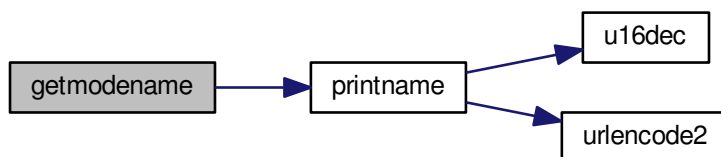
References `MD_NAME_LEN`, `usbmode::name`, and `printname()`.

Referenced by `_cmd_get()`.

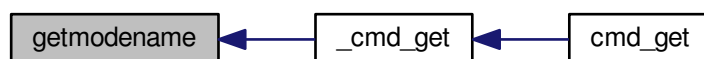
```

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:



9.36.1.15 `char* getprofilename (usbprofile * profile)`

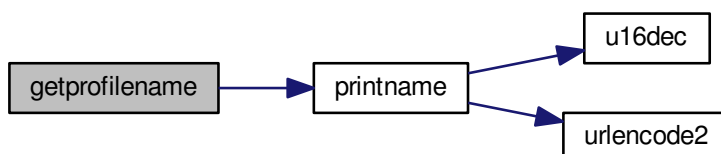
Definition at line 148 of file `profile.c`.

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

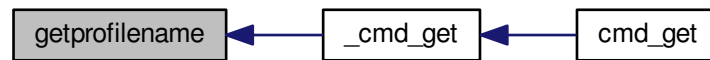
Referenced by `_cmd_get()`.

```
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:



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

Definition at line 235 of file profile.c.

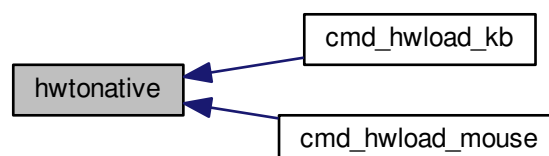
References usbmode::dpi, hwprofile::dpi, dpiset::forceupdate, lighting::forceupdate, usbmode::id, usbprofile::id, hwprofile::id, usbprofile::lastdpi, usbprofile::lastlight, usbmode::light, hwprofile::light, MD_NAME_LEN, usbprofile::mode, usbmode::name, usbprofile::name, hwprofile::name, and PR_NAME_LEN.

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:



9.36.1.17 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().

```

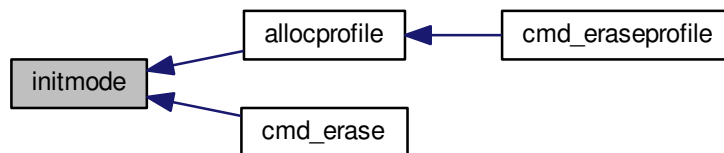
175
176     memset(mode, 0, sizeof(*mode));
177     mode->light.forceupdate = 1;
178     mode->dpi.forceupdate = 1;
179     initbind(&mode->bind);
180 }

```

Here is the call graph for this function:



Here is the caller graph for this function:



9.36.1.18 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 }

```

9.36.1.19 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

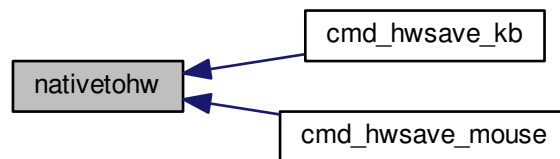
```

```

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:



9.36.1.20 char* printname (ushort * name, int length)

Definition at line 132 of file profile.c.

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

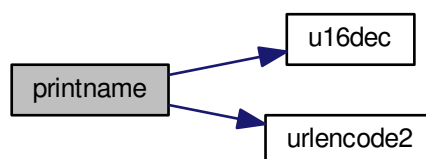
Referenced by `gethwmodename()`, `gethwprofilename()`, `getmodename()`, and `getprofilename()`.

```

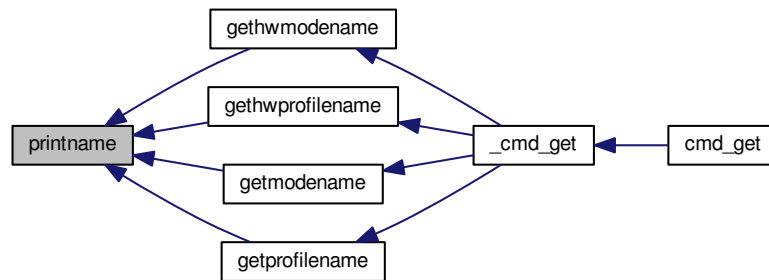
132     {
133         // Convert the name to UTF-8
134         char* buffer = calloc(1, length * 4 - 3);
135         size_t srclen = length, dstlen = length * 4 - 4;
136         u16dec(name, buffer, &srclen, &dstlen);
137         // URL-encode it
138         char* buffer2 = malloc(strlen(buffer) * 3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



9.36.1.21 `int setid (usbid * id, const char * guid)`

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

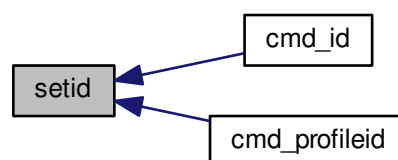
References `usbid::guid`.

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-%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     }
  
```

Here is the caller graph for this function:



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

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

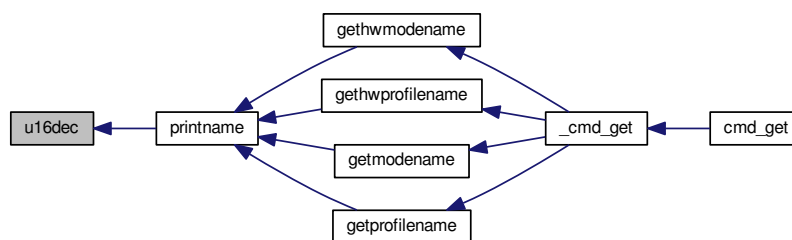
References utf16to8.

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:



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

Definition at line 97 of file profile.c.

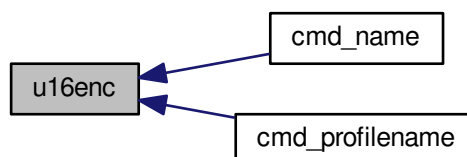
References utf8to16.

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:



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

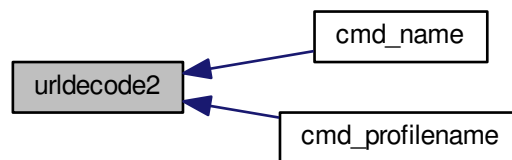
Definition at line 8 of file profile.c.

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:



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

Definition at line 37 of file profile.c.

Referenced by printname().

```

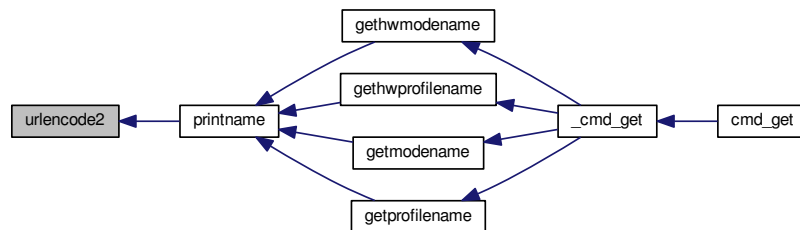
37                                     {
38    char s;
39    while((s = *src++){
40        if(s <= ',' || s == '/' ||
41            (s >= ':' && s <= '@') ||
42            s == '[' || s == ']' ||
43            s >= 0x7F){
44            char a = s >> 4, b = s & 0xF;
45            if(a >= 10)
46      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



9.36.2 Variable Documentation

9.36.2.1 `iconv_t utf16to8 = 0` [static]

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

Referenced by `u16dec()`.

9.36.2.2 `iconv_t utf8to16 = 0` [static]

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

Referenced by `u16enc()`.

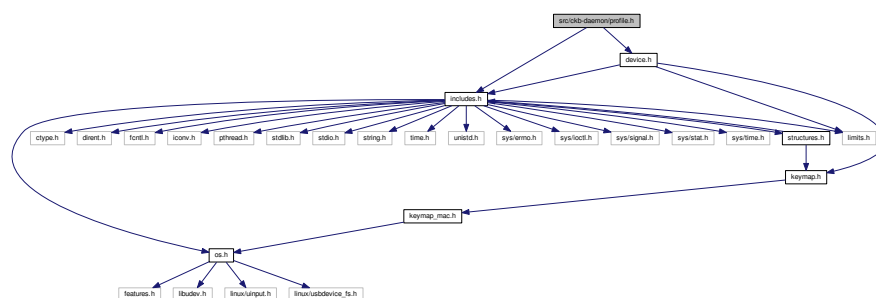
9.37 `src/ckb-daemon/profile.h` File Reference

```

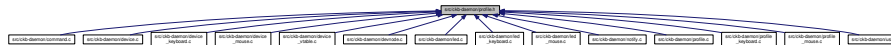
#include "includes.h"
#include "device.h"

```

Include dependency graph for `profile.h`:



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



Macros

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

Functions

- void **allocprofile** (usbdevice *kb)
- int **loadprofile** (usbdevice *kb)
- void **freeprofile** (usbdevice *kb)
- void **cmd_erase** (usbdevice *kb, usbmode *mode, int dummy1, int dummy2, const char *dummy3)
- void **cmd_eraseprofile** (usbdevice *kb, usbmode *dummy1, int dummy2, int dummy3, const char *dummy4)
- void **cmd_name** (usbdevice *kb, usbmode *mode, int dummy1, int dummy2, const char *name)
- void **cmd_profilename** (usbdevice *kb, usbmode *dummy1, int dummy2, int dummy3, const char *name)
- char * **getmodename** (usbmode *mode)
- char * **getprofilename** (usbprofile *profile)
- char * **gethwmodename** (hwprofile *profile, int index)
- char * **gethwprofilename** (hwprofile *profile)
- int **setid** (usbid *id, const char *guid)
- char * **getid** (usbid *id)
- void **hwtonative** (usbprofile *profile, hwprofile *hw, int modecount)
- void **nativetohw** (usbprofile *profile, hwprofile *hw, int modecount)
- void **cmd_id** (usbdevice *kb, usbmode *mode, int dummy1, int dummy2, const char *id)
- void **cmd_profileid** (usbdevice *kb, usbmode *mode, int dummy1, int dummy2, const char *id)
- int **cmd_hwload_kb** (usbdevice *kb, usbmode *dummy1, int dummy2, int apply, const char *dummy3)
- int **cmd_hwload_mouse** (usbdevice *kb, usbmode *dummy1, int dummy2, int apply, const char *dummy3)
- int **cmd_hwsave_kb** (usbdevice *kb, usbmode *dummy1, int dummy2, int dummy3, const char *dummy4)
- int **cmd_hwsave_mouse** (usbdevice *kb, usbmode *dummy1, int dummy2, int dummy3, const char *dummy4)

9.37.1 Macro Definition Documentation

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

Definition at line 52 of file profile.h.

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

9.37.2 Function Documentation

9.37.2.1 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`.

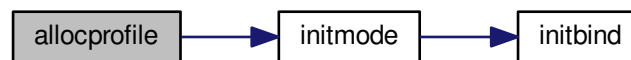
Referenced by cmd eraseprofile().

```

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:



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

Definition at line 203 of file profile.c.

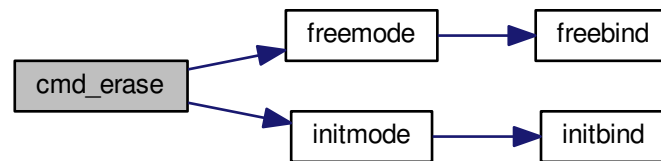
References `freemode()`, `imutex`, and `initmode()`.

```

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:



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

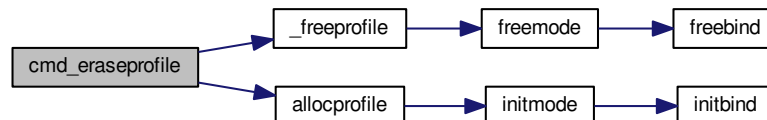
Definition at line 221 of file profile.c.

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:



9.37.2.4 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`.

```

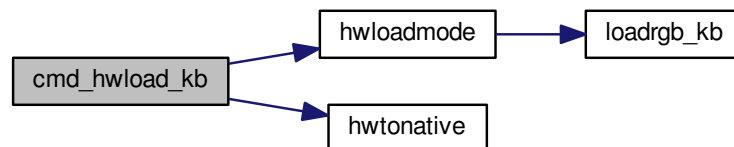
16                                     {
17     DELAY_LONG(kb);
18     hwprofile* hw = calloc(1, sizeof(hwprofile));
19     // Ask for profile and mode IDs
20     uchar data_pkt[2][MSG_SIZE] = {
21         { 0x0e, 0x15, 0x01, 0 },
22         { 0x0e, 0x16, 0x01, 0 }
23     };
24     uchar in_pkt[MSG_SIZE];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



9.37.2.5 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        }

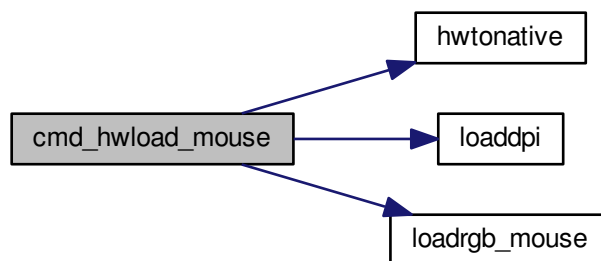
```

```

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:



9.37.2.6 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 `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(!usbsend(kb, data_pkt[0], 1))
74             return -1;
75     }
76     // Save the IDs
77     for(int i = 0; i <= modes; i++){
78         data_pkt[1][3] = i;
79         memcpy(data_pkt[1] + 4, hw->id + i, sizeof(usbid));
80         if(!usbsend(kb, data_pkt[1], 1))
81             return -1;
82     }

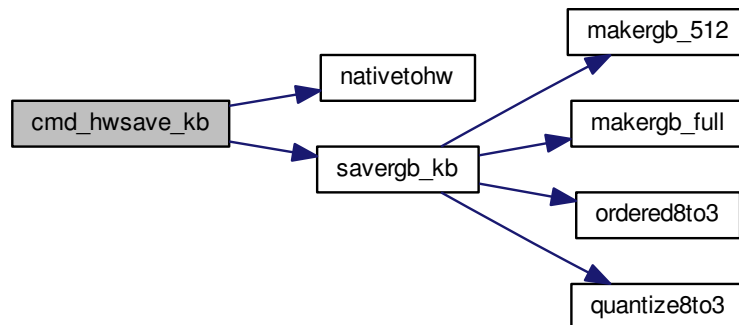
```

```

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:



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

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

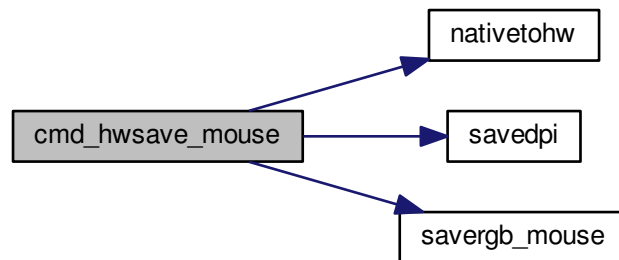
References `DELAY_LONG`, `hwprofile::dpi`, `usbdevice::hw`, `hwprofile::id`, `hwprofile::light`, `MD_NAME_LEN`, `MSG_SIZE`, `hwprofile::name`, `nativetohw()`, `usbdevice::profile`, `savedpi()`, `savergb_mouse()`, and `usb send`.

```

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:



9.37.2.8 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:



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

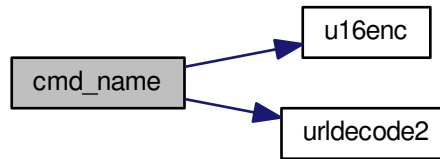
Definition at line 117 of file profile.c.

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:



9.37.2.10 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:



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

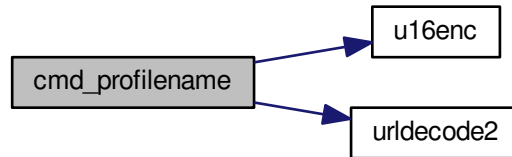
Definition at line 124 of file profile.c.

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:



9.37.2.12 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:



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

Definition at line 152 of file profile.c.

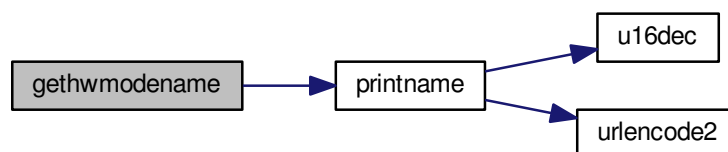
References `MD_NAME_LEN`, `hwprofile::name`, and `printname()`.

Referenced by `_cmd_get()`.

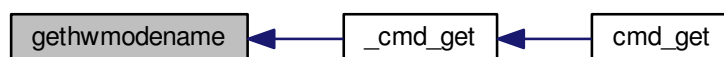
```

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:



9.37.2.14 `char* gethwprofilename (hwprofile * profile)`

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

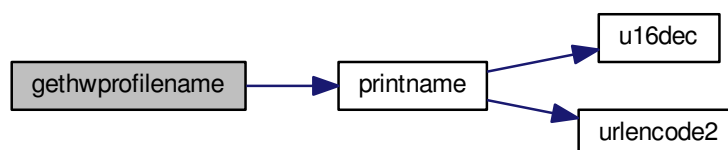
References `MD_NAME_LEN`, `hwprofile::name`, and `printname()`.

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:



9.37.2.15 `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%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:



9.37.2.16 `char* getmodename (usbmode * mode)`

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

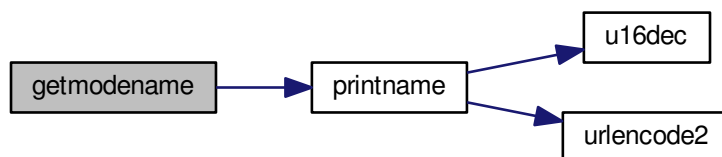
References `MD_NAME_LEN`, `usbmode::name`, and `printname()`.

Referenced by `_cmd_get()`.

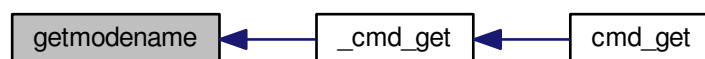
```

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:



9.37.2.17 `char* getprofilename (usbprofile * profile)`

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

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

Referenced by `_cmd_get()`.

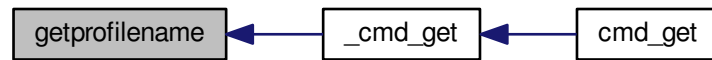
```

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:



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

Definition at line 235 of file profile.c.

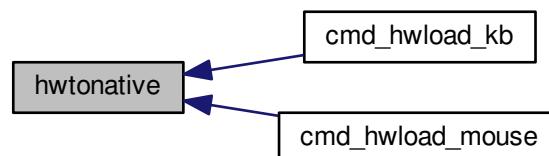
References usbmode::dpi, hwprofile::dpi, dpiset::forceupdate, lighting::forceupdate, usbmode::id, usbprofile::id, hwprofile::id, usbprofile::lastdpi, usbprofile::lastlight, usbmode::light, hwprofile::light, MD_NAME_LEN, usbprofile::mode, usbmode::name, usbprofile::name, hwprofile::name, and PR_NAME_LEN.

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:



9.37.2.19 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     }

```

9.37.2.20 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.

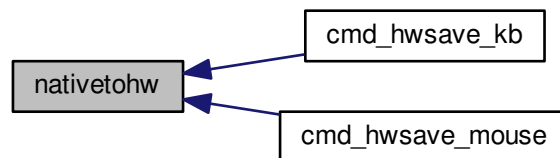
Referenced by cmd_hwsave_kb(), and cmd_hwsave_mouse().

```

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:



9.37.2.21 int setid (usbid * id, const char * guid)

Definition at line 64 of file profile.c.

References usbid::guid.

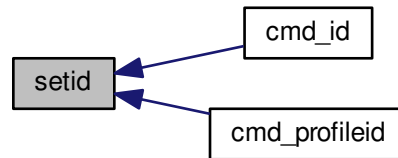
Referenced by cmd_id(), and cmd_profileid().

```

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     }

```

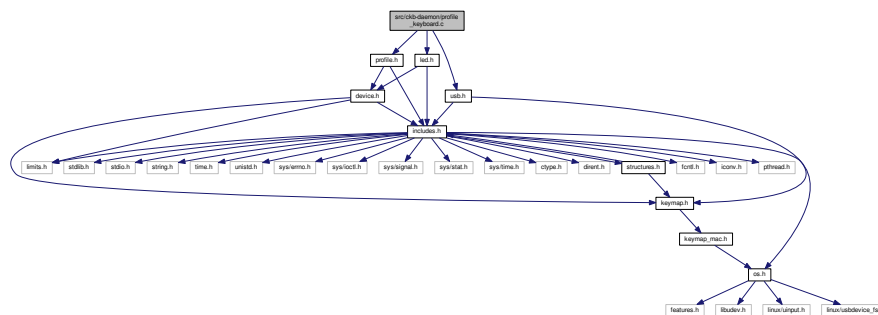
Here is the caller graph for this function:



9.38 src/ckb-daemon/profile_keyboard.c File Reference

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

Include dependency graph for profile_keyboard.c:



Functions

- static int [hwloadmode](#) (usbdevice *kb, hwprofile *hw, int mode)
- int [cmd_hwload_kb](#) (usbdevice *kb, usbmode *dummy1, int dummy2, int apply, const char *dummy3)
- int [cmd_hwsave_kb](#) (usbdevice *kb, usbmode *dummy1, int dummy2, int dummy3, const char *dummy4)

9.38.1 Function Documentation

9.38.1.1 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](#).

```

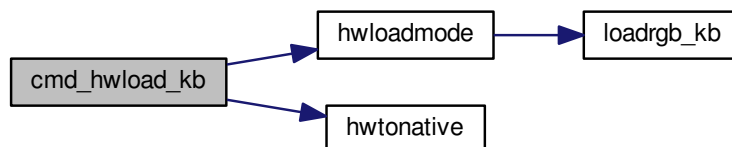
16
17     DELAY_LONG(kb);
18     hwprofile* hw = calloc(1, sizeof(hwprofile));
19     // Ask for profile and mode IDs
20     uchar data_pkt[2][MSG_SIZE] = {
21         { 0x0e, 0x15, 0x01, 0 },
  
```

```

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:



9.38.1.2 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(!usbsend(kb, data_pkt[0], 1))

```

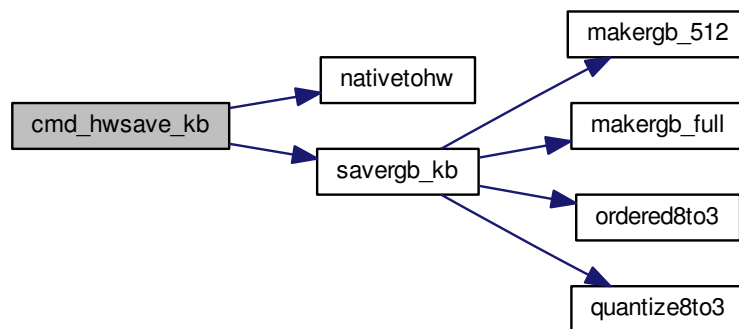


```

74         return -1;
75     }
76     // Save the IDs
77     for(int i = 0; i <= modes; i++){
78         data_pkt[1][3] = i;
79         memcpy(data_pkt[1] + 4, hw->id + i, sizeof(usbid));
80         if(!usbSend(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:



9.38.1.3 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 }

```


9.39.1.1 int cmd_hwload_mouse (usbdevice * kb, usbmode * dummy1, int dummy2, int apply, const char * dummy3)

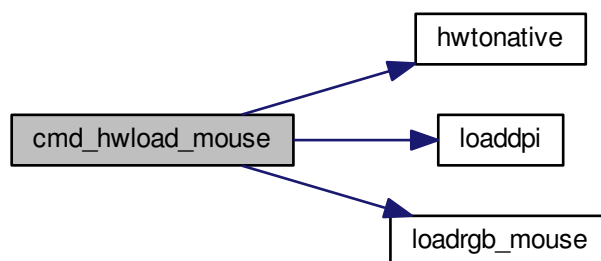
Definition at line 6 of file profile_mouse.c.

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:



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

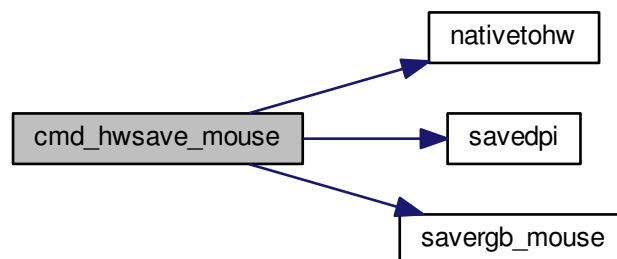
Definition at line 50 of file `profile_mouse.c`.

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:

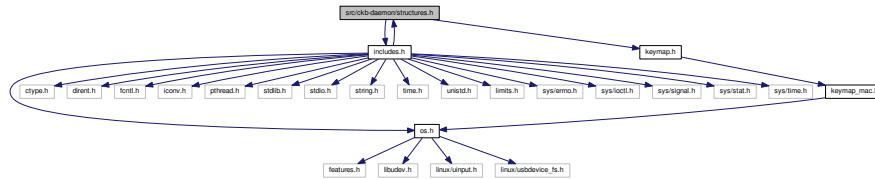


9.40 src/ckb-daemon/structures.h File Reference

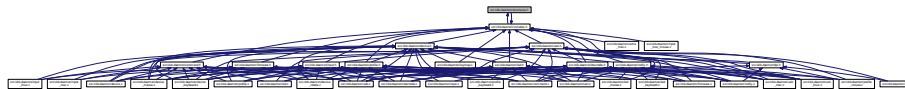
```

#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 40`
- `#define SERIAL_LEN 34`
- `#define MSG_SIZE 64`
- `#define IFACE_MAX 4`

Variables

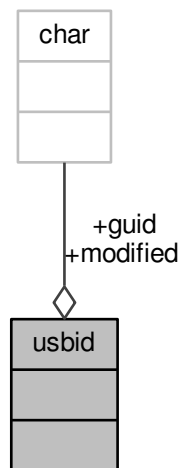
- const union `devcmd vtable_keyboard`
RGB keyboard vtable holds functions for each device type.
- const union `devcmd vtable_keyboard_nonrgb`
- const union `devcmd vtable_mouse`

9.40.1 Data Structure Documentation

9.40.1.1 struct usbid

Definition at line 8 of file structures.h.

Collaboration diagram for usbid:



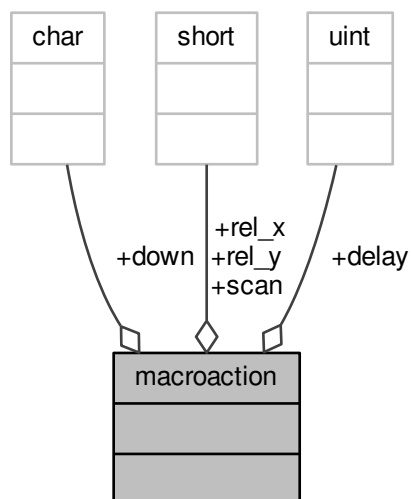
Data Fields

char	guid[16]	
char	modified[4]	

9.40.1.2 struct macroaction

Definition at line 27 of file structures.h.

Collaboration diagram for macroaction:



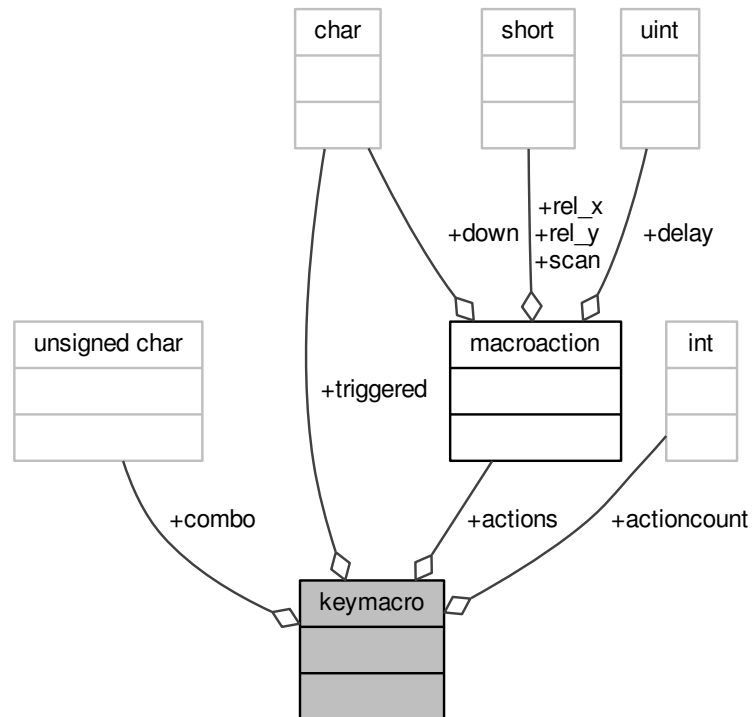
Data Fields

uint	delay	
char	down	
short	rel_x	
short	rel_y	
short	scan	

9.40.1.3 struct keymacro

Definition at line 35 of file structures.h.

Collaboration diagram for keymacro:



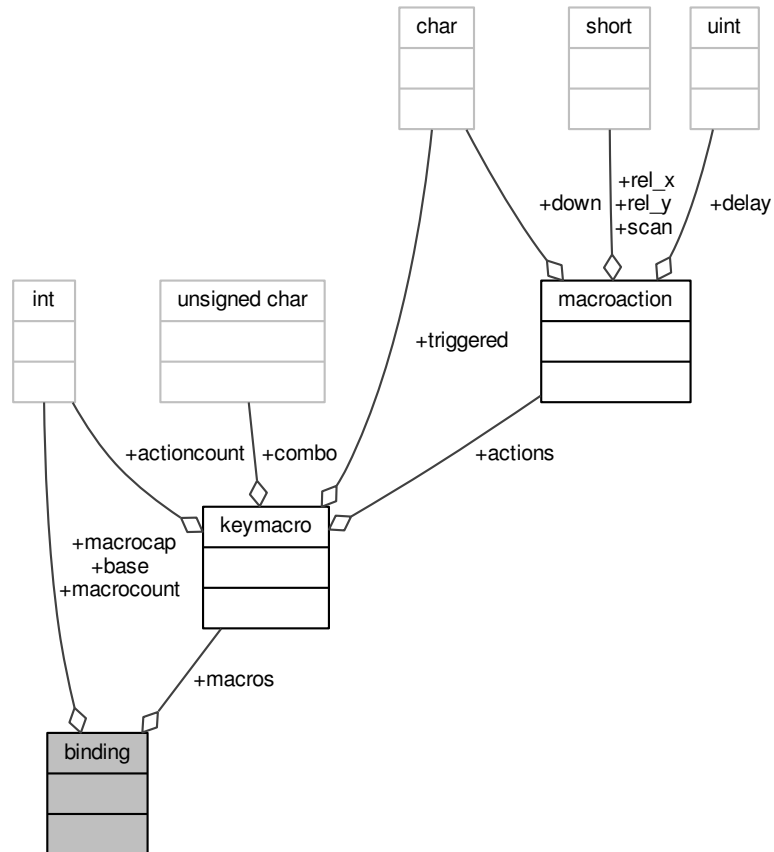
Data Fields

int	actioncount	
macroaction *	actions	
uchar	combo[(((152+3+12)+25)+7)/8]]	
char	triggered	

9.40.1.4 struct binding

Definition at line 43 of file structures.h.

Collaboration diagram for binding:



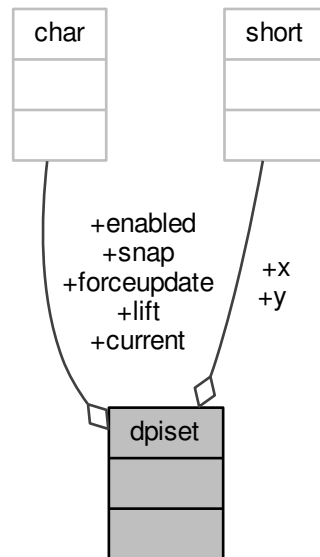
Data Fields

int	base[((152+3+12)+25)]	
int	macrocap	
int	macrocount	
keymacro *	macros	

9.40.1.5 struct dpiset

Definition at line 57 of file structures.h.

Collaboration diagram for dpiset:



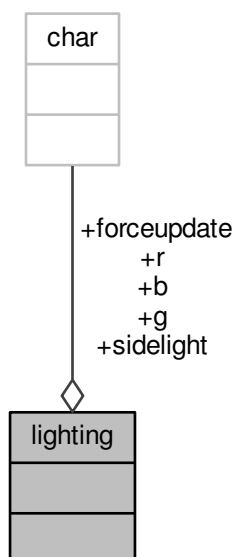
Data Fields

uchar	current	
uchar	enabled	
uchar	forceupdate	
uchar	lift	
uchar	snap	
ushort	x[6]	
ushort	y[6]	

9.40.1.6 struct lighting

Definition at line 73 of file structures.h.

Collaboration diagram for lighting:



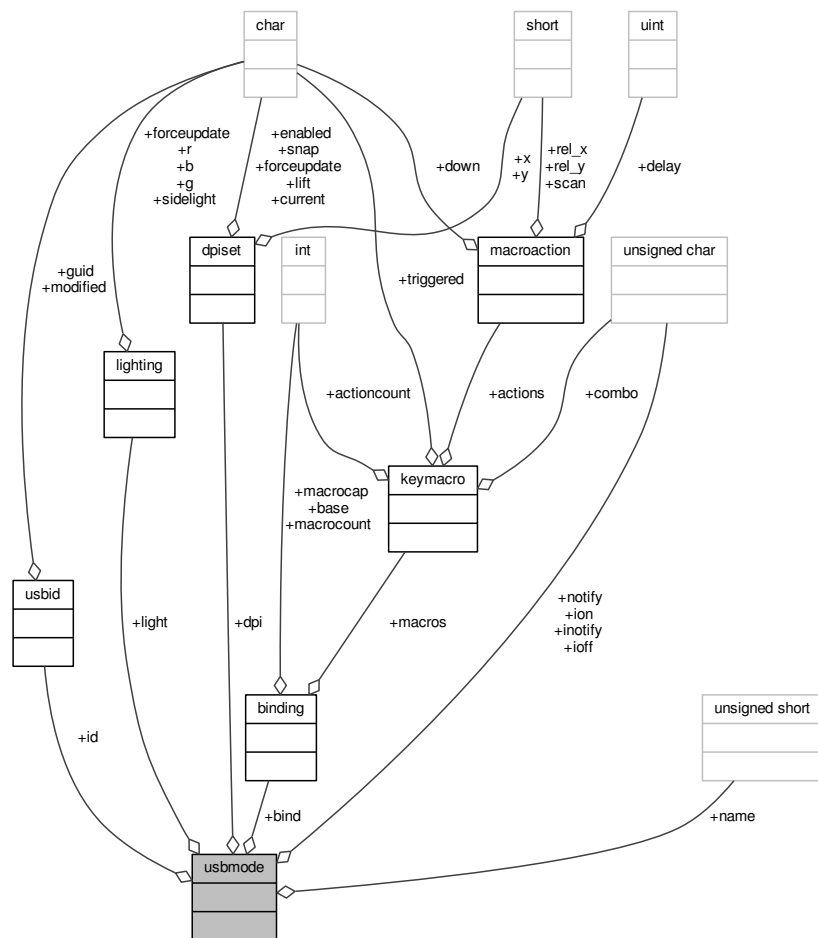
Data Fields

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

9.40.1.7 struct usbmode

Definition at line 83 of file structures.h.

Collaboration diagram for usbmode:



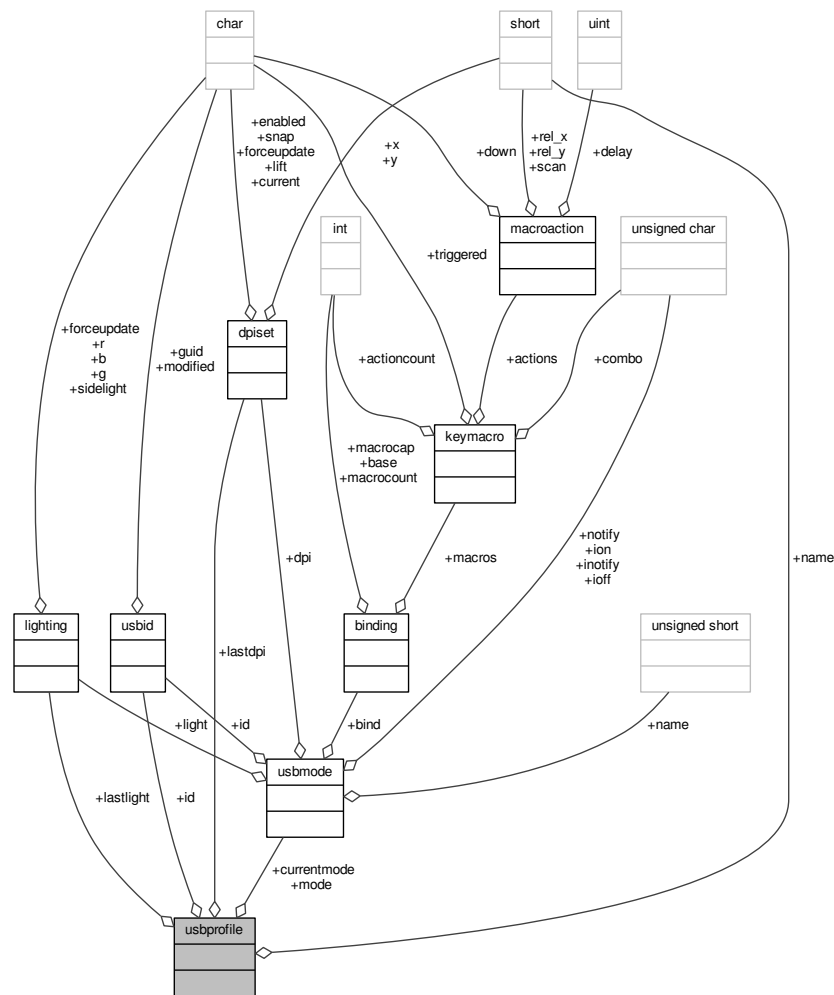
Data Fields

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

9.40.1.8 struct usbprofile

Definition at line 101 of file structures.h.

Collaboration diagram for usbprofile:



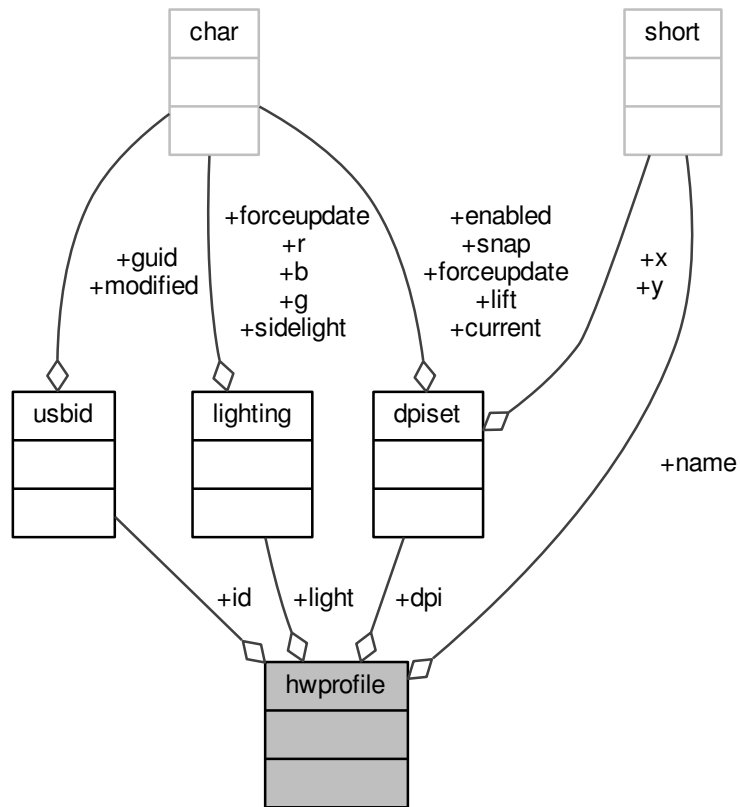
Data Fields

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

9.40.1.9 struct hwprofile

Definition at line 118 of file structures.h.

Collaboration diagram for hwprofile:



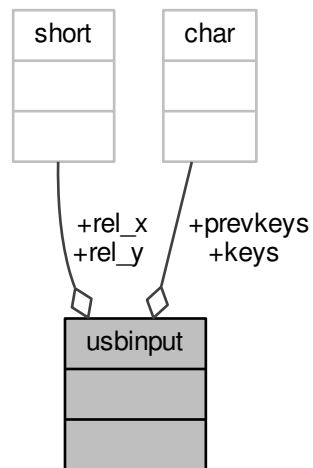
Data Fields

dpiset	<code>dpi[3]</code>	
usbid	<code>id[3+1]</code>	
lighting	<code>light[3]</code>	
ushort	<code>name[3+1][16]</code>	

9.40.1.10 struct usbinput

Definition at line 129 of file structures.h.

Collaboration diagram for usbinput:

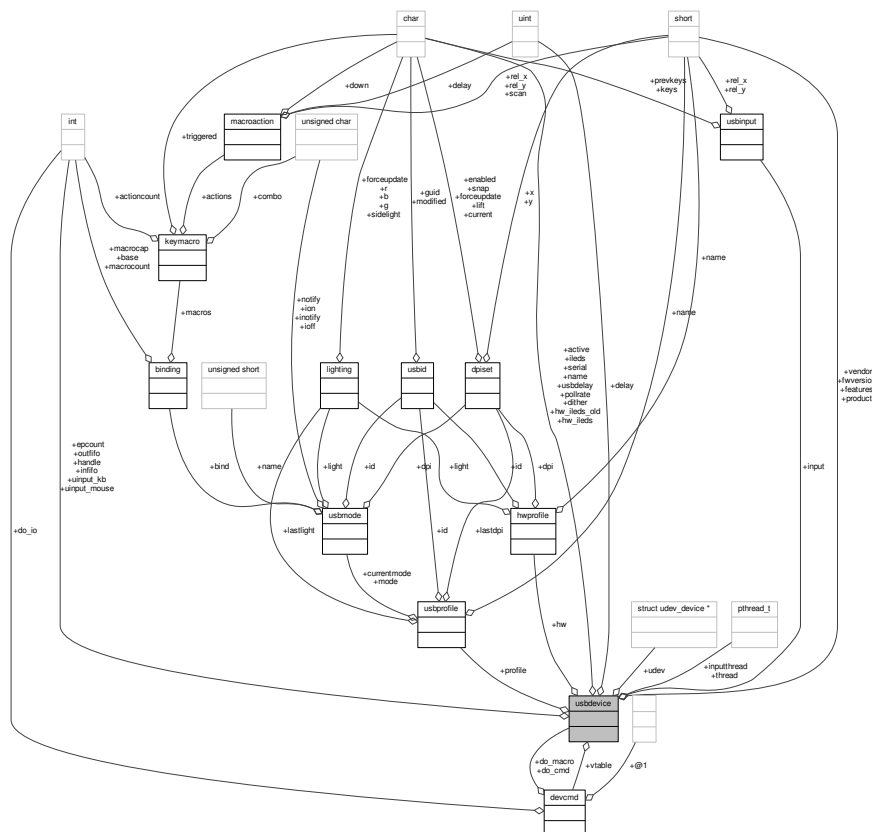


Data Fields

uchar	keys[((((152+3+12)+25)+7)/8)]	
uchar	prevkeys[((((152+3+12)+25)+7)/8)]	
short	rel_x	
short	rel_y	

9.40.1.11 struct usbdevice

Definition at line 178 of file structures.h.



Data Fields

char	active	
uint	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[40+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	

9.40.2 Macro Definition Documentation

9.40.2.1 `#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()`.

9.40.2.2 `#define DPI_COUNT 6`

Definition at line 54 of file structures.h.

Referenced by `cmd_dpi()`, `cmd_dpisel()`, `loaddpi()`, `printdpi()`, `savedpi()`, and `updatedpi()`.

9.40.2.3 `#define FEAT_ADJRATE 0x008`

Definition at line 139 of file structures.h.

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

9.40.2.4 `#define FEAT_ANSI 0x200`

Definition at line 146 of file structures.h.

Referenced by `readcmd()`.

9.40.2.5 `#define FEAT_BIND 0x010`

Definition at line 140 of file structures.h.

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

9.40.2.6 `#define FEAT_COMMON (FEAT_BIND | FEAT_NOTIFY | FEAT_FWVERSION | FEAT_MOUSEACCEL | FEAT_HWLOAD)`

Definition at line 151 of file structures.h.

9.40.2.7 `#define FEAT_FWUPDATE 0x080`

Definition at line 143 of file structures.h.

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

9.40.2.8 #define FEAT_FWVERSION 0x040

Definition at line 142 of file structures.h.

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

9.40.2.9 #define FEAT_HWLOAD 0x100

Definition at line 144 of file structures.h.

Referenced by `_start_dev()`.

9.40.2.10 #define FEAT_ISO 0x400

Definition at line 147 of file structures.h.

Referenced by `readcmd()`.

9.40.2.11 #define FEAT_LMASK (FEAT_ANSI | FEAT_ISO)

Definition at line 154 of file structures.h.

Referenced by `readcmd()`.

9.40.2.12 #define FEAT_MONOCHROME 0x002

Definition at line 137 of file structures.h.

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

9.40.2.13 #define FEAT_MOUSEACCEL 0x800

Definition at line 148 of file structures.h.

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

9.40.2.14 #define FEAT_NOTIFY 0x020

Definition at line 141 of file structures.h.

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

9.40.2.15 #define FEAT_POLLRATE 0x004

Definition at line 138 of file structures.h.

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

9.40.2.16 #define FEAT_RGB 0x001

Definition at line 136 of file structures.h.

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

9.40.2.17 #define FEAT_STD_NRGB (FEAT_COMMON)

Definition at line 153 of file structures.h.

Referenced by `_setupusb()`.

9.40.2.18 #define FEAT_STD_RGB (FEAT_COMMON | FEAT_RGB | FEAT_POLLRATE | FEAT_FWUPDATE)

Definition at line 152 of file structures.h.

Referenced by `_setupusb()`.

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

Definition at line 158 of file structures.h.

9.40.2.20 #define HAS_FEATURES(kb, feat) (((kb)->features & (feat)) == (feat))

Definition at line 157 of file structures.h.

Referenced by `_mkdevpath()`, `_start_dev()`, `cmd_fwupdate()`, `readcmd()`, `revertusb()`, and `usbunclaim()`.

9.40.2.21 #define HWMODE_K70 1

Definition at line 115 of file structures.h.

Referenced by `cmd_hwload_kb()`, and `cmd_hwsave_kb()`.

9.40.2.22 #define HWMODE_K95 3

Definition at line 116 of file structures.h.

Referenced by `cmd_hwload_kb()`, and `cmd_hwsave_kb()`.

9.40.2.23 #define HWMODE_MAX 3

Definition at line 117 of file structures.h.

9.40.2.24 #define I_CAPS 2

Definition at line 20 of file structures.h.

Referenced by `_cmd_get()`, `iselect()`, `nprintind()`, and `updateindicators_kb()`.

9.40.2.25 #define I_NUM 1

Definition at line 19 of file structures.h.

Referenced by `_cmd_get()`, `iselect()`, `nprintind()`, and `updateindicators_kb()`.

9.40.2.26 #define I_SCROLL 4

Definition at line 21 of file structures.h.

Referenced by `_cmd_get()`, `iselect()`, `nprintind()`, and `updateindicators_kb()`.

9.40.2.27 #define IFACE_MAX 4

Definition at line 177 of file structures.h.

9.40.2.28 #define KB_NAME_LEN 40

Definition at line 174 of file structures.h.

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

9.40.2.29 #define LIFT_MAX 5

Definition at line 56 of file structures.h.

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

9.40.2.30 #define LIFT_MIN 1

Definition at line 55 of file structures.h.

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

9.40.2.31 #define MACRO_MAX 1024

Definition at line 51 of file structures.h.

Referenced by `_cmd_macro()`.

9.40.2.32 #define MD_NAME_LEN 16

Definition at line 82 of file structures.h.

Referenced by `cmd_hwsave_kb()`, `cmd_hwsave_mouse()`, `cmd_name()`, `gethwmodename()`, `gethwprofilename()`, `getmodename()`, `hwloadmode()`, `hwtonative()`, and `nativetohw()`.

9.40.2.33 #define MODE_COUNT 6

Definition at line 100 of file structures.h.

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

9.40.2.34 #define MSG_SIZE 64

Definition at line 176 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()`.

9.40.2.35 #define NEEDS_FW_UPDATE(kb) ((kb)->fwversion == 0 && HAS_FEATURES((kb), FEAT_FWUPDATE | FEAT_FWVERSION))

Definition at line 161 of file structures.h.

Referenced by `_start_dev()`, `readcmd()`, `revertusb()`, `setactive_kb()`, and `setactive_mouse()`.

9.40.2.36 `#define OUTFIFO_MAX 10`

Definition at line 24 of file structures.h.

Referenced by `_mknotifynode()`, `_rmnotifynode()`, `inputupdate_keys()`, `nprintf()`, `readcmd()`, `rmdevpath()`, and `updateindicators_kb()`.

9.40.2.37 `#define PR_NAME_LEN 16`

Definition at line 99 of file structures.h.

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

9.40.2.38 `#define SCROLL_ACCELERATED 0`

Definition at line 164 of file structures.h.

Referenced by `readcmd()`.

9.40.2.39 `#define SCROLL_MAX 10`

Definition at line 166 of file structures.h.

Referenced by `readcmd()`.

9.40.2.40 `#define SCROLL_MIN 1`

Definition at line 165 of file structures.h.

Referenced by `readcmd()`.

9.40.2.41 `#define SERIAL_LEN 34`

Definition at line 175 of file structures.h.

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

9.40.2.42 `#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()`.

9.40.3 Variable Documentation

9.40.3.1 `const union devcmd vtable_keyboard`

Definition at line 29 of file device_vtable.c.

Referenced by `get_vtable()`.

9.40.3.2 `const union devcmd vtable_keyboard_nonrgb`

Definition at line 76 of file device_vtable.c.

Referenced by `get_vtable()`.

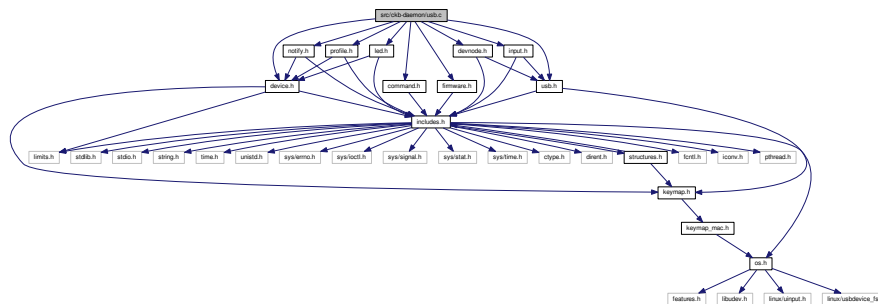
9.40.3.3 const union devcmd vtable_mouse

Definition at line 123 of file device_vtable.c.

Referenced by `get_vtable()`.

9.41 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 [hload_mode](#)
hload_mode is defined in [device.c](#)

9.41.1 Function Documentation

9.41.1.1 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 updatetrgb() with force=true.

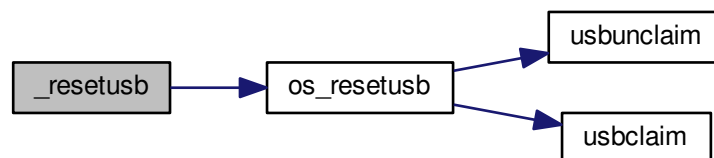
Definition at line 426 of file `usb.c`.

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->updatetrgb(kb, 1) != 0)
437         return -1;
438     return 0;
439 }
```

Here is the call graph for this function:



9.41.1.2 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:



9.41.1.3 `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. < Synchronization between macro and color information

Definition at line 601 of file usb.c.

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

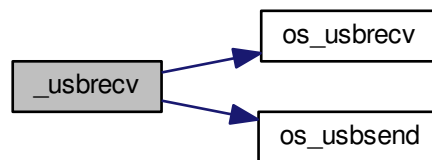
```

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

```

```
632 }
```

Here is the call graph for this function:



9.41.1.4 `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 (...). < Synchronization between macro and color information

Definition at line 532 of file usb.c.

References `DELAY_LONG`, `DELAY_SHORT`, `hwload_mode`, `mmutex`, `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             pthread_mutex_lock(&mutex(kb));
538             DELAY_SHORT(kb);
539             int res = os_usbsend(kb, messages + i * MSG_SIZE, 0, file, line);
540             pthread_mutex_unlock(&mutex(kb));
541             if(res == 0)
542                 return 0;
543             else if(res != -1){
544                 total_sent += res;
545                 break;
546             }
547             // Stop immediately if the program is shutting down or hardware load is set to tryonce
548             if(reset_stop || hwload_mode != 2)
549                 return 0;
550             // Retry as long as the result is temporary failure
551             DELAY_LONG(kb);
552         }
553     }
554     return total_sent;
555 }
```

Here is the call graph for this function:



9.41.1.5 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 677 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`.

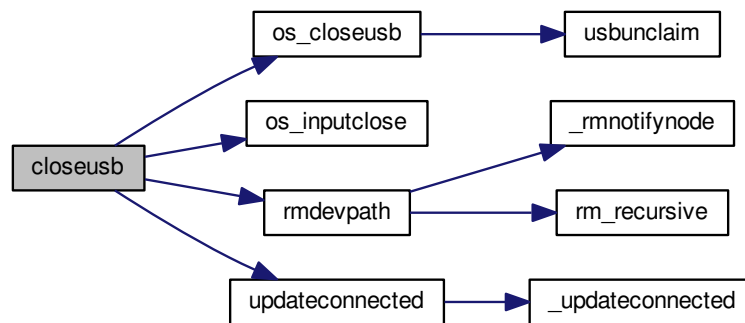
Referenced by `_setupusb()`, `devmain()`, `quitWithLock()`, and `usb_rm_device()`.

```

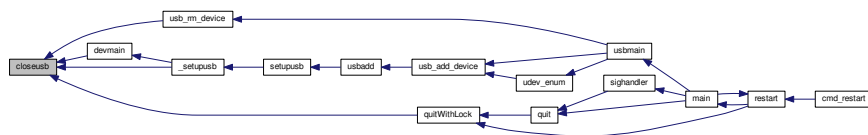
677         {
678     pthread_mutex_lock(imutex(kb));
679     if(kb->handle){
680         int index = INDEX_OF(kb, keyboard);
681         ckb_info("Disconnecting %s%d\n", devpath, index);
682         os_inputclose(kb);
683         updateconnected();
684         // Close USB device
685         os_closeusb(kb);
686     } else
687         updateconnected();
688     rmdevpath(kb);
689
690     // Wait for thread to close
691     pthread_mutex_unlock(imutex(kb));
692     pthread_mutex_unlock(dmutex(kb));
693     pthread_join(kb->thread, 0);
694     pthread_mutex_lock(dmutex(kb));
695
696     // Delete the profile and the control path
697     if(!kb->vtable)
698         return 0;
699     kb->vtable->freeprofile(kb);
700     memset(kb, 0, sizeof(usbdevice));
701     return 0;
702 }

```


Here is the call graph for this function:



Here is the caller graph for this function:



9.41.1.6 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;

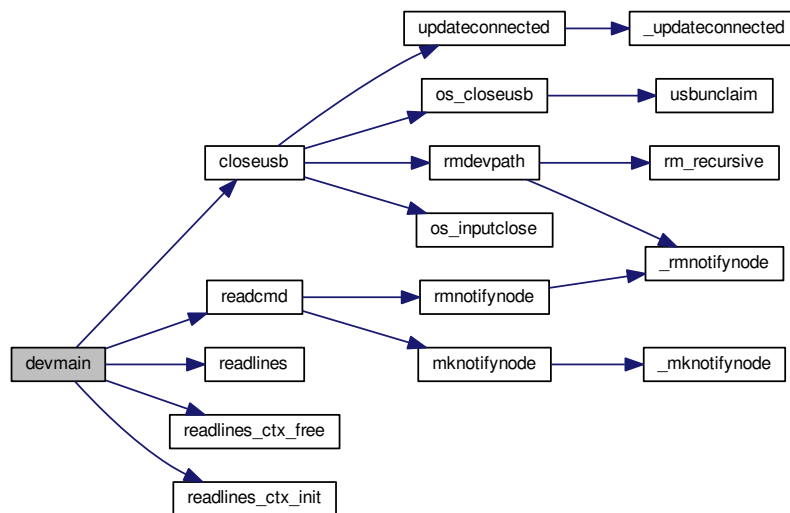
```

```

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:



9.41.1.7 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:



9.41.1.8 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_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().

```

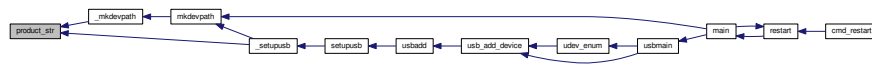
70                                     {
71     if (product == P_K95 || product == P_K95_NRGB || product ==
        P_K95_PLATINUM)
72         return "k95";
73     if (product == P_K70 || product == P_K70_NRGB || product ==
        P_K70_LUX || product == P_K70_LUX_NRGB || product ==
        P_K70_RFIRE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



9.41.1.9 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:



9.41.1.10 void setupusb (usbdevice * kb)

setupusb starts a thread with kb as parameter and `_setupusb()` as entrypoint.

Set up a USB device after its handle is open. Spawns a new thread `_setupusb()` with standard parameter kb. dmutex must be locked prior to calling this function. The function will unlock it when finished. In kb->thread the thread id is mentioned, because `closeusb()` needs this info for joining that thread again.

Definition at line 386 of file usb.c.

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

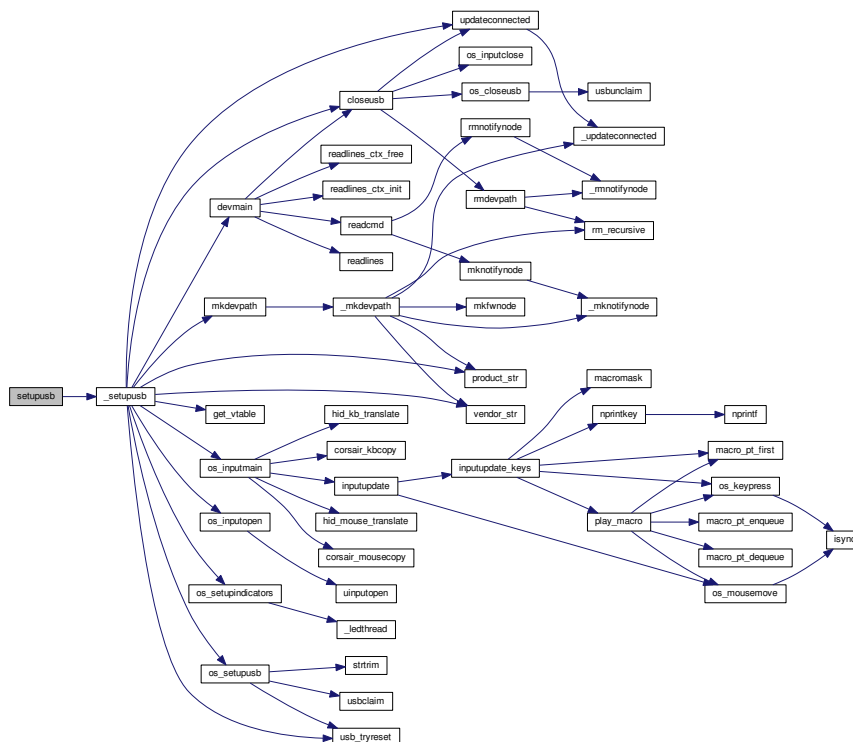
Referenced by `usbadd()`.

```

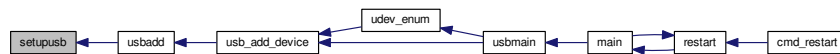
386 {
387     pthread_mutex_lock(&imutex(kb));
388     if(pthread_create(&kb->thread, 0, _setupusb, kb))
389         ckb_err("Failed to create USB thread\n");
390 }

```

Here is the call graph for this function:



Here is the caller graph for this function:



9.41.1.11 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.

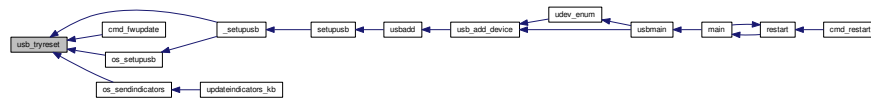
References [ckb_err](#), [ckb_info](#), [reset_stop](#), and [resetusb](#).

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

```

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:



9.41.1.12 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*.

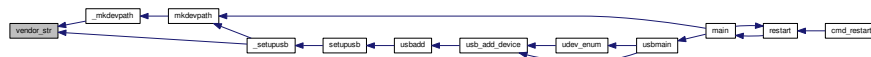
Referenced by *_mkdevpath()*, and *_setupusb()*.

```

43      {
44      if (vendor == V_CORSAIR)
45          return "corsair";
46      return "";
47  }

```

Here is the caller graph for this function:



9.41.2 Variable Documentation

9.41.2.1 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()*.

9.41.2.2 int hwload_mode

hwload_mode is defined in [device.c](#)

Definition at line 7 of file device.c.

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


```

• #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 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))
• #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_rcv, 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\(\)](#)

9.42.1 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](#).

9.42.2 Macro Definition Documentation

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

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

Referenced by [_resetusb\(\)](#), [_setupusb\(\)](#), [_usbrecv\(\)](#), [_usbseend\(\)](#), [cmd_hwload_kb\(\)](#), [cmd_hwload_mouse\(\)](#), [cmd_hwsave_kb\(\)](#), and [cmd_hwsave_mouse\(\)](#).

9.42.2.2 `#define DELAY_MEDIUM(kb) usleep((int)(kb)->usbdelay * 10000)`

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

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

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

Definition at line 147 of file [usb.h](#).

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

9.42.2.4 `#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 136 of file [usb.h](#).

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

9.42.2.5 `#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\(\)](#).

9.42.2.6 `#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](#).

```
9.42.2.7 #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().

```
9.42.2.8 #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().

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

Definition at line 127 of file usb.h.

Referenced by _setupusb().

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

Definition at line 133 of file usb.h.

```
9.42.2.11 #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 139 of file usb.h.

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

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

Definition at line 142 of file usb.h.

Referenced by readcmd().

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

Definition at line 122 of file usb.h.

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

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

Definition at line 130 of file usb.h.

```
9.42.2.15 #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))
```

Definition at line 93 of file usb.h.

Referenced by has_key(), loadrgb_mouse(), and savergb_mouse().

9.42.2.16 `#define IS_SCIMITAR(kb) ((kb)->vendor == V_CORSAIR && ((kb)->product == P_SCIMITAR || (kb)->product == P_SCIMITAR_PRO))`

Definition at line 99 of file usb.h.

Referenced by `has_key()`, `loadrgb_mouse()`, and `savergb_mouse()`.

9.42.2.17 `#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()`.

9.42.2.18 `#define NK95_HWOFF 0x020030`

See Also

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

Definition at line 299 of file usb.h.

Referenced by `start_kb_nrgb()`.

9.42.2.19 `#define NK95_HWON 0x020001`

Definition at line 302 of file usb.h.

Referenced by `revertusb()`.

9.42.2.20 `#define NK95_M1 0x140001`

Definition at line 305 of file usb.h.

Referenced by `setmodeindex_nrgb()`.

9.42.2.21 `#define NK95_M2 0x140002`

Definition at line 308 of file usb.h.

Referenced by `setmodeindex_nrgb()`.

9.42.2.22 `#define NK95_M3 0x140003`

Definition at line 311 of file usb.h.

Referenced by `setmodeindex_nrgb()`.

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

Definition at line 294 of file usb.h.

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

9.42.2.24 #define P_K65 0x1b17

Definition at line 41 of file usb.h.

Referenced by product_str().

9.42.2.25 #define P_K65_LUX 0x1b37

Definition at line 45 of file usb.h.

Referenced by product_str().

9.42.2.26 #define P_K65_LUX_STR "1b37"

Definition at line 46 of file usb.h.

9.42.2.27 #define P_K65_NRGB 0x1b07

Definition at line 43 of file usb.h.

Referenced by product_str().

9.42.2.28 #define P_K65_NRGB_STR "1b07"

Definition at line 44 of file usb.h.

9.42.2.29 #define P_K65_RFIRE 0x1b39

Definition at line 47 of file usb.h.

Referenced by product_str().

9.42.2.30 #define P_K65_RFIRE_STR "1b39"

Definition at line 48 of file usb.h.

9.42.2.31 #define P_K65_STR "1b17"

Definition at line 42 of file usb.h.

9.42.2.32 #define P_K70 0x1b13

Definition at line 51 of file usb.h.

Referenced by product_str().

9.42.2.33 #define P_K70_LUX 0x1b33

Definition at line 55 of file usb.h.

Referenced by loadrgb_kb(), and product_str().

9.42.2.34 #define P_K70_LUX_NRGB 0x1b36

Definition at line 57 of file usb.h.

Referenced by loadrgb_kb(), and product_str().

9.42.2.35 #define P_K70_LUX_NRGB_STR "1b36"

Definition at line 58 of file usb.h.

9.42.2.36 #define P_K70_LUX_STR "1b33"

Definition at line 56 of file usb.h.

9.42.2.37 #define P_K70_NRGB 0x1b09

Definition at line 53 of file usb.h.

Referenced by product_str().

9.42.2.38 #define P_K70_NRGB_STR "1b09"

Definition at line 54 of file usb.h.

9.42.2.39 #define P_K70_RFIRE 0x1b38

Definition at line 59 of file usb.h.

Referenced by product_str().

9.42.2.40 #define P_K70_RFIRE_NRGB 0x1b3a

Definition at line 61 of file usb.h.

Referenced by product_str().

9.42.2.41 #define P_K70_RFIRE_NRGB_STR "1b3a"

Definition at line 62 of file usb.h.

9.42.2.42 #define P_K70_RFIRE_STR "1b38"

Definition at line 60 of file usb.h.

9.42.2.43 #define P_K70_STR "1b13"

Definition at line 52 of file usb.h.

9.42.2.44 #define P_K95 0x1b11

Definition at line 65 of file usb.h.

Referenced by product_str().

9.42.2.45 `#define P_K95_NRGB 0x1b08`

Definition at line 67 of file usb.h.

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

9.42.2.46 `#define P_K95_NRGB_STR "1b08"`

Definition at line 68 of file usb.h.

9.42.2.47 `#define P_K95_PLATINUM 0x1b2d`

Definition at line 69 of file usb.h.

Referenced by `product_str()`.

9.42.2.48 `#define P_K95_PLATINUM_STR "1b2d"`

Definition at line 70 of file usb.h.

9.42.2.49 `#define P_K95_STR "1b11"`

Definition at line 66 of file usb.h.

9.42.2.50 `#define P_M65 0x1b12`

Definition at line 79 of file usb.h.

Referenced by `product_str()`.

9.42.2.51 `#define P_M65_PRO 0x1b2e`

Definition at line 81 of file usb.h.

Referenced by `product_str()`.

9.42.2.52 `#define P_M65_PRO_STR "1b2e"`

Definition at line 82 of file usb.h.

9.42.2.53 `#define P_M65_STR "1b12"`

Definition at line 80 of file usb.h.

9.42.2.54 `#define P_SABRE_L 0x1b19 /* laser */`

Definition at line 87 of file usb.h.

Referenced by `product_str()`.

9.42.2.55 `#define P_SABRE_L_STR "1b19"`

Definition at line 88 of file usb.h.

9.42.2.56 `#define P_SABRE_N 0x1b2f /* new? */`

Definition at line 89 of file usb.h.

Referenced by `product_str()`.

9.42.2.57 `#define P_SABRE_N_STR "1b2f"`

Definition at line 90 of file usb.h.

9.42.2.58 `#define P_SABRE_O 0x1b14 /* optical */`

Definition at line 85 of file usb.h.

Referenced by `product_str()`.

9.42.2.59 `#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()`.

9.42.2.60 `#define P_SABRE_O2_STR "1b32"`

Definition at line 92 of file usb.h.

9.42.2.61 `#define P_SABRE_O_STR "1b14"`

Definition at line 86 of file usb.h.

9.42.2.62 `#define P_SCIMITAR 0x1b1e`

Definition at line 95 of file usb.h.

Referenced by `product_str()`.

9.42.2.63 `#define P_SCIMITAR_PRO 0x1b3e`

Definition at line 97 of file usb.h.

Referenced by `product_str()`.

9.42.2.64 `#define P_SCIMITAR_PRO_STR "1b3e"`

Definition at line 98 of file usb.h.

9.42.2.65 `#define P_SCIMITAR_STR "1b1e"`

Definition at line 96 of file usb.h.

9.42.2.66 `#define P_STRAFE 0x1b20`

Definition at line 73 of file usb.h.

Referenced by `product_str()`.

9.42.2.67 `#define P_STRAFE_NRGB 0x1b15`

Definition at line 75 of file usb.h.

Referenced by `product_str()`.

9.42.2.68 `#define P_STRAFE_NRGB_STR "1b15"`

Definition at line 76 of file usb.h.

9.42.2.69 `#define P_STRAFE_STR "1b20"`

Definition at line 74 of file usb.h.

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

Definition at line 212 of file usb.h.

Referenced by `usb_tryreset()`.

9.42.2.71 `#define USB_DELAY_DEFAULT 5`

Definition at line 158 of file usb.h.

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

9.42.2.72 `#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 254 of file usb.h.

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

9.42.2.73 `#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>/N/</i>	count how many MSG_SIZE buffers is the logical message long?
------------	--

Definition at line 237 of file usb.h.

Referenced by `cmd_hwsave_kb()`, `cmd_hwsave_mouse()`, `cmd_pollrate()`, `fwupdate()`, `loadrgb_kb()`, `savedpi()`, `savergb_kb()`, `savergb_mouse()`, `setactive_kb()`, `setactive_mouse()`, `updatedpi()`, `updatergb_kb()`, and `updatergb_mouse()`.

9.42.2.74 `#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()`.

9.42.2.75 `#define V_CORSAIR_STR "1b1c"`

Definition at line 39 of file usb.h.

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

9.42.3 Function Documentation

9.42.3.1 `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_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 192 of file usb_linux.c.

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

```

192                                     {
193     if(kb->product != P_K95_NRGB)
194         return 0;
195     struct usbdevfs_ctrltransfer transfer = { 0x40, bRequest, wValue, 0, 0, 5000, 0 };
196     int res = ioctl(kb->handle - 1, USBDEVFS_CONTROL, &transfer);
197     if(res <= 0){
198         ckb_err_fn("%s\n", file, line, res ? strerror(errno) : "No data written");
199         return 1;
200     }
201     return 0;
202 }
```

9.42.3.2 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`.

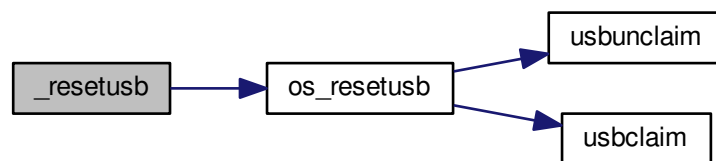
Definition at line 426 of file `usb.c`.

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:



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

Parameters

<i>kb</i>	THE <code>usbdevice*</code>
<i>IN</i>]	<code>out_msg</code> What information does the caller want from the device?
<i>OUT</i>]	<code>in_msg</code> Here comes the answer; The names represent the usb view, not the view of this function! So <i>IN</i> put from usb is <i>OUT</i> put of this function.
<i>IN</i>]	<code>file</code> for debugging
<i>IN</i>]	<code>line</code> for debugging
<i>IN</i>]	<code>reset_stop</code> 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_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_usbsend()`.

`os_usbsend()` 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_usbsend()` 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_usbsend()` 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. < Synchronization between macro and color information

Definition at line 601 of file usb.c.

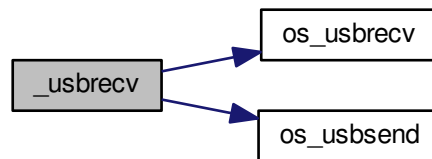
References `ckb_err_fn`, `DELAY_LONG`, `DELAY_MEDIUM`, `DELAY_SHORT`, `hwload_mode`, `mmutex`, `os_usbsend()`, `os_usbsend()`, and `reset_stop`.

```

601                                     {
602     // Try a maximum of 5 times
603     for (int try = 0; try < 5; try++) {
604         // Send the output message
605         pthread_mutex_lock(&mmutex(kb));
606         DELAY_SHORT(kb);
607         int res = os_usbsend(kb, out_msg, 1, file, line);
608         pthread_mutex_unlock(&mmutex(kb));
609         if (res == 0)
610             return 0;
611         else if (res == -1) {
612             // Retry on temporary failure
613             if (reset_stop)
614                 return 0;
615             DELAY_LONG(kb);
616             continue;
617         }
618         // Wait for the response
619         DELAY_MEDIUM(kb);
620         res = os_usbsend(kb, in_msg, file, line);
621         if (res == 0)
622             return 0;
623         else if (res != -1)
624             return res;
625         if (reset_stop || hwload_mode != 2)
626             return 0;
627         DELAY_LONG(kb);
628     }
629     // Give up
630     ckb_err_fn("Too many send/recv failures. Dropping.\n", file, line);
631     return 0;
632 }

```


Here is the call graph for this function:



9.42.3.4 `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
<i>in</i>	<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 **hload_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) `hload_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 (...). < Synchronization between macro and color information

Definition at line 532 of file usb.c.

References `DELAY_LONG`, `DELAY_SHORT`, `hwload_mode`, `mmutex`, `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             pthread_mutex_lock(&mutex(kb));
538             DELAY_SHORT(kb);
539             int res = os_usbsend(kb, messages + i * MSG_SIZE, 0, file, line);
540             pthread_mutex_unlock(&mutex(kb));
541             if(res == 0)
542                 return 0;
543             else if(res != -1){
544                 total_sent += res;
545                 break;
546             }
547             // Stop immediately if the program is shutting down or hardware load is set to tryonce
548             if(reset_stop || hwload_mode != 2)
549                 return 0;
550             // Retry as long as the result is temporary failure
551             DELAY_LONG(kb);
552         }
553     }
554     return total_sent;
555 }
```

Here is the call graph for this function:



9.42.3.5 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 $\neq 0$, 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 677 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()`.

```

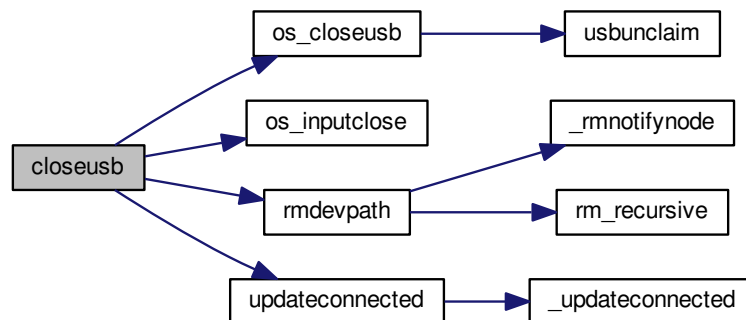
677     {
678     pthread_mutex_lock(&imutex(kb));
679     if(kb->handle) {
680         int index = INDEX_OF(kb, keyboard);
681         ckb_info("Disconnecting %s%d\n", devpath, index);
682         os_inputclose(kb);
683         updateconnected();
684         // Close USB device
685         os_closeusb(kb);
686     } else
687         updateconnected();
688     rmdevpath(kb);
689 
```

```

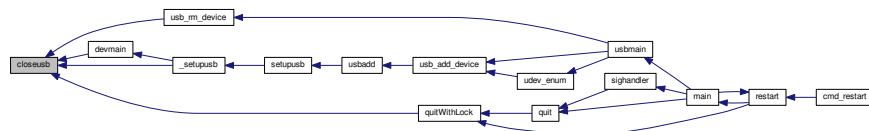
690 // Wait for thread to close
691 pthread_mutex_unlock(imutex(kb));
692 pthread_mutex_unlock(dmutex(kb));
693 pthread_join(kb->thread, 0);
694 pthread_mutex_lock(dmutex(kb));
695
696 // Delete the profile and the control path
697 if(!kb->vtable)
698     return 0;
699 kb->vtable->freeprofile(kb);
700 memset(kb, 0, sizeof(usbdevice));
701 return 0;
702 }

```

Here is the call graph for this function:



Here is the caller graph for this function:



9.42.3.6 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 unreferenced 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 439 of file [usb_linux.c](#).

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

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

```

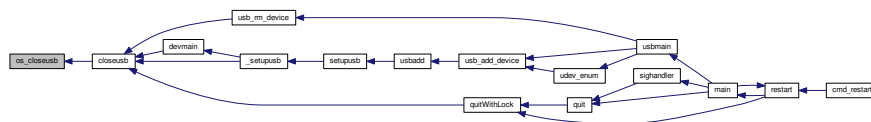
439                                     {
440     if (kb->handle) {
441         usbunclaim(kb, 0);
442         close(kb->handle - 1);
443     }
444     if (kb->udev)
445         udev_device_unref(kb->udev);
446     kb->handle = 0;
447     kb->udev = 0;
448     kbsyspath[INDEX_OF(kb, keyboard)][0] = 0;
449 }

```

Here is the call graph for this function:



Here is the caller graph for this function:



9.42.3.7 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 242 of file `usb_linux.c`.

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

```

242                                     {
243     usbdevice* kb = context;
244     int fd = kb->handle - 1;
245     short vendor = kb->vendor, product = kb->product;
246     int index = INDEX_OF(kb, keyboard);
247     ckb_info("Starting input thread for %s%d\n", devpath, index);
248
249     int urbcount = IS_RGB(vendor, product) ? (kb->epcount - 1) : kb->
250     epcount;
251     if (urbcount == 0) {
252         ckb_err("urbcount = 0, so there is nothing to claim in os_inputmain()\n");
253         return 0;
254     }
255
256     struct usbdevfs_urb urbs[urbcount + 1];
257     memset(urbs, 0, sizeof(urbs));
258
259     urbs[0].buffer_length = 8;
260     if(urbcount > 1 && IS_RGB(vendor, product)) {
261         if(IS_MOUSE(vendor, product))
262             urbs[1].buffer_length = 10;
263         else
264             urbs[1].buffer_length = 21;
265         urbs[2].buffer_length = MSG_SIZE;
266         if(urbcount != 3)
267             urbs[urbcount - 1].buffer_length = MSG_SIZE;
268     } else {
269         urbs[1].buffer_length = 4;
270         urbs[2].buffer_length = 15;
271     }
272
273     for(int i = 0; i < urbcount; i++){
274         urbs[i].type = USBDEVFS_URB_TYPE_INTERRUPT;
275         urbs[i].endpoint = 0x80 | (i + 1);
276         urbs[i].buffer = malloc(urbs[i].buffer_length);
277         ioctl(fd, USBDEVFS_SUBMITURB, urbs + i);
278     }
279
280     while (1) {
281         struct usbdevfs_urb* urb = 0;
282
283         if (ioctl(fd, USBDEVFS_REAPURB, &urb)) {
284             if (errno == ENODEV || errno == ENOENT || errno == ESHUTDOWN)
285                 // Stop the thread if the handle closes
286                 break;
287             else if(errno == EPIPE && urb){
288                 ioctl(fd, USBDEVFS_CLEAR_HALT, &urb->endpoint);
289                 // Re-submit the URB
290                 if(urb)
291                     ioctl(fd, USBDEVFS_SUBMITURB, urb);
292                 urb = 0;
293             }
294             continue;
295         }
296
297         if (urb) {
298             pthread_mutex_lock(&mutex(kb));
299             if(IS_MOUSE(vendor, product)){
300                 switch(urb->actual_length){
301                     case 8:
302                     case 10:
303                     case 11:
304                         // HID mouse input
305                         hid_mouse_translate(kb->input.keys, &kb->
306                         input.rel_x, &kb->input.rel_y, -(urb->endpoint & 0xF), urb->actual_length, urb->buffer)
307                         ;
308                         break;
309                     case MSG_SIZE:
310                         // Corsair mouse input
311                         corsair_mousecopy(kb->input.keys, -(urb->endpoint & 0xF), urb
312                         ->buffer);
313                         break;
314                 }
315             } else if(IS_RGB(vendor, product)){
316                 switch(urb->actual_length){
317                     case 8:
318                         // RGB EP 1: 6KRO (BIOS mode) input
319                         hid_kb_translate(kb->input.keys, -1, urb->actual_length, urb->
320                         buffer);
321                         break;
322                     case 21:
323                     case 5:

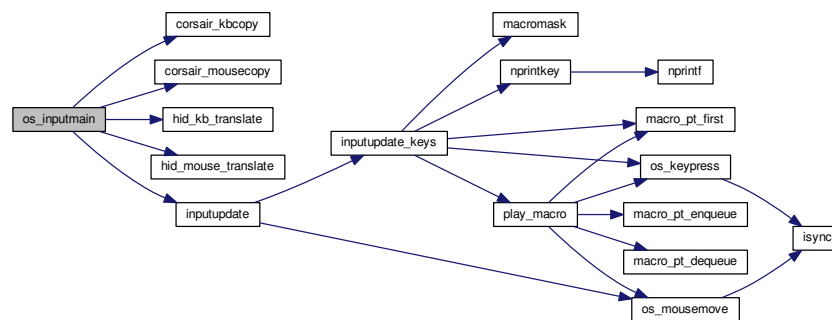
```

```

358         // RGB EP 2: NKRO (non-BIOS) input. Accept only if keyboard is inactive
359         if(!kb->active)
360             hid_kb_translate(kb->input.keys, -2, urb->actual_length,
urb->buffer);
361         break;
362     case MSG_SIZE:
363         // RGB EP 3: Corsair input
364         corsair_kbcopy(kb->input.keys, -(urb->endpoint & 0xF), urb->
buffer);
365         break;
366     }
367     } else {
368         // Non-RGB input
369         hid_kb_translate(kb->input.keys, urb->endpoint & 0xF, urb->
actual_length, urb->buffer);
370     }
371     inputupdate(kb);
372     pthread_mutex_unlock(&mutex(kb));
373
374     ioctl(fd, USBDEVFS_SUBMITURB, urb);
375     urb = 0;
376 }
377 }
378 }
379 }
380 }
381
382 ckb_info("Stopping input thread for %s%d\n", devpath, index);
383 for(int i = 0; i < urbcount; i++){
384     ioctl(fd, USBDEVFS_DISCARDURB, urbs + i);
385     free(urbs[i].buffer);
386 }
387 return 0;
388 }

```

Here is the call graph for this function:



Here is the caller graph for this function:



9.42.3.8 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 501 of file usb_linux.c.

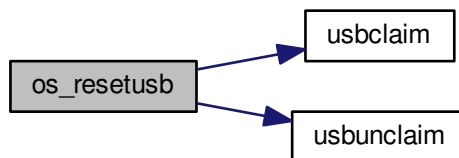
References usbdevice::handle, TEST_RESET, usbclaim(), and usbunclaim().

Referenced by _resetusb().

```

501                                     {
502     TEST_RESET(usbunclaim(kb, 1));
503     TEST_RESET(ioctl(kb->handle - 1, USBDEVFS_RESET));
504     TEST_RESET(usbclaim(kb));
505     // Success!
506     return 0;
507 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



9.42.3.9 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 217 of file usb_linux.c.

References ckb_err, usbdevice::handle, usbdevice::ileds, and usb_tryreset().

Referenced by updateindicators_kb().

```

217     {
218         static int countForReset = 0;
219         struct usbdevfs_ctrltransfer transfer = { 0x21, 0x09, 0x0200, 0x00, 1, 500, &kb->
ileds };
220         int res = ioctl(kb->handle - 1, USBDEVFS_CONTROL, &transfer);
221         if(res <= 0) {
222             ckb_err("%s\n", res ? strerror(errno) : "No data written");
223             if (usb_tryreset(kb) == 0 && countForReset++ < 3) {
224                 os_sendindicators(kb);
225             }
226         }
227     }

```

Here is the call graph for this function:



Here is the caller graph for this function:



9.42.3.10 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

< Try to reset the device and recall the function

< Don't do this endless in recursion

< [os_setupusb\(\)](#) has a return value (used as boolean)

Definition at line 539 of file usb_linux.c.

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

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

```

539     {
540     struct udev_device* dev = kb->udev;
541     const char* name = udev_device_get_sysattr_value(dev, "product");
542     if(name)
543         strncpy(kb->name, name, KB_NAME_LEN);
544     strtrim(kb->name);
545     const char* serial = udev_device_get_sysattr_value(dev, "serial");
546     if(serial)
547         strncpy(kb->serial, serial, SERIAL_LEN);
548     strtrim(kb->serial);
549     const char* firmware = udev_device_get_sysattr_value(dev, "bcdDevice");
550     if(firmware)
551         sscanf(firmware, "%hx", &kb->fwversion);
552     else
553         kb->fwversion = 0;
554     int index = INDEX_OF(kb, keyboard);
555     ckb_info("Connecting %s at %s%d\n", kb->name, devpath, index);
556
557     const char* ep_str = udev_device_get_sysattr_value(dev, "bNumInterfaces");
558     #ifdef DEBUG
559     ckb_info("Connecting %s at %s%d\n", kb->name, devpath, index);
560     ckb_info("claiming interfaces. name=%s, serial=%s, firmware=%s; Got >>%s<< as ep_str\n", name,
561         serial, firmware, ep_str);
562     #endif //DEBUG
563     kb->epcount = 0;
564     if(ep_str)
565         sscanf(ep_str, "%d", &kb->epcount);
566     if(kb->epcount < 2){
567         // IF we have an RGB KB with 0 or 1 endpoints, it will be in BIOS mode.
568         ckb_err("Unable to read endpoint count from udev, assuming %d and reading >>%s<< or device
569         is in BIOS mode\n", kb->epcount, ep_str);
570         if (usb_tryreset(kb) == 0) {
571             static int retryCount = 0;

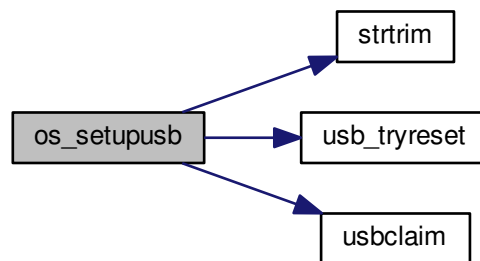
```

```

581         if (retryCount++ < 5) {
582             return os_setupusb(kb);
583         }
584     }
585     return -1;
586     // ToDo are there special versions we have to detect? If there are, that was the old code to handle
it:
587     // This shouldn't happen, but if it does, assume EP count based on ckb_warn what the device is
supposed to have
588     // kb->epcount = (HAS_FEATURES(kb, FEAT_RGB) ? 4 : 3);
589     // ckb_warn("Unable to read endpoint count from udev, assuming %d and reading >>%s<<...\n",
kb->epcount, ep_str);
590 }
591 if(usbclaim(kb)) {
592     ckb_err("Failed to claim interfaces: %s\n", strerror(errno));
593     return -1;
594 }
595 return 0;
596 }

```

Here is the call graph for this function:



Here is the caller graph for this function:



9.42.3.11 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_usbrecv`. 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 132 of file `usb_linux.c`.

References `ckb_err_fn`, `ckb_warn_fn`, `usbdevice::epcount`, `usbdevice::handle`, and `MSG_SIZE`.

Referenced by `_usbrecv()`.

```

132                                     {
133     int res;
134     // This is what CUE does, but it doesn't seem to work on linux.
135     /*if(kb->fwversion >= 0x130){
136         struct usbdevfs_bulktransfer transfer;
137         memset(&transfer, 0, sizeof(transfer));
138         transfer.ep = 0x84;
139         transfer.len = MSG_SIZE;
140         transfer.timeout = 5000;
141         transfer.data = in_msg;
142         res = ioctl(kb->handle - 1, USBDEVFS_BULK, &transfer);
143     } else {*/
144         struct usbdevfs_ctrltransfer transfer = { 0xa1, 0x01, 0x0300, kb->
145 epcount - 1, MSG_SIZE, 5000, in_msg };
146         res = ioctl(kb->handle - 1, USBDEVFS_CONTROL, &transfer);
147     }
148     if(res <= 0){
149         ckb_err_fn("%s\n", file, line, res ? strerror(errno) : "No data read");
150         if(res == -1 && errno == ETIMEDOUT)
151             return -1;
152         else
153             return 0;
154     } else if(res != MSG_SIZE)
155         ckb_warn_fn("Read %d bytes (expected %d)\n", file, line, res,
156 MSG_SIZE);
157 #ifdef DEBUG_USB_RECV
158     char converted[MSG_SIZE*3 + 1];
159     for(int i=0;i<MSG_SIZE;i++)
160         sprintf(&converted[i*3], "%02x ", in_msg[i]);
161     ckb_warn_fn("Recv %s\n", file, line, converted);
162 #endif
163     return res;
164 }
```

Here is the caller graph for this function:



9.42.3.12 `int os_usbrecv (usbdevice * kb, const uchar * out_msg, int is_recv, const char * file, int line)`

Parameters

<i>kb</i>	THE usbdevice*
<i>out_msg</i>	the MSGSIZE char long buffer to send
<i>is_recv</i>	if true, just send an ioctl for further reading packets. If false, send the data at out_msg .
<i>file</i>	for debugging
<i>line</i>	for debugging

Returns

-1 on timeout (try again), 0 on hard error, number of bytes sent otherwise

os_usbrecv has two functions:

- if `is_recv == false`, it tries to send a given MSG_SIZE buffer via the usb interface given with kb.
- otherwise a request is sent via the usb device to initiate the receiving of a message from the remote device.

The functionality for sending distinguishes two cases, depending on the version number of the firmware of the connected device:

If the firmware is less or equal 1.2, the transmission is done via an `ioctl()`. The `ioctl()` is given a struct `usbdevfs_ctrltransfer`, in which the relevant parameters are entered:

bRequest-Type	bRequest	wValue	EP	size	Timeout	data
0x21	0x09	0x0200	endpoint / IF to be addressed from epcount-1	MSG_SIZE	5000 (=5ms)	the message buffer pointer
Host to Device, Type=Class, Recipient=Interface	9 = Send data?	specific	last or pre-last device #	64	5000	out_msg

The `ioctl` command is `USBDEVFS_CONTROL`.

The same constellation is used if the device is requested to send its data (`is_recv = true`).

For a more recent firmware and `is_recv = false`, the `ioctl` command `USBDEVFS_CONTROL` is not used (this tells the bus to enter the control mode), but the bulk method is used: `USBDEVFS_BULK`. This is astonishing, because all of the endpoints are type Interrupt, not bulk.

Anyhow, for this purpose a different structure is used for the `ioctl()` (struct **usbdevfs_bulktransfer**) and this is also initialized differently:

The length and timeout parameters are given the same values as above. The formal parameter `out_msg` is also passed as a buffer pointer. For the endpoints, the firmware version is differentiated again:

For a firmware version between 1.3 and <2.0 endpoint 4 is used, otherwise (it can only be ≥ 2.0) endpoint 3 is used.

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

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

The `ioctl()` - no matter what type - returns the number of bytes sent. Now comes the usual check:

- If the return value is -1 AND the error is a timeout (ETIMEDOUT), `os_usb_send()` 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 70 of file `usb_linux.c`.

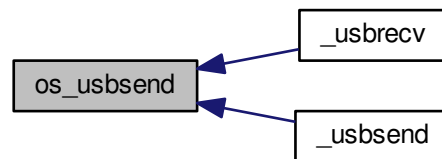
References `ckb_err_fn`, `ckb_warn_fn`, `usbdevice::epcount`, `usbdevice::fwversion`, `usbdevice::handle`, and `MSG_SIZE`.

Referenced by `_usb_recv()`, and `_usb_send()`.

```

70                                     {
71     int res;
72     if (kb->fwversion >= 0x120 && !is_recv) {
73         struct usbdevfs_bulktransfer transfer;
74         memset(&transfer, 0, sizeof(transfer));
75         transfer.ep = (kb->fwversion >= 0x130 && kb->fwversion < 0x200) ? 4 : 3;
76         transfer.len = MSG_SIZE;
77         transfer.timeout = 5000;
78         transfer.data = (void*)out_msg;
79         res = ioctl(kb->handle - 1, USBDEVFS_BULK, &transfer);
80     } else {
81         struct usbdevfs_ctrltransfer transfer = { 0x21, 0x09, 0x0200, kb->
82         epcount - 1, MSG_SIZE, 5000, (void*)out_msg };
83         res = ioctl(kb->handle - 1, USBDEVFS_CONTROL, &transfer);
84     }
85     if (res <= 0) {
86         ckb_err_fn(" %s, res = 0x%x\n", file, line, res ? strerror(errno) : "No data written",
87         res);
88         if (res == -1 && errno == ETIMEDOUT)
89             return -1;
90         else
91             return 0;
92     } else if (res != MSG_SIZE)
93         ckb_warn_fn("Wrote %d bytes (expected %d)\n", file, line, res,
94         MSG_SIZE);
95 #ifdef DEBUG_USB
96     char converted[MSG_SIZE*3 + 1];
97     for(int i=0; i<MSG_SIZE; i++)
98         sprintf(&converted[i*3], "%02x ", out_msg[i]);
99     ckb_warn_fn("Sent %s\n", file, line, converted);
100 #endif
101     return res;
102 }
```


Here is the caller graph for this function:



9.42.3.13 `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_K65`, `P_K65_LUX`, `P_K65_NRGB`, `P_K65_RFIRE`, `P_K70`, `P_K70_LUX`, `P_K70_LUX_NRGB`, `P_K70_NRGB`, `P_K70_RFIRE`, `P_K70_RFIRE_NRGB`, `P_K95`, `P_K95_NRGB`, `P_K95_PLATINUM`, `P_M65`, `P_M65_PRO`, `P_SABRE_L`, `P_SABRE_N`, `P_SABRE_O`, `P_SABRE_O2`, `P_SCIMITAR`, `P_SCIMITAR_PRO`, `P_STRAFE`, and `P_STRAFE_NRGB`.

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

```

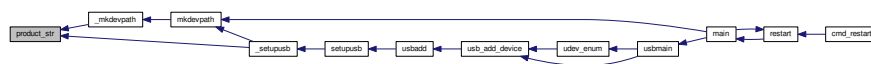
70
71     if (product == P_K95 || product == P_K95_NRGB || product ==
72         P_K95_PLATINUM)
73         return "k95";
74     if (product == P_K70 || product == P_K70_NRGB || product ==
75         P_K70_LUX || product == P_K70_LUX_NRGB || product ==
  
```

```

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)
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:



9.42.3.14 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:



9.42.3.15 void setupusb (usbdevice * kb)

Attention

Lock a device's dmux (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. dmux 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](#).

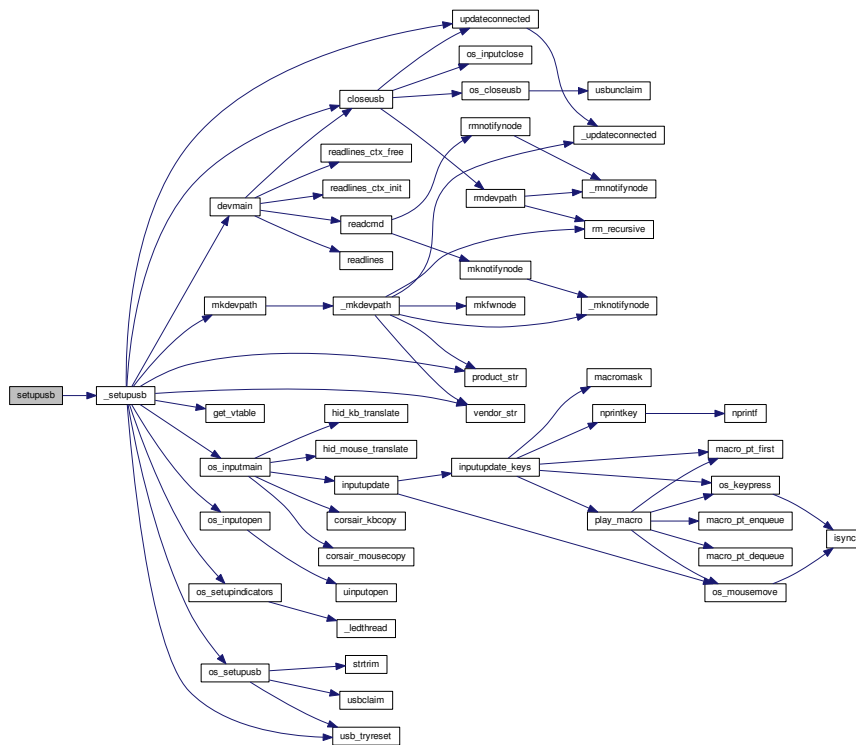
Referenced by [usbadd\(\)](#).

```

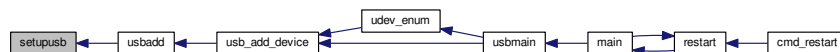
386     {
387         pthread_mutex_lock(&imutex(kb));
388         if (pthread_create(&kb->thread, 0, _setupusb, kb))
389             ckb_err("Failed to create USB thread\n");
390     }

```

Here is the call graph for this function:



Here is the caller graph for this function:



9.42.3.16 int usb_tryreset (usbdevice * kb)

Parameters

in, out	kb	THE usbdevice*
in	reset_stop	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.

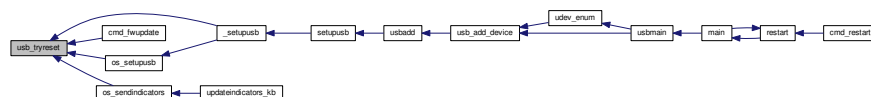
References [ckb_err](#), [ckb_info](#), [reset_stop](#), and [resetusb](#).

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

```

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:



9.42.3.17 void usbkill ()

Definition at line 840 of file usb_linux.c.

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

```

840     {
841         udev_unref(udev);
842         udev = 0;
843     }

```

Here is the caller graph for this function:



9.42.3.18 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 780 of file [usb_linux.c](#).

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

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

```

780     {
781         // Load the uinput module (if it's not loaded already)
782         if(system("modprobe uinput") != 0)
783             ckb_warn("Failed to load uinput module\n");
784
785         if(!(udev = udev_new())) {
786             ckb_fatal("Failed to initialize udev in usbmain(), usb_linux.c\n");
787             return -1;
788         }
789
790         udev_enum();
791
792         // Done scanning. Enter a loop to poll for device updates
793         struct udev_monitor* monitor = udev_monitor_new_from_netlink(udev, "udev");
794         udev_monitor_filter_add_match_subsystem_devtype(monitor, "usb", 0);
795         udev_monitor_enable_receiving(monitor);
796         // Get an fd for the monitor
797         int fd = udev_monitor_get_fd(monitor);
798         fd_set fds;
799         while(udev){
800             FD_ZERO(&fds);
801             FD_SET(fd, &fds);
802             // Block until an event is read
803             if(select(fd + 1, &fds, 0, 0, 0) > 0 && FD_ISSET(fd, &fds)){
804                 struct udev_device* dev = udev_monitor_receive_device(monitor);
805                 if(!dev)
806                     continue;

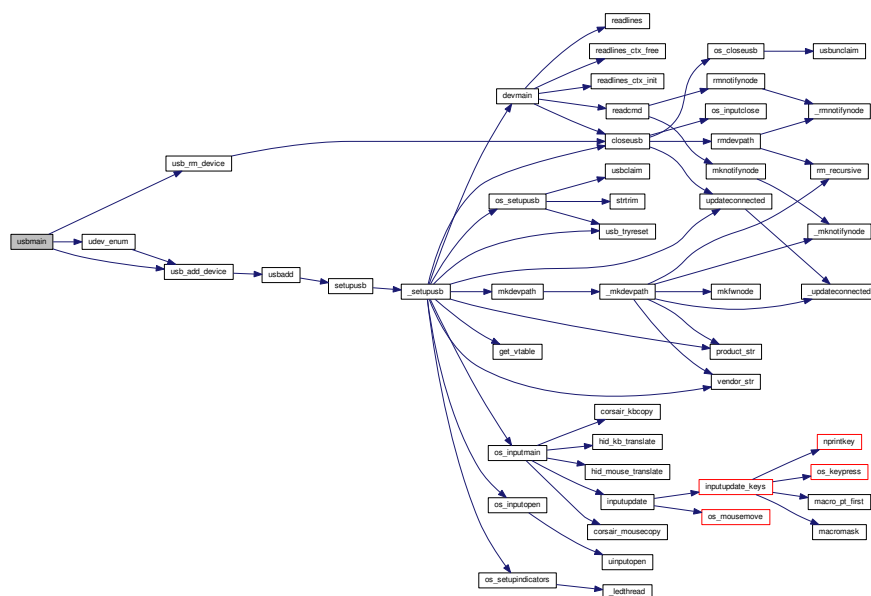
```

```

818         const char* action = udev_device_get_action(dev);
819         if(!action){
820             udev_device_unref(dev);
821             continue;
822         }
823         // Add/remove device
824         if(!strcmp(action, "add")){
825             int res = usb_add_device(dev);
826             if(res == 0)
827                 continue;
828             // If the device matched but the handle wasn't opened correctly, re-enumerate (this
            sometimes solves the problem)
829             if(res == -1)
830                 udev_enumerate();
831             } else if(!strcmp(action, "remove"))
832                 usb_rm_device(dev);
833             udev_device_unref(dev);
834         }
835     }
836     udev_monitor_unref(monitor);
837     return 0;
838 }

```

Here is the call graph for this function:



Here is the caller graph for this function:



9.42.3.19 const char* vendor_str (short vendor)

`vendor_str` Vendor/product string representations

Macros

- `#define DEBUG`
all open usb devices have their system path names here in this array.
- `#define DEBUG`
all open usb devices have their system path names here in this array.
- `#define TEST_RESET(op)`
TEST_RESET doesa "try / catch" for resetting the usb interface.
- `#define N_MODELS (sizeof(models) / sizeof(_model))`

Functions

- `int os_usbsend (usbdevice *kb, const uchar *out_msg, int is_recv, const char *file, int line)`
os_usbsend sends a data packet (MSG_SIZE = 64) Bytes long
- `int os_usbrecv (usbdevice *kb, uchar *in_msg, const char *file, int line)`
os_usbrecv receives a max MSGSIZE long buffer from usb device
- `int _nk95cmd (usbdevice *kb, uchar bRequest, ushort wValue, const char *file, int line)`
_nk95cmd If we control a non RGB keyboard, set the keyboard via ioctl with usbdevfs_ctrltransfer
- `void os_sendindicators (usbdevice *kb)`
- `void * os_inputmain (void *context)`
os_inputmain This function is run in a separate thread and will be detached from the main thread, so it needs to clean up its own resources.
- `static int usbunclaim (usbdevice *kb, int resetting)`
- `void os_closeusb (usbdevice *kb)`
- `static int usbclaim (usbdevice *kb)`
- `int os_resetusb (usbdevice *kb, const char *file, int line)`
- `void strtrim (char *string)`
- `int os_setupusb (usbdevice *kb)`
- `int usbadd (struct udev_device *dev, short vendor, short product)`
- `static int usb_add_device (struct udev_device *dev)`
Add a udev device. 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

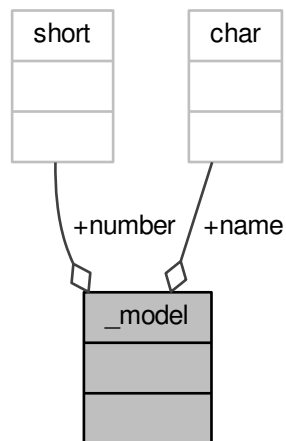
- `static char kbsyspath [9][FILENAME_MAX]`
- `static struct udev * udev`
struct udev is defined in `/usr/include/libudev.h`
- `pthread_t usbthread`
- `pthread_t udevthread`
- `static _model models []`

9.43.1 Data Structure Documentation

9.43.1.1 struct _model

Definition at line 650 of file usb_linux.c.

Collaboration diagram for _model:



Data Fields

const char *	name	
short	number	

9.43.2 Macro Definition Documentation

9.43.2.1 #define DEBUG

Definition at line 13 of file usb_linux.c.

9.43.2.2 #define DEBUG

Definition at line 13 of file usb_linux.c.

9.43.2.3 #define N_MODELS (sizeof(models) / sizeof(_model))

Definition at line 687 of file usb_linux.c.

Referenced by usb_add_device().

9.43.2.4 #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 483 of file usb_linux.c.

Referenced by os_resetusb().

9.43.3 Function Documentation

9.43.3.1 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 192 of file `usb_linux.c`.

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

```

192                                     {
193     if(kb->product != P_K95_NRGB)
194         return 0;
195     struct usbdevfs_ctrltransfer transfer = { 0x40, bRequest, wValue, 0, 0, 5000, 0 };
196     int res = ioctl(kb->handle - 1, USBDEVFS_CONTROL, &transfer);
197     if(res <= 0){
198         ckb_err_fn("%s\n", file, line, res ? strerror(errno) : "No data written");
199         return 1;
200     }
201     return 0;
202 }
```

9.43.3.2 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 439 of file `usb_linux.c`.

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

Referenced by `closeusb()`.

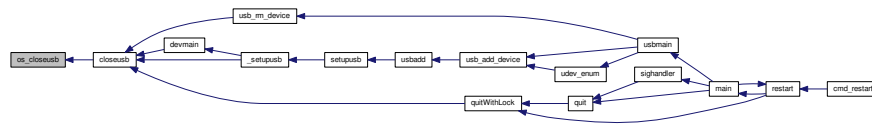
```

439                                     {
440     if(kb->handle){
441         usbunclaim(kb, 0);
442         close(kb->handle - 1);
443     }
444     if(kb->udev)
445         udev_device_unref(kb->udev);
446     kb->handle = 0;
447     kb->udev = 0;
448     kbsyspath[INDEX_OF(kb, keyboard)][0] = 0;
449 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



9.43.3.3 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 242 of file `usb_linux.c`.

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

```

242                                     {
243     usbdevice* kb = context;
244     int fd = kb->handle - 1;
245     short vendor = kb->vendor, product = kb->product;
246     int index = INDEX_OF(kb, keyboard);
247     ckb_info("Starting input thread for %s%d\n", devpath, index);
248
249     int urbcount = IS_RGB(vendor, product) ? (kb->epcount - 1) : kb->
250     epcount;
251     if (urbcount == 0) {
252         ckb_err("urbcount = 0, so there is nothing to claim in os_inputmain()\n");
253         return 0;
254     }
255
256     struct usbdevfs_urb urbs[urbcount + 1];
257     memset(urbs, 0, sizeof(urbs));
258
259     urbs[0].buffer_length = 8;
260     if (urbcount > 1 && IS_RGB(vendor, product)) {
261         if (IS_MOUSE(vendor, product))
262             urbs[1].buffer_length = 10;
263         else
264             urbs[1].buffer_length = 21;
265         urbs[2].buffer_length = MSG_SIZE;
266         if (urbcount != 3)
267             urbs[urbcount - 1].buffer_length = MSG_SIZE;
268     } else {
269         urbs[1].buffer_length = 4;
270         urbs[2].buffer_length = 15;
271     }
272
273     for (int i = 0; i < urbcount; i++) {
274         urbs[i].type = USBDEVFS_URB_TYPE_INTERRUPT;
275         urbs[i].endpoint = 0x80 | (i + 1);
276         urbs[i].buffer = malloc(urbs[i].buffer_length);
277         ioctl(fd, USBDEVFS_SUBMITURB, urbs + i);
278     }
279
280     while (1) {
281         struct usbdevfs_urb* urb = 0;

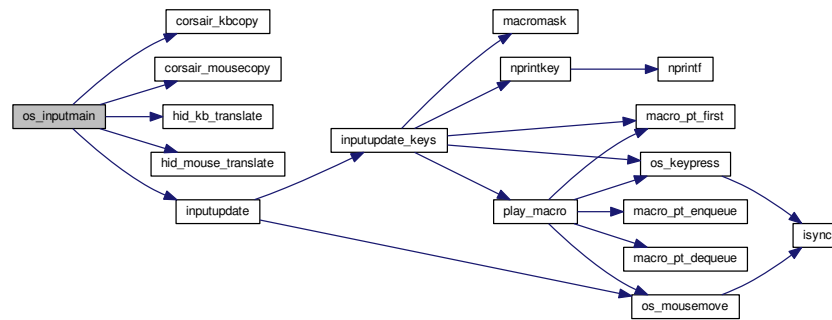
```

```

302
305     if (ioctl(fd, USBDEVFS_REAPURB, &urb)) {
306         if (errno == ENODEV || errno == ENOENT || errno == ESHUTDOWN)
307             // Stop the thread if the handle closes
308             break;
309         else if (errno == EPIPE && urb) {
310             ioctl(fd, USBDEVFS_CLEAR_HALT, &urb->endpoint);
311             // Re-submit the URB
312             if (urb)
313                 ioctl(fd, USBDEVFS_SUBMITURB, urb);
314             urb = 0;
315         }
316         continue;
317     }
318 }
319
323     if (urb) {
324
325         pthread_mutex_lock(&mutex(kb));
326         if (IS_MOUSE(vendor, product)) {
327             switch (urb->actual_length) {
328                 case 8:
329                 case 10:
330                 case 11:
331                     // HID mouse input
332                     hid_mouse_translate(kb->input.keys, &kb->
333 input_rel_x, &kb->input_rel_y, -(urb->endpoint & 0xF), urb->actual_length, urb->buffer)
334 ;
335                     break;
336                 case MSG_SIZE:
337                     // Corsair mouse input
338                     corsair_mousecopy(kb->input.keys, -(urb->endpoint & 0xF), urb
339 ->buffer);
340                     break;
341             }
342         } else if (IS_RGB(vendor, product)) {
343             switch (urb->actual_length) {
344                 case 8:
345                     // RGB EP 1: 6KRO (BIOS mode) input
346                     hid_kb_translate(kb->input.keys, -1, urb->actual_length, urb->
347 buffer);
348                     break;
349                 case 21:
350                 case 5:
351                     // RGB EP 2: NKRO (non-BIOS) input. Accept only if keyboard is inactive
352                     if (!kb->active)
353                         hid_kb_translate(kb->input.keys, -2, urb->actual_length,
354 urb->buffer);
355                     break;
356                 case MSG_SIZE:
357                     // RGB EP 3: Corsair input
358                     corsair_kbcopy(kb->input.keys, -(urb->endpoint & 0xF), urb->
359 buffer);
360                     break;
361             }
362         } else {
363             // Non-RGB input
364             hid_kb_translate(kb->input.keys, urb->endpoint & 0xF, urb->
365 actual_length, urb->buffer);
366         }
367         input_update(kb);
368         pthread_mutex_unlock(&mutex(kb));
369
370         ioctl(fd, USBDEVFS_SUBMITURB, urb);
371         urb = 0;
372     }
373 }
374
375     ckb_info("Stopping input thread for %s%d\n", devpath, index);
376     for (int i = 0; i < urbcount; i++) {
377         ioctl(fd, USBDEVFS_DISCARDURB, urbs + i);
378         free(urbs[i].buffer);
379     }
380     return 0;
381 }

```

Here is the call graph for this function:



Here is the caller graph for this function:



9.43.3.4 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 501 of file usb_linux.c.

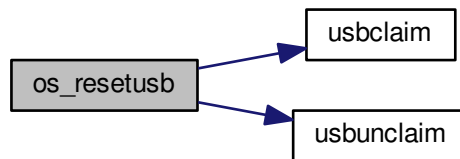
References usbdevice::handle, TEST_RESET, usbclaim(), and usbunclaim().

Referenced by _resetusb().

```

501                                     {
502     TEST_RESET(usbunclaim(kb, 1));
503     TEST_RESET(ioctl(kb->handle - 1, USBDEVFS_RESET));
504     TEST_RESET(usbclaim(kb));
505     // Success!
506     return 0;
507 }
  
```


Here is the call graph for this function:



Here is the caller graph for this function:



9.43.3.5 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, Recipient=Interface (why not endpoint?)	9 = SEND?	specific	0	1	500	struct* kb->ileds

The ioctl command is USBDEVFS_CONTROL.

Definition at line 217 of file usb_linux.c.

References ckb_err, usbdevice::handle, usbdevice::ileds, and usb_tryreset().

Referenced by updateindicators_kb().

```

217                                     {
218     static int countForReset = 0;
219     struct usbdevfs_ctrltransfer transfer = { 0x21, 0x09, 0x0200, 0x00, 1, 500, &kb->
ileds };
220     int res = ioctl(kb->handle - 1, USBDEVFS_CONTROL, &transfer);
221     if(res <= 0) {
  
```

```

222         ckb_err("%s\n", res ? strerror(errno) : "No data written");
223         if (usb_tryreset(kb) == 0 && countForReset++ < 3) {
224             os_sendindicators(kb);
225         }
226     }
227 }

```

Here is the call graph for this function:



Here is the caller graph for this function:



9.43.3.6 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

- < Try to reset the device and recall the function
- < Don't do this endless in recursion
- < [os_setupusb\(\)](#) has a return value (used as boolean)

Definition at line 539 of file usb_linux.c.

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

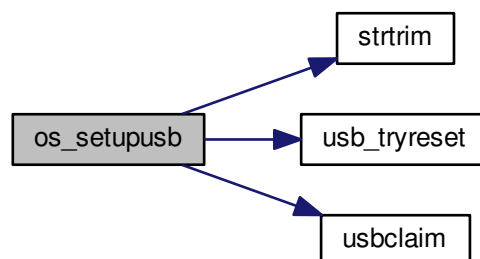
Referenced by [_setupusb\(\)](#).

```

539     {
540     struct udev_device* dev = kb->udev;
541     const char* name = udev_device_get_sysattr_value(dev, "product");
542     if(name)
543         strncpy(kb->name, name, KB_NAME_LEN);
544     strtrim(kb->name);
545     const char* serial = udev_device_get_sysattr_value(dev, "serial");
546     if(serial)
547         strncpy(kb->serial, serial, SERIAL_LEN);
548     strtrim(kb->serial);
549     const char* firmware = udev_device_get_sysattr_value(dev, "bcdDevice");
550     if(firmware)
551         sscanf(firmware, "%hx", &kb->fwversion);
552     else
553         kb->fwversion = 0;
554     int index = INDEX_OF(kb, keyboard);
555     ckb_info("Connecting %s at %s%d\n", kb->name, devpath, index);
556
557     const char* ep_str = udev_device_get_sysattr_value(dev, "bNumInterfaces");
558 #ifdef DEBUG
559     ckb_info("Connecting %s at %s%d\n", kb->name, devpath, index);
560     ckb_info("claiming interfaces. name=%s, serial=%s, firmware=%s; Got >>%s<< as ep_str\n", name,
561     serial, firmware, ep_str);
562 #endif //DEBUG
563     kb->epcount = 0;
564     if(ep_str)
565         sscanf(ep_str, "%d", &kb->epcount);
566     if(kb->epcount < 2){
567         // IF we have an RGB KB with 0 or 1 endpoints, it will be in BIOS mode.
568         ckb_err("Unable to read endpoint count from udev, assuming %d and reading >>%s<< or device
569         is in BIOS mode\n", kb->epcount, ep_str);
570         if (usb_tryreset(kb) == 0) {
571             static int retryCount = 0;
572             if (retryCount++ < 5) {
573                 return os_setupusb(kb);
574             }
575         }
576         return -1;
577         // ToDo are there special versions we have to detect? If there are, that was the old code to handle
578         it:
579         // This shouldn't happen, but if it does, assume EP count based on ckb_warn what the device is
580         supposed to have
581         // kb->epcount = (HAS_FEATURES(kb, FEAT_RGB) ? 4 : 3);
582         // ckb_warn("Unable to read endpoint count from udev, assuming %d and reading >>%s<<...\n",
583         kb->epcount, ep_str);
584     }
585     if(usbclaim(kb)){
586         ckb_err("Failed to claim interfaces: %s\n", strerror(errno));
587         return -1;
588     }
589     return 0;
590 }

```

Here is the call graph for this function:



Here is the caller graph for this function:



9.43.3.7 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-usb send ()).

So all the receiving is done via an ioctl() like in os_usb send. The ioctl() is given a struct usbdevfs_ctrltransfer, in which the relevant parameters are entered:

bRequest-Type	bRequest	wValue	EP	size	Timeout	data
0xA1	0x01	0x0200	endpoint to be addressed from epcount - 1	MSG_SIZE	5ms	the message buffer pointer
Device to Host, Type=Class, Recipient=Interface	1 = RECEIVE?	specific	Interface #	64	5000	in_msg

The ioctl() returns the number of bytes received. Here is the usual check again:

- If the return value is -1 AND the error is a timeout (ETIMEDOUT), [os_usbrecv\(\)](#) will return -1 to indicate that it is probably a recoverable problem and a retry is recommended.
- For another negative value or other error identifier OR 0 bytes are received, 0 is returned as an identifier for a heavy error.
- In all other cases, the function returns the number of bytes received.

If this is not the entire blocksize (MSG_SIZE bytes), an error message is issued on the standard error channel [warning "Read YY bytes (expected 64)"].

Definition at line 132 of file usb_linux.c.

References ckb_err_fn, ckb_warn_fn, usbdevice::epcount, usbdevice::handle, and MSG_SIZE.

Referenced by _usbrecv().

```

132                                     {
133     int res;
134     // This is what CUE does, but it doesn't seem to work on linux.
135     /*if(kb->fwversion >= 0x130){
136         struct usbdevfs_bulktransfer transfer;
137         memset(&transfer, 0, sizeof(transfer));
138         transfer.ep = 0x84;
139         transfer.len = MSG_SIZE;
140         transfer.timeout = 5000;
141         transfer.data = in_msg;
142         res = ioctl(kb->handle - 1, USBDEVFS_BULK, &transfer);
143     } else {*/
144     struct usbdevfs_ctrltransfer transfer = { 0xa1, 0x01, 0x0300, kb->
145         epcount - 1, MSG_SIZE, 5000, in_msg };
146     res = ioctl(kb->handle - 1, USBDEVFS_CONTROL, &transfer);
  
```

```

146     //}
147     if(res <= 0){
148         ckb_err_fn("%s\n", file, line, res ? strerror(errno) : "No data read");
149         if(res == -1 && errno == ETIMEDOUT)
150             return -1;
151         else
152             return 0;
153     } else if(res != MSG_SIZE)
154         ckb_warn_fn("Read %d bytes (expected %d)\n", file, line, res,
MSG_SIZE);
155 #ifdef DEBUG_USB_RECV
156     char converted[MSG_SIZE*3 + 1];
157     for(int i=0;i<MSG_SIZE;i++)
158         sprintf(&converted[i*3], "%02x ", in_msg[i]);
159     ckb_warn_fn("Recv %s\n", file, line, converted);
160 #endif
161     return res;
162 }

```

Here is the caller graph for this function:



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

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_usb_send()` 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 70 of file `usb_linux.c`.

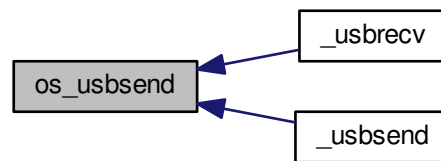
References `ckb_err_fn`, `ckb_warn_fn`, `usbdevice::epcount`, `usbdevice::fwversion`, `usbdevice::handle`, and `MSG_SIZE`.

Referenced by `_usb_recv()`, and `_usb_send()`.

```

70                                     {
71     int res;
72     if (kb->fwversion >= 0x120 && !is_recv) {
73         struct usbdevfs_bulktransfer transfer;
74         memset(&transfer, 0, sizeof(transfer));
75         transfer.ep = (kb->fwversion >= 0x130 && kb->fwversion < 0x200) ? 4 : 3;
76         transfer.len = MSG_SIZE;
77         transfer.timeout = 5000;
78         transfer.data = (void*)out_msg;
79         res = ioctl(kb->handle - 1, USBDEVFS_BULK, &transfer);
80     } else {
81         struct usbdevfs_ctrltransfer transfer = { 0x21, 0x09, 0x0200, kb->
82         epcount - 1, MSG_SIZE, 5000, (void*)out_msg };
83         res = ioctl(kb->handle - 1, USBDEVFS_CONTROL, &transfer);
84     }
85     if (res <= 0) {
86         ckb_err_fn(" %s, res = 0x%x\n", file, line, res ? strerror(errno) : "No data written",
87         res);
88         if (res == -1 && errno == ETIMEDOUT)
89             return -1;
90         else
91             return 0;
92     } else if (res != MSG_SIZE)
93         ckb_warn_fn("Wrote %d bytes (expected %d)\n", file, line, res,
94         MSG_SIZE);
95 #ifdef DEBUG_USB
96     char converted[MSG_SIZE*3 + 1];
97     for(int i=0; i<MSG_SIZE; i++)
98         sprintf(&converted[i*3], "%02x ", out_msg[i]);
99     ckb_warn_fn("Sent %s\n", file, line, converted);
100 #endif
101     return res;
102 }
```

Here is the caller graph for this function:



9.43.3.9 void `strtrim` (`char *` *string*)

`strtrim` trims a string by removing leading and trailing spaces.

Parameters

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

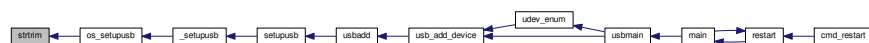
Definition at line 514 of file `usb_linux.c`.

Referenced by `os_setupusb()`.

```

514     {
515         // Find last non-space
516         char* last = string;
517         for(char* c = string; *c != 0; c++){
518             if(!isspace(*c))
519                 last = c;
520         }
521         last[1] = 0;
522         // Find first non-space
523         char* first = string;
524         for(; *first != 0; first++){
525             if(!isspace(*first))
526                 break;
527         }
528         if(first != string)
529             memmove(string, first, last - first);
530     }
  
```

Here is the caller graph for this function:



9.43.3.10 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 *`.

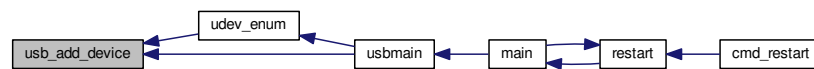
Use `udev_list_entry_foreach()` to iterate through the hit set.

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\(\)](#).

9.43.3.11 `static int usb_add_device (struct udev_device * dev)` `[static]`

If the device id can be found, call [usbadd\(\)](#) with the appropriate parameters.

Here is the caller graph for this function:



9.43.3.12 static void usb_rm_device (struct udev_device * dev) [static]

Parameters

<i>dev</i>	the functions usb_*_device get a struct udev* 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 725 of file usb_linux.c.

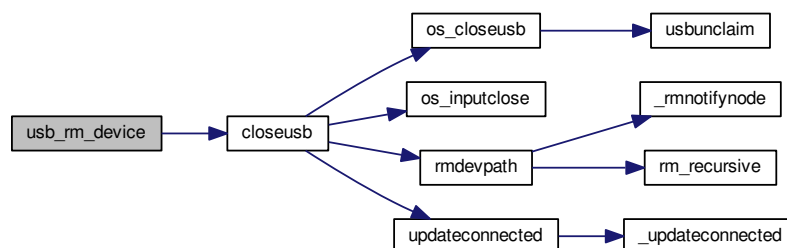
References [closeusb\(\)](#), [DEV_MAX](#), [devmutex](#), [kbsyspath](#), and [keyboard](#).

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

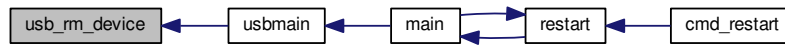
```

725                                     {
726     // Device removed. Look for it in our list of keyboards
727     const char* syspath = udev_device_get_syspath(dev);
728     if(!syspath || syspath[0] == 0)
729         return;
730     for(int i = 1; i < DEV_MAX; i++){
731         pthread_mutex_lock(devmutex + i);
732         if(!strcmp(syspath, kbsyspath[i]))
733             closeusb(keyboard + i);
734         pthread_mutex_unlock(devmutex + i);
735     }
736 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



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

Definition at line 598 of file usb_linux.c.

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

Referenced by usb_add_device().

```

598                                     {
599     const char* path = udev_device_get_devnode(dev);
600     const char* syspath = udev_device_get_syspath(dev);
601     if(!path || !syspath || path[0] == 0 || syspath[0] == 0){
602         ckb_err("Failed to get device path\n");
603         return -1;
604     }
605     #ifdef DEBUG
606         ckb_info(">>>vendor = 0x%x, product = 0x%x, path = %s, syspath = %s\n", vendor, product, path,
        syspath);
607     #endif // DEBUG
608     // Find a free USB slot
609     for(int index = 1; index < DEV_MAX; index++){
610         usbdevice* kb = keyboard + index;
611         if(pthread_mutex_trylock(dmutex(kb))){
612             // If the mutex is locked then the device is obviously in use, so keep going
613             if(!strcmp(syspath, kbsyspath[index])){
614                 // Make sure this existing keyboard doesn't have the same syspath (this shouldn't happen)
615                 return 0;
616             }
617             continue;
618         }
619         if(!IS_CONNECTED(kb)){
620             // Open the sysfs device
621             kb->handle = open(path, O_RDWR) + 1;
622             if(kb->handle <= 0){
623                 ckb_err("Failed to open USB device: %s\n", strerror(errno));
624                 kb->handle = 0;
625                 pthread_mutex_unlock(dmutex(kb));
626                 return -1;
627             } else {
628                 // Set up device
629                 kb->udev = dev;
630                 kb->vendor = vendor;
631                 kb->product = product;
632                 strncpy(kbsyspath[index], syspath, FILENAME_MAX);
633                 // Mutex remains locked
634                 setupusb(kb);
635                 return 0;
636             }
637         }
638         pthread_mutex_unlock(dmutex(kb));
639     }
640     ckb_err("No free devices\n");
641     return -1;
642 }
  
```

```

graph LR
    usbmain --> udev_enum
    udev_enum --> usbmain
    udev_enum --> usb_add_device
    usbmain --> usb_add_device
    usbmain --> restart
    restart --> main
    main --> restart
    main --> usbmain
    main --> cmd_restart
    cmd_restart --> restart
    usbmain --> usb_badd[usbadd]
    usb_badd --> usbmain
    usbmain --> usb_add_device
    usb_add_device --> usbmain

```

usbclaim does claiming all EPs for the usb device gicen by kb.

<i>kb</i>	THE usbdevice*
-----------	----------------

0 on success, -1 otherwise.

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.

References `ckb_err`, `ckb_info`, `usbdevice::epcount`, and `usbdevice::handle`.

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

```

463     {
464         int count = kb->epcount;
465 #ifdef DEBUG
466         ckb_info("claiming %d endpoints\n", count);
467 #endif // DEBUG
468
469         for (int i = 0; i < count; i++) {
470             struct usbdevfs_ioctl ctl = { i, USBDEVFS_DISCONNECT, 0 };
471             ioctl(kb->handle - 1, USBDEVFS_IOCTL, &ctl);
472             if (ioctl(kb->handle - 1, USBDEVFS_CLAIMINTERFACE, &i)) {
473                 ckb_err("Failed to claim interface %d: %s\n", i, strerror(errno));
474                 return -1;
475             }
476         }
477         return 0;
478     }

```

Here is the caller graph for this function:



9.43.3.15 void usbkill ()

Definition at line 840 of file usb_linux.c.

Referenced by quitWithLock().

```

840     {
841         udev_unref(udev);
842         udev = 0;
843     }

```

Here is the caller graph for this function:



9.43.3.16 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 780 of file usb_linux.c.

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

Referenced by main().

```

780     {
785         // Load the uinput module (if it's not loaded already)
786         if(system("modprobe uinput") != 0)
787             ckb_warn("Failed to load uinput module\n");
788
792         if(!(udev = udev_new())) {
793             ckb_fatal("Failed to initialize udev in usbmain(), usb_linux.c\n");
794             return -1;
795         }
796
799         udev_enum();
800
803         // Done scanning. Enter a loop to poll for device updates
804         struct udev_monitor* monitor = udev_monitor_new_from_netlink(udev, "udev");
805         udev_monitor_filter_add_match_subsystem_devtype(monitor, "usb", 0);
806         udev_monitor_enable_receiving(monitor);
807         // Get an fd for the monitor
808         int fd = udev_monitor_get_fd(monitor);
809         fd_set fds;
810         while(udev){
811             FD_ZERO(&fds);
812             FD_SET(fd, &fds);
813             // Block until an event is read
814             if(select(fd + 1, &fds, 0, 0, 0) > 0 && FD_ISSET(fd, &fds)){
815                 struct udev_device* dev = udev_monitor_receive_device(monitor);
816                 if(!dev)
817                     continue;
818                 const char* action = udev_device_get_action(dev);
819                 if(!action){
820                     udev_device_unref(dev);
821                     continue;
822                 }
823                 // Add/remove device
824                 if(!strcmp(action, "add")){
825                     int res = usb_add_device(dev);
826                     if(res == 0)
827                         continue;
828                     // If the device matched but the handle wasn't opened correctly, re-enumerate (this
829                     // sometimes solves the problem)
830                     if(res == -1)
831                         udev_enum();
832                 } else if(!strcmp(action, "remove"))
833                     usb_rm_device(dev);
834                 udev_device_unref(dev);
835             }
836             udev_monitor_unref(monitor);
837             return 0;
838 }

```


Index

- [_model](#), [348](#)
- [_readlines_ctx](#), [69](#)
- [_DEFAULT_SOURCE](#)
 - [os.h](#), [233](#)
- [_GNU_SOURCE](#)
 - [os.h](#), [233](#)
- [__FILE_NOPATH__](#)
 - [includes.h](#), [109](#)
- [_cmd_get](#)
 - [notify.c](#), [221](#)
- [_cmd_macro](#)
 - [input.c](#), [113](#)
- [_freeprofile](#)
 - [profile.c](#), [235](#)
- [_ledthread](#)
 - [input_linux.c](#), [137](#)
- [_mkdevpath](#)
 - [devnode.c](#), [70](#)
- [_mknotifynode](#)
 - [devnode.c](#), [73](#)
- [_nk95cmd](#)
 - [usb.h](#), [319](#)
 - [usb_linux.c](#), [349](#)
- [_resetusb](#)
 - [usb.c](#), [290](#)
 - [usb.h](#), [320](#)
- [_rmnotifynode](#)
 - [devnode.c](#), [74](#)
- [_setupusb](#)
 - [usb.c](#), [290](#)
- [_start_dev](#)
 - [device.c](#), [46](#)
- [_updateconnected](#)
 - [devnode.c](#), [74](#)
- [_usbrecv](#)
 - [usb.c](#), [294](#)
 - [usb.h](#), [321](#)
- [_usbsend](#)
 - [usb.c](#), [296](#)
 - [usb.h](#), [323](#)
- [ACCEL](#)
 - [command.h](#), [39](#)
- [ACTIVE](#)
 - [command.h](#), [39](#)
- [ACT_LIGHT](#)
 - [device.h](#), [50](#)
- [ACT_LOCK](#)
 - [device.h](#), [50](#)
- [ACT_M1](#)
 - [device.h](#), [50](#)
- [ACT_M2](#)
 - [device.h](#), [50](#)
- [ACT_M3](#)
 - [device.h](#), [51](#)
- [ACT_MR_RING](#)
 - [device.h](#), [51](#)
- [ACT_NEXT](#)
 - [device.h](#), [51](#)
- [ACT_NEXT_NOWRAP](#)
 - [device.h](#), [51](#)
- [active](#)
 - [devcmd.__unnamed__](#), [28](#)
- [allocprofile](#)
 - [devcmd.__unnamed__](#), [28](#)
 - [profile.c](#), [235](#)
 - [profile.h](#), [251](#)
- [BIND](#)
 - [command.h](#), [40](#)
- [BR1](#)
 - [led_keyboard.c](#), [197](#)
- [BR2](#)
 - [led_keyboard.c](#), [197](#)
- [BR4](#)
 - [led_keyboard.c](#), [197](#)
- [BTN_WHEELDOWN](#)
 - [keymap.h](#), [150](#)
- [BTN_WHEELUP](#)
 - [keymap.h](#), [150](#)
- [BUILD.md](#), [31](#)
- [BUTTON_HID_COUNT](#)
 - [keymap.c](#), [144](#)
- [bind](#)
 - [devcmd.__unnamed__](#), [28](#)
- [binding](#), [275](#)
- [bit_reverse_table](#)
 - [led_keyboard.c](#), [205](#)
- [CMD_FIRST](#)
 - [command.h](#), [39](#)
- [CMD_LAST](#)
 - [command.h](#), [40](#)
- [CMD_VT_FIRST](#)
 - [command.h](#), [39](#)
- [CLEAR_KEYBIT](#)
 - [structures.h](#), [284](#)
- [CMD_COUNT](#)
 - [command.h](#), [38](#)
- [CMD_DEV_COUNT](#)

- command.h, 38
- ckb_err
 - includes.h, 109
- ckb_err_fn
 - includes.h, 109
- ckb_err_nofile
 - includes.h, 109
- ckb_fatal
 - includes.h, 109
- ckb_fatal_fn
 - includes.h, 109
- ckb_fatal_nofile
 - includes.h, 109
- ckb_info
 - includes.h, 109
- ckb_info_fn
 - includes.h, 109
- ckb_info_nofile
 - includes.h, 110
- ckb_s_err
 - includes.h, 110
- ckb_s_out
 - includes.h, 110
- ckb_warn
 - includes.h, 110
- ckb_warn_fn
 - includes.h, 110
- ckb_warn_nofile
 - includes.h, 110
- closeusb
 - usb.c, 297
 - usb.h, 324
- cmd
 - command.h, 39
- cmd_active_kb
 - device.h, 52
 - device_keyboard.c, 59
- cmd_active_mouse
 - device.h, 52
 - device_mouse.c, 63
- cmd_bind
 - input.c, 115
 - input.h, 128
- cmd_dpi
 - dpi.c, 90
 - dpi.h, 95
- cmd_dpisel
 - dpi.c, 91
 - dpi.h, 96
- cmd_erase
 - profile.c, 236
 - profile.h, 252
- cmd_eraseprofile
 - profile.c, 237
 - profile.h, 253
- cmd_fwupdate
 - firmware.c, 101
 - firmware.h, 105
- cmd_get
 - notify.c, 223
 - notify.h, 229
- cmd_hwload_kb
 - profile.h, 253
 - profile_keyboard.c, 265
- cmd_hwload_mouse
 - profile.h, 254
 - profile_mouse.c, 268
- cmd_hwsave_kb
 - profile.h, 255
 - profile_keyboard.c, 266
- cmd_hwsave_mouse
 - profile.h, 256
 - profile_mouse.c, 269
- cmd_iauto
 - led.c, 158
 - led.h, 163
- cmd_id
 - profile.c, 237
 - profile.h, 257
- cmd_idle_kb
 - device.h, 53
 - device_keyboard.c, 60
- cmd_idle_mouse
 - device.h, 53
 - device_mouse.c, 64
- cmd_inotify
 - led.c, 158
 - led.h, 164
- cmd_io_none
 - device_vtable.c, 67
- cmd_ioff
 - led.c, 158
 - led.h, 164
- cmd_ion
 - led.c, 159
 - led.h, 165
- cmd_lift
 - dpi.c, 91
 - dpi.h, 96
- cmd_macro
 - input.c, 115
 - input.h, 129
- cmd_macro_none
 - device_vtable.c, 67
- cmd_name
 - profile.c, 238
 - profile.h, 257
- cmd_none
 - device_vtable.c, 67
- cmd_notify
 - notify.c, 224
 - notify.h, 229
- cmd_pollrate
 - device.h, 54
 - device_mouse.c, 64
- cmd_profileid

- profile.c, 238
- profile.h, 258
- cmd_profilename
 - profile.c, 239
 - profile.h, 258
- cmd_rebind
 - input.c, 115
 - input.h, 129
- cmd_restart
 - notify.c, 224
 - notify.h, 230
- cmd_rgb
 - led.c, 159
 - led.h, 165
- cmd_snap
 - dpi.c, 91
 - dpi.h, 96
- cmd_strings
 - command.c, 36
- cmd_unbind
 - input.c, 115
 - input.h, 129
- cmdhandler
 - command.h, 39
- cmdhandler_io
 - command.h, 39
- cmdhandler_mac
 - command.h, 39
- command.h
 - ACCEL, 39
 - ACTIVE, 39
 - BIND, 40
 - CMD_FIRST, 39
 - CMD_LAST, 40
 - CMD_VT_FIRST, 39
 - DELAY, 39
 - DITHER, 39
 - DPI, 40
 - DPISEL, 40
 - ERASE, 39
 - ERASEPROFILE, 39
 - FPS, 39
 - FWUPDATE, 39
 - GET, 40
 - HWLOAD, 39
 - HWSAVE, 39
 - IAUTO, 40
 - ID, 40
 - IDLE, 39
 - INOTIFY, 40
 - IOFF, 40
 - ION, 40
 - LAYOUT, 39
 - LIFT, 40
 - MACRO, 40
 - MODE, 39
 - NAME, 40
 - NONE, 39
 - NOTIFY, 40
 - NOTIFYOFF, 39
 - NOTIFYON, 39
 - POLLRATE, 39
 - PROFILEID, 40
 - PROFILENAME, 40
 - REBIND, 40
 - RESTART, 40
 - RGB, 40
 - SCROLLSPEED, 39
 - SNAP, 40
 - SWITCH, 39
 - UNBIND, 40
- command.c
 - cmd_strings, 36
 - readcmd, 32
 - TRY_WITH_RESET, 32
- command.h
 - CMD_COUNT, 38
 - CMD_DEV_COUNT, 38
 - cmd, 39
 - cmdhandler, 39
 - cmdhandler_io, 39
 - cmdhandler_mac, 39
 - devcmd, 39
 - readcmd, 41
- corsair_kbcopy
 - keymap.c, 144
 - keymap.h, 153
- corsair_mousecopy
 - keymap.c, 145
 - keymap.h, 153
- DELAY
 - command.h, 39
- DITHER
 - command.h, 39
- DPI
 - command.h, 40
- DPISEL
 - command.h, 40
- DAEMON.md, 31
- DEBUG
 - usb_linux.c, 348
- DELAY_LONG
 - usb.h, 311
- DELAY_MEDIUM
 - usb.h, 311
- DELAY_SHORT
 - usb.h, 311
- DEV_MAX
 - device.h, 51
- DPI_COUNT
 - structures.h, 284
- devcmd, 38
 - command.h, 39
- devcmd.__unnamed__, 27
 - active, 28
 - allocprofile, 28

- bind, 28
- dpi, 28
- dpisel, 28
- erase, 28
- eraseprofile, 28
- freeprofile, 28
- fwupdate, 28
- get, 28
- hwload, 28
- hwsave, 28
- iauto, 29
- id, 29
- idle, 29
- inotify, 29
- ioff, 29
- ion, 29
- lift, 29
- loadprofile, 29
- macro, 29
- name, 29
- notify, 29
- pollrate, 29
- profileid, 29
- profilename, 29
- rebind, 29
- restart, 29
- rgb, 29
- setmodeindex, 29
- snap, 29
- start, 29
- unbind, 29
- updatedpi, 29
- updateindicators, 29
- updatergb, 29
- device.c
 - _start_dev, 46
 - devlistmutex, 48
 - devmutex, 48
 - hwload_mode, 48
 - inputmutex, 48
 - keyboard, 48
 - macromutex, 48
 - macromutex2, 48
 - macrovar, 49
 - start_dev, 47
- device.h
 - ACT_LIGHT, 50
 - ACT_LOCK, 50
 - ACT_M1, 50
 - ACT_M2, 50
 - ACT_M3, 51
 - ACT_MR_RING, 51
 - ACT_NEXT, 51
 - ACT_NEXT_NOWRAP, 51
 - cmd_active_kb, 52
 - cmd_active_mouse, 52
 - cmd_idle_kb, 53
 - cmd_idle_mouse, 53
 - cmd_pollrate, 54
 - DEV_MAX, 51
 - devmutex, 58
 - dmutex, 51
 - IN_CORSAIR, 51
 - IN_HID, 51
 - IS_CONNECTED, 51
 - imutex, 51
 - inputmutex, 58
 - keyboard, 58
 - macromutex, 58
 - macromutex2, 59
 - macrovar, 59
 - mmutex, 52
 - mmutex2, 52
 - mvar, 52
 - setactive, 52
 - setactive_kb, 54
 - setactive_mouse, 56
 - setmodeindex_nrgb, 57
 - start_dev, 57
 - start_kb_nrgb, 58
- device_keyboard.c
 - cmd_active_kb, 59
 - cmd_idle_kb, 60
 - setactive_kb, 60
 - setmodeindex_nrgb, 62
 - start_kb_nrgb, 62
- device_mouse.c
 - cmd_active_mouse, 63
 - cmd_idle_mouse, 64
 - cmd_pollrate, 64
 - setactive_mouse, 64
- device_vtable.c
 - cmd_io_none, 67
 - cmd_macro_none, 67
 - cmd_none, 67
 - int1_int_none, 67
 - int1_void_none, 67
 - loadprofile_none, 67
 - vtable_keyboard, 68
 - vtable_keyboard_nonrgb, 68
 - vtable_mouse, 68
- devlistmutex
 - device.c, 48
- devmain
 - usb.c, 299
- devmutex
 - device.c, 48
 - device.h, 58
- devnode.c
 - _mkdevpath, 70
 - _mknotifynode, 73
 - _rmnotifynode, 74
 - _updateconnected, 74
 - devpath, 82
 - gid, 82
 - MAX_BUFFER, 70

- mkdevpath, [75](#)
- mkfwnode, [76](#)
- mknotifynode, [77](#)
- readlines, [77](#)
- readlines_ctx_free, [78](#)
- readlines_ctx_init, [79](#)
- rm_recursive, [79](#)
- rmdevpath, [79](#)
- rmnotifynode, [80](#)
- S_GID_READ, [70](#)
- updateconnected, [81](#)
- devnode.h
 - devpath, [90](#)
 - gid, [90](#)
 - mkdevpath, [84](#)
 - mkfwnode, [84](#)
 - mknotifynode, [85](#)
 - readlines, [86](#)
 - readlines_ctx, [84](#)
 - readlines_ctx_free, [87](#)
 - readlines_ctx_init, [87](#)
 - rmdevpath, [87](#)
 - rmnotifynode, [88](#)
 - S_CUSTOM, [83](#)
 - S_CUSTOM_R, [83](#)
 - S_READ, [83](#)
 - S_READDIR, [83](#)
 - S_READWRITE, [83](#)
 - updateconnected, [89](#)
- devpath
 - devnode.c, [82](#)
 - devnode.h, [90](#)
- dmutex
 - device.h, [51](#)
- dpi
 - devcmd.__unnamed__, [28](#)
- dpi.c
 - cmd_dpi, [90](#)
 - cmd_dpisel, [91](#)
 - cmd_lift, [91](#)
 - cmd_snap, [91](#)
 - loaddpi, [92](#)
 - printdpi, [93](#)
 - savedpi, [93](#)
 - updatedpi, [94](#)
- dpi.h
 - cmd_dpi, [95](#)
 - cmd_dpisel, [96](#)
 - cmd_lift, [96](#)
 - cmd_snap, [96](#)
 - loaddpi, [97](#)
 - printdpi, [98](#)
 - savedpi, [98](#)
 - updatedpi, [99](#)
- dpisel
 - devcmd.__unnamed__, [28](#)
- dpiset, [276](#)
- ERASE
 - command.h, [39](#)
- ERASEPROFILE
 - command.h, [39](#)
- erase
 - devcmd.__unnamed__, [28](#)
- eraseprofile
 - devcmd.__unnamed__, [28](#)
- euid_guard_start
 - os.h, [233](#)
- euid_guard_stop
 - os.h, [233](#)
- FPS
 - command.h, [39](#)
- FWUPDATE
 - command.h, [39](#)
- FEAT_ADJRATE
 - structures.h, [284](#)
- FEAT_ANSI
 - structures.h, [284](#)
- FEAT_BIND
 - structures.h, [284](#)
- FEAT_COMMON
 - structures.h, [284](#)
- FEAT_FWUPDATE
 - structures.h, [284](#)
- FEAT_FWVERSION
 - structures.h, [284](#)
- FEAT_HWLOAD
 - structures.h, [285](#)
- FEAT_ISO
 - structures.h, [285](#)
- FEAT_LMASK
 - structures.h, [285](#)
- FEAT_MONOCHROME
 - structures.h, [285](#)
- FEAT_MOUSEACCEL
 - structures.h, [285](#)
- FEAT_NOTIFY
 - structures.h, [285](#)
- FEAT_POLLRATE
 - structures.h, [285](#)
- FEAT_RGB
 - structures.h, [285](#)
- FEAT_STD_NRGB
 - structures.h, [285](#)
- FEAT_STD_RGB
 - structures.h, [286](#)
- FW_MAXSIZE
 - firmware.c, [101](#)
- FW_NOFILE
 - firmware.c, [101](#)
- FW_OK
 - firmware.c, [101](#)
- FW_USBFAIL
 - firmware.c, [101](#)
- FW_WRONGDEV
 - firmware.c, [101](#)
- features_mask

- main.c, [219](#)
- usb.c, [306](#)
- firmware.c
 - cmd_fwupdate, [101](#)
 - FW_MAXSIZE, [101](#)
 - FW_NOFILE, [101](#)
 - FW_OK, [101](#)
 - FW_USBFAIL, [101](#)
 - FW_WRONGDEV, [101](#)
 - fwupdate, [102](#)
 - getfwversion, [104](#)
- firmware.h
 - cmd_fwupdate, [105](#)
 - getfwversion, [106](#)
- freebind
 - input.c, [116](#)
 - input.h, [130](#)
- freemode
 - profile.c, [239](#)
- freeprofile
 - devcmd.__unnamed__, [28](#)
 - profile.c, [240](#)
 - profile.h, [259](#)
- fwupdate
 - devcmd.__unnamed__, [28](#)
 - firmware.c, [102](#)
- GET
 - command.h, [40](#)
- get
 - devcmd.__unnamed__, [28](#)
- get_vtable
 - usb.c, [301](#)
- getfwversion
 - firmware.c, [104](#)
 - firmware.h, [106](#)
- gethwmodename
 - profile.c, [240](#)
 - profile.h, [259](#)
- gethwprofilename
 - profile.c, [241](#)
 - profile.h, [260](#)
- getid
 - profile.c, [242](#)
 - profile.h, [261](#)
- getmodename
 - profile.c, [242](#)
 - profile.h, [261](#)
- getprofilename
 - profile.c, [243](#)
 - profile.h, [262](#)
- gid
 - devnode.c, [82](#)
 - devnode.h, [90](#)
- HWLOAD
 - command.h, [39](#)
- HWSAVE
 - command.h, [39](#)
- HAS_ANY_FEATURE
 - structures.h, [286](#)
- HAS_FEATURES
 - structures.h, [286](#)
- HW_STANDARD
 - notify.c, [220](#)
- HWMODE_K70
 - structures.h, [286](#)
- HWMODE_K95
 - structures.h, [286](#)
- HWMODE_MAX
 - structures.h, [286](#)
- HWMODE_OR_RETURN
 - notify.c, [220](#)
- has_key
 - led.c, [160](#)
- hid_kb_translate
 - keymap.c, [145](#)
 - keymap.h, [153](#)
- hid_mouse_translate
 - keymap.c, [147](#)
 - keymap.h, [155](#)
- hwload
 - devcmd.__unnamed__, [28](#)
- hwload_mode
 - device.c, [48](#)
 - main.c, [219](#)
 - usb.c, [306](#)
- hwloadmode
 - profile_keyboard.c, [267](#)
- hwloadprofile
 - profile.h, [251](#)
- hwprofile, [280](#)
- hwsave
 - devcmd.__unnamed__, [28](#)
- hwtonative
 - profile.c, [244](#)
 - profile.h, [263](#)
- IAUTO
 - command.h, [40](#)
- ID
 - command.h, [40](#)
- IDLE
 - command.h, [39](#)
- INOTIFY
 - command.h, [40](#)
- IOFF
 - command.h, [40](#)
- ION
 - command.h, [40](#)
- I_CAPS
 - structures.h, [286](#)
- I_NUM
 - structures.h, [286](#)
- I_SCROLL
 - structures.h, [286](#)
- IFACE_MAX
 - structures.h, [286](#)

- IN_CORSAIR
 - device.h, [51](#)
- IN_HID
 - device.h, [51](#)
- INDEX_OF
 - includes.h, [110](#)
- IS_CONNECTED
 - device.h, [51](#)
- IS_FULLRANGE
 - usb.h, [311](#)
- IS_K65
 - usb.h, [311](#)
- IS_K70
 - usb.h, [311](#)
- IS_K95
 - usb.h, [311](#)
- IS_M65
 - usb.h, [312](#)
- IS_MOD
 - input.h, [128](#)
- IS_MONOCHROME
 - usb.h, [312](#)
- IS_MONOCHROME_DEV
 - usb.h, [312](#)
- IS_MOUSE
 - usb.h, [312](#)
- IS_MOUSE_DEV
 - usb.h, [312](#)
- IS_RGB
 - usb.h, [312](#)
- IS_RGB_DEV
 - usb.h, [312](#)
- IS_SABRE
 - usb.h, [312](#)
- IS_SCIMITAR
 - usb.h, [312](#)
- IS_STRAFE
 - usb.h, [313](#)
- IS_WHEEL
 - input.c, [113](#)
- iauto
 - devcmd.__unnamed__, [29](#)
- id
 - devcmd.__unnamed__, [29](#)
- idle
 - devcmd.__unnamed__, [29](#)
- imutex
 - device.h, [51](#)
- includes.h
 - __FILE_NOPATH__, [109](#)
 - ckb_err, [109](#)
 - ckb_err_fn, [109](#)
 - ckb_err_nofile, [109](#)
 - ckb_fatal, [109](#)
 - ckb_fatal_fn, [109](#)
 - ckb_fatal_nofile, [109](#)
 - ckb_info, [109](#)
 - ckb_info_fn, [109](#)
 - ckb_info_nofile, [110](#)
 - ckb_s_err, [110](#)
 - ckb_s_out, [110](#)
 - ckb_warn, [110](#)
 - ckb_warn_fn, [110](#)
 - ckb_warn_nofile, [110](#)
 - INDEX_OF, [110](#)
 - timespec_add, [111](#)
 - timespec_eq, [110](#)
 - timespec_ge, [110](#)
 - timespec_gt, [111](#)
 - timespec_le, [111](#)
 - timespec_lt, [111](#)
 - uchar, [111](#)
 - ushort, [111](#)
- initbind
 - input.c, [116](#)
 - input.h, [130](#)
- initmode
 - profile.c, [244](#)
- inotify
 - devcmd.__unnamed__, [29](#)
- input.c
 - _cmd_macro, [113](#)
 - cmd_bind, [115](#)
 - cmd_macro, [115](#)
 - cmd_rebind, [115](#)
 - cmd_unbind, [115](#)
 - freebind, [116](#)
 - IS_WHEEL, [113](#)
 - initbind, [116](#)
 - inputupdate, [117](#)
 - inputupdate_keys, [118](#)
 - macro_pt_dequeue, [120](#)
 - macro_pt_enqueue, [121](#)
 - macro_pt_first, [121](#)
 - macromask, [122](#)
 - play_macro, [122](#)
 - pt_head, [125](#)
 - pt_tail, [125](#)
 - updateindicators_kb, [124](#)
- input.h
 - cmd_bind, [128](#)
 - cmd_macro, [129](#)
 - cmd_rebind, [129](#)
 - cmd_unbind, [129](#)
 - freebind, [130](#)
 - IS_MOD, [128](#)
 - initbind, [130](#)
 - inputupdate, [131](#)
 - os_inputclose, [132](#)
 - os_inputopen, [132](#)
 - os_keypress, [133](#)
 - os_mousemove, [134](#)
 - os_setupindicators, [135](#)
 - parameter_t, [128](#)
 - ptlist_t, [128](#)
 - updateindicators_kb, [136](#)

- input_linux.c
 - _ledthread, 137
 - isync, 138
 - os_inputclose, 138
 - os_inputopen, 139
 - os_keypress, 140
 - os_mousemove, 140
 - os_setupindicators, 141
 - uinputopen, 142
- inputmutex
 - device.c, 48
 - device.h, 58
- inputupdate
 - input.c, 117
 - input.h, 131
- inputupdate_keys
 - input.c, 118
- int1_int_none
 - device_vtable.c, 67
- int1_void_none
 - device_vtable.c, 67
- ioff
 - devcmd.__unnamed__, 29
- ion
 - devcmd.__unnamed__, 29
- isblack
 - led_mouse.c, 207
- iselect
 - led.c, 160
- isync
 - input_linux.c, 138
- KB_NAME_LEN
 - structures.h, 287
- KEY_BACKSLASH_ISO
 - keymap.h, 150
- KEY_CORSAIR
 - keymap.h, 150
- KEY_NONE
 - keymap.h, 150
- KEY_UNBOUND
 - keymap.h, 151
- kbsyspath
 - usb_linux.c, 371
- key, 149
- keyboard
 - device.c, 48
 - device.h, 58
- keymacro, 274
- keymap
 - keymap.c, 148
 - keymap.h, 156
- keymap.c
 - BUTTON_HID_COUNT, 144
 - corsair_kbcopy, 144
 - corsair_mousecopy, 145
 - hid_kb_translate, 145
 - hid_mouse_translate, 147
 - keymap, 148
- keymap.h
 - BTN_WHEELDOWN, 150
 - BTN_WHEELUP, 150
 - corsair_kbcopy, 153
 - corsair_mousecopy, 153
 - hid_kb_translate, 153
 - hid_mouse_translate, 155
 - KEY_BACKSLASH_ISO, 150
 - KEY_CORSAIR, 150
 - KEY_NONE, 150
 - KEY_UNBOUND, 151
 - keymap, 156
 - LED_DPI, 151
 - LED_MOUSE, 151
 - MOUSE_BUTTON_FIRST, 151
 - MOUSE_EXTRA_FIRST, 151
 - N_BUTTONS_EXTENDED, 151
 - N_BUTTONS_HW, 151
 - N_KEY_ZONES, 151
 - N_KEYBYTES_EXTENDED, 151
 - N_KEYBYTES_HW, 151
 - N_KEYBYTES_INPUT, 152
 - N_KEYS_EXTENDED, 152
 - N_KEYS_EXTRA, 152
 - N_KEYS_HW, 152
 - N_KEYS_INPUT, 152
 - N_MOUSE_ZONES, 152
 - SCAN_KBD, 152
 - SCAN_MOUSE, 152
 - SCAN_SILENT, 152
- LAYOUT
 - command.h, 39
- LIFT
 - command.h, 40
- LED_DPI
 - keymap.h, 151
- LED_MOUSE
 - keymap.h, 151
- LIFT_MAX
 - structures.h, 287
- LIFT_MIN
 - structures.h, 287
- led.c
 - cmd_iauto, 158
 - cmd_inotify, 158
 - cmd_ioff, 158
 - cmd_ion, 159
 - cmd_rgb, 159
 - has_key, 160
 - iselect, 160
 - printrgb, 161
- led.h
 - cmd_iauto, 163
 - cmd_inotify, 164
 - cmd_ioff, 164
 - cmd_ion, 165
 - cmd_rgb, 165
 - loadrgb_kb, 165

- loadrgb_mouse, [167](#)
- printrgb, [168](#)
- savergb_kb, [170](#)
- savergb_mouse, [171](#)
- updatergb_kb, [171](#)
- updatergb_mouse, [173](#)
- led_keyboard.c
 - BR1, [197](#)
 - BR2, [197](#)
 - BR4, [197](#)
 - bit_reverse_table, [205](#)
 - loadrgb_kb, [198](#)
 - makergb_512, [199](#)
 - makergb_full, [200](#)
 - O0, [197](#)
 - O1, [197](#)
 - O2, [197](#)
 - O3, [197](#)
 - O4, [197](#)
 - O5, [197](#)
 - O6, [197](#)
 - O7, [197](#)
 - O8, [197](#)
 - ordered8to3, [201](#)
 - quantize8to3, [201](#)
 - rgbcmp, [202](#)
 - savergb_kb, [202](#)
 - updatergb_kb, [203](#)
- led_mouse.c
 - isblack, [207](#)
 - loadrgb_mouse, [208](#)
 - rgbcmp, [208](#)
 - savergb_mouse, [209](#)
 - updatergb_mouse, [209](#)
- lift
 - devcmd.__unnamed__, [29](#)
- lighting, [277](#)
- loaddpi
 - dpi.c, [92](#)
 - dpi.h, [97](#)
- loadprofile
 - devcmd.__unnamed__, [29](#)
 - profile.c, [245](#)
 - profile.h, [263](#)
- loadprofile_none
 - device_vtable.c, [67](#)
- loadrgb_kb
 - led.h, [165](#)
 - led_keyboard.c, [198](#)
- loadrgb_mouse
 - led.h, [167](#)
 - led_mouse.c, [208](#)
- localecase
 - main.c, [211](#)
- MACRO
 - command.h, [40](#)
- MODE
 - command.h, [39](#)
- MACRO_MAX
 - structures.h, [287](#)
- MAX_BUFFER
 - devnode.c, [70](#)
- MD_NAME_LEN
 - structures.h, [287](#)
- MODE_COUNT
 - structures.h, [287](#)
- MOUSE_BUTTON_FIRST
 - keymap.h, [151](#)
- MOUSE_EXTRA_FIRST
 - keymap.h, [151](#)
- MSG_SIZE
 - structures.h, [287](#)
- macro
 - devcmd.__unnamed__, [29](#)
- macro_pt_dequeue
 - input.c, [120](#)
- macro_pt_enqueue
 - input.c, [121](#)
- macro_pt_first
 - input.c, [121](#)
- macroaction, [273](#)
- macromask
 - input.c, [122](#)
- macromutex
 - device.c, [48](#)
 - device.h, [58](#)
- macromutex2
 - device.c, [48](#)
 - device.h, [59](#)
- macrovar
 - device.c, [49](#)
 - device.h, [59](#)
- main
 - main.c, [212](#)
- main.c
 - features_mask, [219](#)
 - hwload_mode, [219](#)
 - localecase, [211](#)
 - main, [212](#)
 - main_ac, [219](#)
 - main_av, [219](#)
 - quit, [214](#)
 - quitWithLock, [215](#)
 - reset_stop, [219](#)
 - restart, [216](#)
 - sighandler, [217](#)
 - sighandler2, [218](#)
 - timespec_add, [218](#)
- main_ac
 - main.c, [219](#)
- main_av
 - main.c, [219](#)
- makergb_512
 - led_keyboard.c, [199](#)
- makergb_full
 - led_keyboard.c, [200](#)

- mkdevpath
 - devnode.c, [75](#)
 - devnode.h, [84](#)
- mkfwnode
 - devnode.c, [76](#)
 - devnode.h, [84](#)
- mknotifynode
 - devnode.c, [77](#)
 - devnode.h, [85](#)
- mmutex
 - device.h, [52](#)
- mmutex2
 - device.h, [52](#)
- models
 - usb_linux.c, [371](#)
- mvar
 - device.h, [52](#)
- NAME
 - command.h, [40](#)
- NONE
 - command.h, [39](#)
- NOTIFY
 - command.h, [40](#)
- NOTIFYOFF
 - command.h, [39](#)
- NOTIFYON
 - command.h, [39](#)
- N_BUTTONS_EXTENDED
 - keymap.h, [151](#)
- N_BUTTONS_HW
 - keymap.h, [151](#)
- N_KEY_ZONES
 - keymap.h, [151](#)
- N_KEYBYTES_EXTENDED
 - keymap.h, [151](#)
- N_KEYBYTES_HW
 - keymap.h, [151](#)
- N_KEYBYTES_INPUT
 - keymap.h, [152](#)
- N_KEYS_EXTENDED
 - keymap.h, [152](#)
- N_KEYS_EXTRA
 - keymap.h, [152](#)
- N_KEYS_HW
 - keymap.h, [152](#)
- N_KEYS_INPUT
 - keymap.h, [152](#)
- N_MODELS
 - usb_linux.c, [348](#)
- N_MOUSE_ZONES
 - keymap.h, [152](#)
- NEEDS_FW_UPDATE
 - structures.h, [287](#)
- NK95_HWOFF
 - usb.h, [313](#)
- NK95_HWON
 - usb.h, [313](#)
- NK95_M1
 - usb.h, [313](#)
- NK95_M2
 - usb.h, [313](#)
- NK95_M3
 - usb.h, [313](#)
- name
 - devcmd.__unnamed__, [29](#)
- nativetohw
 - profile.c, [245](#)
 - profile.h, [264](#)
- nk95cmd
 - usb.h, [313](#)
- notify
 - devcmd.__unnamed__, [29](#)
- notify.c
 - _cmd_get, [221](#)
 - cmd_get, [223](#)
 - cmd_notify, [224](#)
 - cmd_restart, [224](#)
 - HW_STANDARD, [220](#)
 - HWMODE_OR_RETURN, [220](#)
 - nprintf, [225](#)
 - nprintind, [225](#)
 - nprintkey, [226](#)
 - restart, [227](#)
- notify.h
 - cmd_get, [229](#)
 - cmd_notify, [229](#)
 - cmd_restart, [230](#)
 - nprintf, [230](#)
 - nprintind, [231](#)
 - nprintkey, [232](#)
- nprintf
 - notify.c, [225](#)
 - notify.h, [230](#)
- nprintind
 - notify.c, [225](#)
 - notify.h, [231](#)
- nprintkey
 - notify.c, [226](#)
 - notify.h, [232](#)
- O0
 - led_keyboard.c, [197](#)
- O1
 - led_keyboard.c, [197](#)
- O2
 - led_keyboard.c, [197](#)
- O3
 - led_keyboard.c, [197](#)
- O4
 - led_keyboard.c, [197](#)
- O5
 - led_keyboard.c, [197](#)
- O6
 - led_keyboard.c, [197](#)
- O7
 - led_keyboard.c, [197](#)
- O8
 - led_keyboard.c, [197](#)

- led_keyboard.c, [197](#)
- OUTFIFO_MAX
 - structures.h, [287](#)
- ordered8to3
 - led_keyboard.c, [201](#)
- os.h
 - _DEFAULT_SOURCE, [233](#)
 - _GNU_SOURCE, [233](#)
 - euid_guard_start, [233](#)
 - euid_guard_stop, [233](#)
 - UINPUT_VERSION, [233](#)
- os_closeusb
 - usb.h, [326](#)
 - usb_linux.c, [350](#)
- os_inputclose
 - input.h, [132](#)
 - input_linux.c, [138](#)
- os_inputmain
 - usb.h, [327](#)
 - usb_linux.c, [351](#)
- os_inputopen
 - input.h, [132](#)
 - input_linux.c, [139](#)
- os_keypress
 - input.h, [133](#)
 - input_linux.c, [140](#)
- os_mousemove
 - input.h, [134](#)
 - input_linux.c, [140](#)
- os_resetusb
 - usb.h, [330](#)
 - usb_linux.c, [354](#)
- os_sendindicators
 - usb.h, [331](#)
 - usb_linux.c, [355](#)
- os_setupindicators
 - input.h, [135](#)
 - input_linux.c, [141](#)
- os_setupusb
 - usb.h, [333](#)
 - usb_linux.c, [356](#)
- os_usbrecv
 - usb.h, [335](#)
 - usb_linux.c, [358](#)
- os_usbsend
 - usb.h, [337](#)
 - usb_linux.c, [359](#)
- POLLRATE
 - command.h, [39](#)
- PROFILEID
 - command.h, [40](#)
- PROFILENAME
 - command.h, [40](#)
- P_K65
 - usb.h, [313](#)
- P_K65_LUX
 - usb.h, [314](#)
- P_K65_LUX_STR
 - usb.h, [314](#)
- P_K65_NRGB
 - usb.h, [314](#)
- P_K65_NRGB_STR
 - usb.h, [314](#)
- P_K65_RFIRE
 - usb.h, [314](#)
- P_K65_RFIRE_STR
 - usb.h, [314](#)
- P_K65_STR
 - usb.h, [314](#)
- P_K70
 - usb.h, [314](#)
- P_K70_LUX
 - usb.h, [314](#)
- P_K70_LUX_NRGB
 - usb.h, [314](#)
- P_K70_LUX_NRGB_STR
 - usb.h, [315](#)
- P_K70_LUX_STR
 - usb.h, [315](#)
- P_K70_NRGB
 - usb.h, [315](#)
- P_K70_NRGB_STR
 - usb.h, [315](#)
- P_K70_RFIRE
 - usb.h, [315](#)
- P_K70_RFIRE_NRGB
 - usb.h, [315](#)
- P_K70_RFIRE_NRGB_STR
 - usb.h, [315](#)
- P_K70_RFIRE_STR
 - usb.h, [315](#)
- P_K70_STR
 - usb.h, [315](#)
- P_K95
 - usb.h, [315](#)
- P_K95_NRGB
 - usb.h, [315](#)
- P_K95_NRGB_STR
 - usb.h, [316](#)
- P_K95_PLATINUM
 - usb.h, [316](#)
- P_K95_PLATINUM_STR
 - usb.h, [316](#)
- P_K95_STR
 - usb.h, [316](#)
- P_M65
 - usb.h, [316](#)
- P_M65_PRO
 - usb.h, [316](#)
- P_M65_PRO_STR
 - usb.h, [316](#)
- P_M65_STR
 - usb.h, [316](#)
- P_SABRE_L
 - usb.h, [316](#)
- P_SABRE_L_STR
 - usb.h, [316](#)

- usb.h, 316
- P_SABRE_N
 - usb.h, 316
- P_SABRE_N_STR
 - usb.h, 317
- P_SABRE_O
 - usb.h, 317
- P_SABRE_O2
 - usb.h, 317
- P_SABRE_O2_STR
 - usb.h, 317
- P_SABRE_O_STR
 - usb.h, 317
- P_SCIMITAR
 - usb.h, 317
- P_SCIMITAR_PRO
 - usb.h, 317
- P_SCIMITAR_PRO_STR
 - usb.h, 317
- P_SCIMITAR_STR
 - usb.h, 317
- P_STRAFE
 - usb.h, 317
- P_STRAFE_NRGB
 - usb.h, 318
- P_STRAFE_NRGB_STR
 - usb.h, 318
- P_STRAFE_STR
 - usb.h, 318
- PR_NAME_LEN
 - structures.h, 288
- parameter, 127
- parameter_t
 - input.h, 128
- play_macro
 - input.c, 122
- pollrate
 - devcmd.__unnamed__, 29
- printdpi
 - dpi.c, 93
 - dpi.h, 98
- printname
 - profile.c, 246
- printrgb
 - led.c, 161
 - led.h, 168
- product_str
 - usb.c, 302
 - usb.h, 339
- profile.c
 - _freeprofile, 235
 - allocprofile, 235
 - cmd_erase, 236
 - cmd_eraseprofile, 237
 - cmd_id, 237
 - cmd_name, 238
 - cmd_profileid, 238
 - cmd_profilename, 239
- freemode, 239
- freeprofile, 240
- gethwmodename, 240
- gethwprofilename, 241
- getid, 242
- getmodename, 242
- getprofilename, 243
- hwtonative, 244
- initmode, 244
- loadprofile, 245
- nativetohw, 245
- printname, 246
- setid, 247
- u16dec, 247
- u16enc, 248
- urldecode2, 248
- urlencode2, 249
- utf16to8, 250
- utf8to16, 250
- profile.h
 - allocprofile, 251
 - cmd_erase, 252
 - cmd_eraseprofile, 253
 - cmd_hwload_kb, 253
 - cmd_hwload_mouse, 254
 - cmd_hwsave_kb, 255
 - cmd_hwsave_mouse, 256
 - cmd_id, 257
 - cmd_name, 257
 - cmd_profileid, 258
 - cmd_profilename, 258
 - freeprofile, 259
 - gethwmodename, 259
 - gethwprofilename, 260
 - getid, 261
 - getmodename, 261
 - getprofilename, 262
 - hwloadprofile, 251
 - hwtonative, 263
 - loadprofile, 263
 - nativetohw, 264
 - setid, 264
- profile_keyboard.c
 - cmd_hwload_kb, 265
 - cmd_hwsave_kb, 266
 - hwloadmode, 267
- profile_mouse.c
 - cmd_hwload_mouse, 268
 - cmd_hwsave_mouse, 269
- profileid
 - devcmd.__unnamed__, 29
- profilename
 - devcmd.__unnamed__, 29
- pt_head
 - input.c, 125
- pt_tail
 - input.c, 125
- ptlist, 128

- ptlist_t
 - input.h, [128](#)
- quantize8to3
 - led_keyboard.c, [201](#)
- quit
 - main.c, [214](#)
- quitWithLock
 - main.c, [215](#)
- REBIND
 - command.h, [40](#)
- RESTART
 - command.h, [40](#)
- RGB
 - command.h, [40](#)
- README.md, [31](#)
- ROADMAP.md, [31](#)
- readcmd
 - command.c, [32](#)
 - command.h, [41](#)
- readlines
 - devnode.c, [77](#)
 - devnode.h, [86](#)
- readlines_ctx
 - devnode.h, [84](#)
- readlines_ctx_free
 - devnode.c, [78](#)
 - devnode.h, [87](#)
- readlines_ctx_init
 - devnode.c, [79](#)
 - devnode.h, [87](#)
- rebind
 - devcmd.__unnamed__, [29](#)
- reset_stop
 - main.c, [219](#)
 - usb.c, [306](#)
- resetusb
 - usb.h, [318](#)
- restart
 - devcmd.__unnamed__, [29](#)
 - main.c, [216](#)
 - notify.c, [227](#)
- revertusb
 - usb.c, [303](#)
 - usb.h, [340](#)
- rgb
 - devcmd.__unnamed__, [29](#)
- rgbcmp
 - led_keyboard.c, [202](#)
 - led_mouse.c, [208](#)
- rm_recursive
 - devnode.c, [79](#)
- rmdevpath
 - devnode.c, [79](#)
 - devnode.h, [87](#)
- rmnotifynode
 - devnode.c, [80](#)
 - devnode.h, [88](#)
- SCROLLSPEED
 - command.h, [39](#)
- SNAP
 - command.h, [40](#)
- SWITCH
 - command.h, [39](#)
- S_CUSTOM
 - devnode.h, [83](#)
- S_CUSTOM_R
 - devnode.h, [83](#)
- S_GID_READ
 - devnode.c, [70](#)
- S_READ
 - devnode.h, [83](#)
- S_READDIR
 - devnode.h, [83](#)
- S_READWRITE
 - devnode.h, [83](#)
- SCAN_KBD
 - keymap.h, [152](#)
- SCAN_MOUSE
 - keymap.h, [152](#)
- SCAN_SILENT
 - keymap.h, [152](#)
- SCROLL_ACCELERATED
 - structures.h, [288](#)
- SCROLL_MAX
 - structures.h, [288](#)
- SCROLL_MIN
 - structures.h, [288](#)
- SERIAL_LEN
 - structures.h, [288](#)
- SET_KEYBIT
 - structures.h, [288](#)
- savedpi
 - dpi.c, [93](#)
 - dpi.h, [98](#)
- savergb_kb
 - led.h, [170](#)
 - led_keyboard.c, [202](#)
- savergb_mouse
 - led.h, [171](#)
 - led_mouse.c, [209](#)
- setactive
 - device.h, [52](#)
- setactive_kb
 - device.h, [54](#)
 - device_keyboard.c, [60](#)
- setactive_mouse
 - device.h, [56](#)
 - device_mouse.c, [64](#)
- setid
 - profile.c, [247](#)
 - profile.h, [264](#)
- setmodeindex
 - devcmd.__unnamed__, [29](#)
- setmodeindex_nrgb
 - device.h, [57](#)

- device_keyboard.c, 62
- setupusb
 - usb.c, 304
 - usb.h, 341
- sighandler
 - main.c, 217
- sighandler2
 - main.c, 218
- snap
 - devcmd.__unnamed__, 29
- src/ckb-daemon/command.c, 31
- src/ckb-daemon/command.h, 36
- src/ckb-daemon/device.c, 45
- src/ckb-daemon/device.h, 49
- src/ckb-daemon/device_keyboard.c, 59
- src/ckb-daemon/device_mouse.c, 63
- src/ckb-daemon/device_vtable.c, 66
- src/ckb-daemon/devnode.c, 68
- src/ckb-daemon/devnode.h, 82
- src/ckb-daemon/dpi.c, 90
- src/ckb-daemon/dpi.h, 95
- src/ckb-daemon/extra_mac.c, 100
- src/ckb-daemon/firmware.c, 100
- src/ckb-daemon/firmware.h, 105
- src/ckb-daemon/includes.h, 107
- src/ckb-daemon/input.c, 111
- src/ckb-daemon/input.h, 125
- src/ckb-daemon/input_linux.c, 136
- src/ckb-daemon/input_mac.c, 143
- src/ckb-daemon/input_mac_mouse.c, 143
- src/ckb-daemon/keymap.c, 143
- src/ckb-daemon/keymap.h, 148
- src/ckb-daemon/keymap_mac.h, 156
- src/ckb-daemon/led.c, 157
- src/ckb-daemon/led.h, 163
- src/ckb-daemon/led_keyboard.c, 174
- src/ckb-daemon/led_mouse.c, 207
- src/ckb-daemon/main.c, 210
- src/ckb-daemon/notify.c, 219
- src/ckb-daemon/notify.h, 228
- src/ckb-daemon/os.h, 232
- src/ckb-daemon/profile.c, 234
- src/ckb-daemon/profile.h, 250
- src/ckb-daemon/profile_keyboard.c, 265
- src/ckb-daemon/profile_mouse.c, 268
- src/ckb-daemon/structures.h, 270
- src/ckb-daemon/usb.c, 289
- src/ckb-daemon/usb.h, 307
- src/ckb-daemon/usb_linux.c, 346
- src/ckb-daemon/usb_mac.c, 372
- start
 - devcmd.__unnamed__, 29
- start_dev
 - device.c, 47
 - device.h, 57
- start_kb_nrgb
 - device.h, 58
 - device_keyboard.c, 62
- strtrim
 - usb_linux.c, 361
- structures.h
 - CLEAR_KEYBIT, 284
 - DPI_COUNT, 284
 - FEAT_ADJRATE, 284
 - FEAT_ANSI, 284
 - FEAT_BIND, 284
 - FEAT_COMMON, 284
 - FEAT_FWUPDATE, 284
 - FEAT_FWVERSION, 284
 - FEAT_HWLOAD, 285
 - FEAT_ISO, 285
 - FEAT_LMASK, 285
 - FEAT_MONOCHROME, 285
 - FEAT_MOUSEACCEL, 285
 - FEAT_NOTIFY, 285
 - FEAT_POLLRATE, 285
 - FEAT_RGB, 285
 - FEAT_STD_NRGB, 285
 - FEAT_STD_RGB, 286
 - HAS_ANY_FEATURE, 286
 - HAS_FEATURES, 286
 - HWMODE_K70, 286
 - HWMODE_K95, 286
 - HWMODE_MAX, 286
 - I_CAPS, 286
 - I_NUM, 286
 - I_SCROLL, 286
 - IFACE_MAX, 286
 - KB_NAME_LEN, 287
 - LIFT_MAX, 287
 - LIFT_MIN, 287
 - MACRO_MAX, 287
 - MD_NAME_LEN, 287
 - MODE_COUNT, 287
 - MSG_SIZE, 287
 - NEEDS_FW_UPDATE, 287
 - OUTFIFO_MAX, 287
 - PR_NAME_LEN, 288
 - SCROLL_ACCELERATED, 288
 - SCROLL_MAX, 288
 - SCROLL_MIN, 288
 - SERIAL_LEN, 288
 - SET_KEYBIT, 288
 - vtable_keyboard, 288
 - vtable_keyboard_nonrgb, 288
 - vtable_mouse, 288
- TEST_RESET
 - usb_linux.c, 348
- TRY_WITH_RESET
 - command.c, 32
- timespec_add
 - includes.h, 111
 - main.c, 218
- timespec_eq
 - includes.h, 110
- timespec_ge

- includes.h, 110
- timespec_gt
 - includes.h, 111
- timespec_le
 - includes.h, 111
- timespec_lt
 - includes.h, 111
- u16dec
 - profile.c, 247
- u16enc
 - profile.c, 248
- UNBIND
 - command.h, 40
- UINPUT_VERSION
 - os.h, 233
- USB_DELAY_DEFAULT
 - usb.h, 318
- uchar
 - includes.h, 111
- udev
 - usb_linux.c, 372
- udev_enum
 - usb_linux.c, 361
- udevthread
 - usb_linux.c, 372
- uinputopen
 - input_linux.c, 142
- unbind
 - devcmd.__unnamed__, 29
- updateconnected
 - devnode.c, 81
 - devnode.h, 89
- updatedpi
 - devcmd.__unnamed__, 29
 - dpi.c, 94
 - dpi.h, 99
- updateindicators
 - devcmd.__unnamed__, 29
- updateindicators_kb
 - input.c, 124
 - input.h, 136
- updatergb
 - devcmd.__unnamed__, 29
- updatergb_kb
 - led.h, 171
 - led_keyboard.c, 203
- updatergb_mouse
 - led.h, 173
 - led_mouse.c, 209
- urldecode2
 - profile.c, 248
- urlencode2
 - profile.c, 249
- usb.c
 - _resetusb, 290
 - _setupusb, 290
 - _usbrecv, 294
 - _usbsend, 296
- closeusb, 297
- devmain, 299
- features_mask, 306
- get_vtable, 301
- hwload_mode, 306
- product_str, 302
- reset_stop, 306
- revertusb, 303
- setupusb, 304
- usb_tryreset, 305
- usbmutex, 307
- vendor_str, 306
- usb.h
 - _nk95cmd, 319
 - _resetusb, 320
 - _usbrecv, 321
 - _usbsend, 323
 - closeusb, 324
 - DELAY_LONG, 311
 - DELAY_MEDIUM, 311
 - DELAY_SHORT, 311
 - IS_FULLRANGE, 311
 - IS_K65, 311
 - IS_K70, 311
 - IS_K95, 311
 - IS_M65, 312
 - IS_MONOCHROME, 312
 - IS_MONOCHROME_DEV, 312
 - IS_MOUSE, 312
 - IS_MOUSE_DEV, 312
 - IS_RGB, 312
 - IS_RGB_DEV, 312
 - IS_SABRE, 312
 - IS_SCIMITAR, 312
 - IS_STRAFE, 313
 - NK95_HWOFF, 313
 - NK95_HWON, 313
 - NK95_M1, 313
 - NK95_M2, 313
 - NK95_M3, 313
 - nk95cmd, 313
 - os_closeusb, 326
 - os_inputmain, 327
 - os_resetusb, 330
 - os_sendindicators, 331
 - os_setupusb, 333
 - os_usbrecv, 335
 - os_usbsend, 337
 - P_K65, 313
 - P_K65_LUX, 314
 - P_K65_LUX_STR, 314
 - P_K65_NRGB, 314
 - P_K65_NRGB_STR, 314
 - P_K65_RFIRE, 314
 - P_K65_RFIRE_STR, 314
 - P_K65_STR, 314
 - P_K70, 314
 - P_K70_LUX, 314

- P_K70_LUX_NRGB, 314
- P_K70_LUX_NRGB_STR, 315
- P_K70_LUX_STR, 315
- P_K70_NRGB, 315
- P_K70_NRGB_STR, 315
- P_K70_RFIRE, 315
- P_K70_RFIRE_NRGB, 315
- P_K70_RFIRE_STR, 315
- P_K70_STR, 315
- P_K95, 315
- P_K95_NRGB, 315
- P_K95_NRGB_STR, 316
- P_K95_PLATINUM, 316
- P_K95_PLATINUM_STR, 316
- P_K95_STR, 316
- P_M65, 316
- P_M65_PRO, 316
- P_M65_PRO_STR, 316
- P_M65_STR, 316
- P_SABRE_L, 316
- P_SABRE_L_STR, 316
- P_SABRE_N, 316
- P_SABRE_N_STR, 317
- P_SABRE_O, 317
- P_SABRE_O2, 317
- P_SABRE_O2_STR, 317
- P_SABRE_O_STR, 317
- P_SCIMITAR, 317
- P_SCIMITAR_PRO, 317
- P_SCIMITAR_PRO_STR, 317
- P_SCIMITAR_STR, 317
- P_STRAFE, 317
- P_STRAFE_NRGB, 318
- P_STRAFE_NRGB_STR, 318
- P_STRAFE_STR, 318
- product_str, 339
- resetusb, 318
- revertusb, 340
- setupusb, 341
- USB_DELAY_DEFAULT, 318
- usb_tryreset, 342
- usbkill, 343
- usbmain, 344
- usbrecv, 318
- usbseend, 318
- V_CORSAIR, 319
- V_CORSAIR_STR, 319
- vendor_str, 345
- usb_add_device
 - usb_linux.c, 362
- usb_linux.c
 - _nk95cmd, 349
 - DEBUG, 348
 - kbsyspath, 371
 - models, 371
 - N_MODELS, 348
 - os_closeusb, 350
 - os_inputmain, 351
 - os_resetusb, 354
 - os_sendindicators, 355
 - os_setupusb, 356
 - os_usbrecv, 358
 - os_usbseend, 359
 - strtrim, 361
 - TEST_RESET, 348
 - udev, 372
 - udev_enum, 361
 - udevthread, 372
 - usb_add_device, 362
 - usb_rm_device, 365
 - usbadd, 366
 - usbclaim, 367
 - usbkill, 368
 - usbmain, 368
 - usbthread, 372
 - usbunclaim, 370
- usb_rm_device
 - usb_linux.c, 365
- usb_tryreset
 - usb.c, 305
 - usb.h, 342
- usbadd
 - usb_linux.c, 366
- usbclaim
 - usb_linux.c, 367
- usbdevice, 282
- usbid, 272
- usbinput, 281
- usbkill
 - usb.h, 343
 - usb_linux.c, 368
- usbmain
 - usb.h, 344
 - usb_linux.c, 368
- usbmode, 278
- usbmutex
 - usb.c, 307
- usbprofile, 279
- usbrecv
 - usb.h, 318
- usbseend
 - usb.h, 318
- usbthread
 - usb_linux.c, 372
- usbunclaim
 - usb_linux.c, 370
- ushort
 - includes.h, 111
- utf16to8
 - profile.c, 250
- utf8to16
 - profile.c, 250
- V_CORSAIR
 - usb.h, 319
- V_CORSAIR_STR
 - usb.h, 319

- vendor_str
 - usb.c, [306](#)
 - usb.h, [345](#)
- vtable_keyboard
 - device_vtable.c, [68](#)
 - structures.h, [288](#)
- vtable_keyboard_nonrgb
 - device_vtable.c, [68](#)
 - structures.h, [288](#)
- vtable_mouse
 - device_vtable.c, [68](#)
 - structures.h, [288](#)